DB2 Universal Database for z/OS

Version 8

Administration Guide

Information Management software

SC18-7413-05

DB2 Universal Database for z/OS

Version 8

Administration Guide

Information Management software

SC18-7413-05

Note

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 1301.

Sixth Edition, Softcopy Only (February 2008)

This edition applies to Version 8 of IBM DB2 Universal Database for z/OS (DB2 UDB for z/OS), product number 5625-DB2, and to any subsequent releases until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

This softcopy version is based on the printed edition of the book and includes the changes indicated in the printed version by vertical bars. Additional changes made to this softcopy version of the book since the hardcopy book was published are indicated by the hash (#) symbol in the left-hand margin. Editorial changes that have no technical significance are not noted.

This and other books in the DB2 UDB for z/OS library are periodically updated with technical changes. These updates are made available to licensees of the product on CD-ROM and on the Web (currently at www.ibm.com/software/data/db2/zos/library.html). Check these resources to ensure that you are using the most current information.

© Copyright International Business Machines Corporation 1982, 2008. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

About this book . . . xxv
Who should read this book. . . . xxv
Terminology and citations . . . xxv
How to read the syntax diagrams . . . xxvi
Accessibility . . . xxvii
How to send your comments . . . xxviii

Summary of changes to this book . . . xxix

Part 1. Introduction . . . 1

Chapter 1. System planning concepts . . . 3
The structure of DB2. . . . 3
Data structures . . . 3
System structures . . . 7
More information about data structures . . . 10
Control and maintenance of DB2 . . . 11
Commands . . . 11
Utilities . . . 11
High availability. . . . 12
More information about control and maintenance of DB2 . . . 13
The DB2 environment . . . 13
Address spaces . . . 13
DB2 lock manager . . . 14
DB2 attachment facilities . . . 14
DB2 and distributed data . . . 18
DB2 and z/OS . . . 19
DB2 and the Parallel Sysplex . . . 19
DB2 and the Security Server for z/OS . . . 19
DB2 and DFSMS. . . . 20
More information about the z/OS environment . . . 20

Part 2. Designing a database: advanced topics . . . 23

Chapter 2. Introduction to designing a database: advanced topics . . . 25

Chapter 3. Creating storage groups and managing DB2 data sets . . . 27
Managing data sets with DB2 storage groups . . . 27
Creating DB2 storage groups . . . 29
Using SMS to manage DB2 storage groups . . . 30
Deferring allocation of DB2-managed data sets . . . 30
Extending DB2-managed data sets. . . . 31
Managing DB2 data sets with DFSMShsm . . . 35
Migrating to DFSMShsm . . . 35
Recalling archive logs . . . 36
Using DFSMShsm with the RECOVER utility . . . 36
Using DFSMShsm with the BACKUP SYSTEM utility . . . 37
Managing your own data sets . . . 37
Requirements for your own data sets . . . 38
Using the DEFINE CLUSTER command . . . 40
Extending user-managed data sets. . . . 41
Defining index space storage . . . 41
Creating EA-enabled table spaces and index spaces . . . 42

© Copyright IBM Corp. 1982, 2008

Chapter 4. Implementing your database design . . . 43
Implementing databases . . . 43
Implementing table spaces . . . 44
Creating a table space explicitly . . . 44
Creating a table space implicitly . . . 45
Choosing a page size . . . 45
Choosing a page size for LOBs . . . 46
Implementing tables . . . 48
Distinctions between DB2 base tables and temporary tables . . . 48
Implementing table-controlled partitioning . . . 51
Implementing indexes . . . 54
Types of indexes . . . 54
Using the NOT PADDED clause for indexes with varying-length columns . . . 57
Using indexes to avoid sorts . . . 58
Using schemas . . . 58
Creating a schema . . . 58
Authorization to process schema definitions . . . 59
Processing schema definitions . . . 59

Chapter 5. Loading data into DB2 tables . . . 61
Loading tables with the LOAD utility . . . 61
Making corrections after LOAD . . . 63
Loading data using the SQL INSERT statement . . . 63
Inserting a single row . . . 64
Inserting multiple rows . . . 64
Special considerations when using INSERT statement to load tables . . . 64
Loading data from DL/I . . . 65

Chapter 6. Altering your database design . . . 67
Altering DB2 storage groups . . . 67
Letting SMS manage your DB2 storage groups . . . 67
Adding or removing volumes from a DB2 storage group . . . 67
Altering DB2 databases . . . 68
Altering table spaces . . . 69
Changing the space allocation for user-managed data sets . . . 69
Dropping, re-creating, or converting a table space . . . 69
Altering tables . . . 71
Adding a new column to a table . . . 71
Altering the data type of a column . . . 73
Altering a table for referential integrity . . . 77
Adding or dropping table check constraints . . . 80
Adding a partition . . . 80
Altering partitions . . . 82
Registering or changing materialized query tables . . . 85
Altering the assignment of a validation routine . . . 86
Altering a table for capture of changed data . . . 87
Changing an edit procedure or a field procedure . . . 87
Altering the subtype of a string column . . . 88
Altering the attributes of an identity column . . . 88
Changing data types by dropping and re-creating the table . . . 88
Moving a table to a table space of a different page size . . . 91
Altering indexes . . . 92
Adding a new column to an index . . . 92
Altering how varying-length index columns are stored . . . 93
Altering the clustering index . . . 93
Rebalancing data in partitioned table spaces . . . 93
Dropping and redefining an index . . . 94
Index versions . . . 94
Altering views . . . 96
Altering stored procedures . . . 96

Altering user-defined functions . . . 96
| Moving from index-controlled to table-controlled partitioning . . . 97
Changing the high-level qualifier for DB2 data sets . . . 98
Defining a new integrated catalog alias . . . 99
Changing the qualifier for system data sets . . . 99
Changing qualifiers for other databases and user data sets . . . 102
Moving DB2 data . . . 105
Tools for moving DB2 data . . . 106
Moving a DB2 data set . . . 108
Copying a relational database . . . 109
Copying an entire DB2 subsystem . . . 109

Chapter 7. Estimating disk storage for user data . . . 111
Factors that affect storage . . . 111
Calculating the space required for a table . . . 112
Calculating record lengths and pages . . . 113
Saving space with data compression . . . 114
Estimating storage for LOBs . . . 114
Estimating storage when using the LOAD utility . . . 114
Calculating the space required for a dictionary . . . 116
Disk requirements . . . 116
Virtual storage requirements . . . 117
Calculating the space required for an index . . . 117
Levels of index pages . . . 117
Estimating storage from number of index pages . . . 118

Part 3. Security and auditing . . . 123

Chapter 8. Introduction to security and auditing in DB2 . . . 127
Reading strategies for security . . . 127
If you are new to DB2 . . . 127
| If you have used DB2 before . . . 128
Reading strategies for auditing . . . 128
Controlling data access . . . 128
Access control within DB2 . . . 129
Controlling access to a DB2 subsystem . . . 131
Data set protection . . . 132

Chapter 9. Controlling access to DB2 objects . . . 133
Explicit privileges and authorities . . . 133
Authorization IDs . . . 134
Granting explicit privileges . . . 134
Administrative authorities . . . 138
Field-level access control by views . . . 143
Authority over the catalog and directory . . . 144
Implicit privileges of ownership . . . 145
Establishing ownership of objects with unqualified names . . . 145
Establishing ownership of objects with qualified names . . . 146
Privileges by type of object . . . 146
Granting implicit privileges . . . 147
Changing ownership . . . 148
Privileges exercised through a plan or a package . . . 148
Establishing or changing ownership of a plan or a package . . . 148
Qualifying unqualified names . . . 149
Authorization to execute . . . 149
Controls in the program . . . 153
Privileges required for remote packages . . . 154
Access control for user-defined functions and stored procedures . . . 155
Additional authorization for stored procedures . . . 156
Controlling access to catalog tables for stored procedures . . . 156

Example of roles and authorizations for a routine . . . 157
Which IDs can exercise which privileges . . . 161
Authorization for dynamic SQL statements . . . 164
Composite privileges . . . 171
Multiple actions in one statement. . . . 171
Matching job titles with privileges . . . 171
Examples of granting and revoking privileges . . . 173
Examples using the GRANT statement . . . 174
Examples with secondary IDs . . . 176
Examples using the REVOKE statement . . . 180
Finding catalog information about privileges . . . 187
Retrieving information in the catalog . . . 187
Creating views of the DB2 catalog tables . . . 191
| Multilevel security. . . . 192
| Introduction to multilevel security . . . 192
| Implementing multilevel security with DB2 . . . 195
| Working with data in a multilevel-secure environment . . . 199
| Implementing multilevel security in a distributed environment . . . 206
| Data encryption through built-in functions . . . 207
| Defining columns for encrypted data . . . 208
| Defining encryption at the column level . . . 208
| Defining encryption at the value level . . . 210
| Ensuring accurate predicate evaluation for encrypted data . . . 211
| Encrypting non-character values . . . 211
Performance recommendations for data encryption . . . 212

Chapter 10. Controlling access through a closed application . . . 215
Registration tables . . . 216
Columns of the ART . . . 216
Columns of the ORT . . . 217
Controlling data definition . . . 218
Installing data definition control support . . . 219
Controlling data definition by application name . . . 220
Controlling data definition by application name with exceptions . . . 221
Controlling data definition by object name . . . 222
Controlling data definition by object name with exceptions . . . 224
Registering sets of objects . . . 225
Managing the registration tables and their indexes . . . 226
Creating the tables and indexes . . . 226
Adding columns . . . 228
Updating the tables . . . 228
Stopping data definition control . . . 228

Chapter 11. Controlling access to a DB2 subsystem . . . 231
Controlling local requests . . . 232
Processing connections . . . 232
Steps in processing connections . . . 233
Supplying secondary IDs for connection requests . . . 234
Required CICS specifications . . . 235
Processing sign-ons . . . 236
Steps in processing sign-ons . . . 236
Supplying secondary IDs for sign-on requests . . . 237
Controlling requests from remote applications . . . 238
Overview of security mechanisms for DRDA and SNA . . . 238
The communications database for the server . . . 240
Controlling inbound connections that use SNA protocols . . . 242
Controlling inbound connections that use TCP/IP protocols . . . 249
Planning to send remote requests. . . . 252
The communications database for the requester . . . 253
What IDs you send . . . 257

Translating outbound IDs . . . 260
Sending passwords . . . 262
Establishing RACF protection for DB2 . . . 264
Defining DB2 resources to RACF . . . 265
Permitting RACF access . . . 267
| Issuing DB2 commands . . . 274
Establishing RACF protection for stored procedures . . . 274
Establishing RACF protection for TCP/IP . . . 278
Establishing Kerberos authentication through RACF . . . 278
Other methods of controlling access . . . 279

Chapter 12. Protecting data sets through RACF . . . 281
Adding groups to control DB2 data sets . . . 281
Creating generic profiles for data sets . . . 281
Permitting DB2 authorization IDs to use the profiles . . . 283
Allowing DB2 authorization IDs to create data sets . . . 283

Chapter 13. Auditing . . . 285
Using the audit trace . . . 285
Starting the audit trace . . . 286
Stopping the audit trace . . . 286
Audit class descriptions . . . 287
Limitations of the audit trace . . . 287
Auditing in a distributed data environment . . . 288
Auditing a specific table . . . 288
The role of authorization IDs in auditing . . . 289
Auditing specific IDs . . . 290
Determining which IDs hold privileges and authorities . . . 290
Using audit records . . . 290
The contents of audit records . . . 291
Formatting audit records . . . 291
Suggested audit trace reports . . . 291
Using other sources of audit information . . . 292
Determining which security measures are enabled . . . 292
Ensuring data accuracy and consistency . . . 293
Ensuring that the required data is present . . . 293
Ensuring that data is unique . . . 293
Ensuring that data fits a pattern or value range . . . 293
Ensuring that data is consistent . . . 294
Ensuring that changes are tracked . . . 295
Ensuring that concurrent users access consistent data . . . 295
Checking for lost and incomplete transactions . . . 296
Determining whether data is consistent . . . 296
Automatically checking the consistency of data . . . 297
Submitting SQL queries to check data consistency . . . 297
Checking data consistency with the CHECK utility . . . 297
Checking data consistency with the DISPLAY DATABASE command . . . 298
Checking data consistency with the REPORT utility . . . 298
Checking data consistency with the operation log . . . 298
Using internal integrity reports to check data consistency . . . 298

Chapter 14. A sample security plan for employee data . . . 301
Securing manager access . . . 301
Granting the SELECT privilege to managers . . . 302
Securing distributed access . . . 303
Auditing manager use . . . 304
Securing payroll operations access . . . 305
Securing compensation updates . . . 305
Additional controls for compensation updates . . . 306
Granting privileges to payroll operations and payroll management . . . 307

Auditing payroll operations and payroll management . . . 307
Securing administrator, owner, and other access . . . 308
Securing access by IDs with database administrator authority . . . 308
Securing access by IDs with system administrator authority . . . 308
Securing access by owners with implicit privileges on objects . . . 309
Securing access by other users . . . 310

Part 4. Operation and recovery . . . 311

Chapter 15. Basic operation . . . 317
Entering commands . . . 317
DB2 operator commands . . . 318
Authorities for DB2 commands . . . 323
Starting and stopping DB2 . . . 324
Starting DB2 . . . 324
Stopping DB2 . . . 326
Submitting work . . . 327
Using DB2I (DB2 Interactive) . . . 327
Running TSO application programs . . . 327
Running IMS application programs . . . 328
Running CICS application programs . . . 329
Running batch application programs . . . 330
Running application programs using CAF . . . 330
Running application programs using RRSAF . . . 331
Receiving messages . . . 331
Receiving unsolicited DB2 messages . . . 332
Determining operational control . . . 332

Chapter 16. Scheduling administrative tasks . . . 335
Interacting with the administrative task scheduler . . . 335
Adding a task . . . 335
Listing scheduled tasks . . . 347
Listing the last execution status of scheduled tasks . . . 347
Removing a scheduled task . . . 348
Manually starting the administrative task scheduler . . . 351
Manually stopping the administrative task scheduler . . . 351
Synchronization between administrative task schedulers in a data sharing environment . . . 351
Troubleshooting the administrative task scheduler . . . 352
Architecture of the administrative task scheduler . . . 355
The life cycle of the administrative task scheduler . . . 356
Scheduler task lists . . . 356
Architecture of the administrative task scheduler in a data sharing environment . . . 356
Security guidelines for the administrative task scheduler . . . 357
User roles in the administrative task scheduler . . . 357
Protection of the interface of the administrative task scheduler . . . 357
Protection of the resources of the administrative task scheduler . . . 358
Secure execution of tasks in the administrative task scheduler . . . 358
Execution of scheduled tasks in the administrative task scheduler . . . 359
Multi-threading in the administrative task scheduler . . . 359
Scheduled execution of a stored procedure . . . 360
How the administrative task scheduler works with Unicode . . . 360
Scheduled execution of a JCL job . . . 361
Execution of scheduled tasks in a data sharing environment . . . 361

Chapter 17. Monitoring and controlling DB2 and its connections . . . 363
Controlling DB2 databases and buffer pools . . . 363
Starting databases . . . 364
Monitoring databases . . . 365
Stopping databases . . . 371
Altering buffer pools . . . 372

Monitoring buffer pools . . . 373
Controlling user-defined functions . . . 373
Starting user-defined functions . . . 373
Monitoring user-defined functions . . . 374
Stopping user-defined functions . . . 374
Controlling DB2 utilities . . . 375
Starting online utilities . . . 375
Monitoring online utilities . . . 375
Stand-alone utilities . . . 376
Controlling the IRLM . . . 376
Starting the IRLM . . . 377
Modifying the IRLM . . . 377
Monitoring the IRLM connection . . . 378
Stopping the IRLM . . . 378
Monitoring threads . . . 379
Controlling TSO connections . . . 380
Connecting to DB2 from TSO . . . 381
Monitoring TSO and CAF connections . . . 381
Disconnecting from DB2 while under TSO . . . 383
Controlling CICS connections . . . 383
Connecting from CICS . . . 384
Controlling CICS connections . . . 385
Disconnecting from CICS . . . 387
Controlling IMS connections . . . 388
Connecting to the IMS control region . . . 388
Controlling IMS dependent region connections . . . 393
Disconnecting from IMS . . . 396
Controlling RRS connections . . . 397
Connecting to RRS using RRSAF . . . 397
Monitoring RRSAF connections . . . 399
Controlling connections to remote systems . . . 400
Starting the DDF . . . 401
Suspending and resuming DDF server activity . . . 401
Monitoring connections to other systems . . . 402
Monitoring and controlling stored procedures . . . 413
Using NetView to monitor errors . . . 416
Stopping the DDF . . . 417
Controlling traces . . . 418
Controlling the DB2 trace . . . 419
Diagnostic traces for attachment facilities . . . 420
Diagnostic trace for the IRLM . . . 420
Controlling the resource limit facility (governor) . . . 421
Changing subsystem parameter values . . . 421

Chapter 18. Managing the log and the bootstrap data set . . . 423
How database changes are made . . . 423
Units of recovery . . . 423
Rolling back work . . . 424
Establishing the logging environment . . . 425
Creation of log records . . . 425
Retrieval of log records . . . 425
Writing the active log . . . 425
Writing the archive log (offloading) . . . 426
Controlling the log . . . 430
Archiving the log . . . 430
Dynamically changing the checkpoint frequency . . . 433
| Monitoring the system checkpoint . . . 433
Setting limits for archive log tape units . . . 433
Displaying log information . . . 434
Resetting the log RBA . . . 434
Log RBA range . . . 434

Resetting the log RBA value in a data sharing environment . . . 435
Resetting the log RBA value in a non-data sharing environment . . . 435
Managing the bootstrap data set (BSDS) . . . 436
BSDS copies with archive log data sets . . . 437
Changing the BSDS log inventory . . . 438
Discarding archive log records. . . . 438
Deleting archive logs automatically . . . 438
Locating archive log data sets . . . 439

Chapter 19. Restarting DB2 after termination . . . 443
Termination . . . 443
Normal termination . . . 443
Abends . . . 444
Normal restart and recovery . . . 444
Phase 1: Log initialization . . . 445
Phase 2: Current status rebuild . . . 446
Phase 3: Forward log recovery. . . . 447
Phase 4: Backward log recovery . . . 448
Restarting automatically . . . 449
Deferring restart processing . . . 450
Restarting with conditions . . . 451
Resolving postponed units of recovery . . . 452
Choosing recovery operations for conditional restart . . . 453
Conditional restart records . . . 454

Chapter 20. Maintaining consistency across multiple systems . . . 455
Multiple system consistency . . . 455
The two-phase commit process . . . 455
Illustration of two-phase commit . . . 456
Maintaining consistency after termination or failure . . . 457
Termination for multiple systems. . . . 458
Normal restart and recovery for multiple systems . . . 458
Restarting multiple systems with conditions . . . 459
Resolving indoubt units of recovery . . . 459
Resolution of IMS indoubt units of recovery . . . 460
Resolution of CICS indoubt units of recovery. . . . 461
| Resolution of WebSphere Application Server indoubt units of recovery . . . 461
Resolution of remote DBMS indoubt units of recovery. . . . 463
Resolution of RRS indoubt units of recovery . . . 466
Consistency across more than two systems . . . 467
Commit coordinator and multiple participants . . . 467
Illustration of multi-site update . . . 468

Chapter 21. Backing up and recovering databases . . . 471
Planning for backup and recovery . . . 471
Considerations for recovering distributed data . . . 472
Extended recovery facility (XRF) toleration . . . 472
Considerations for recovering indexes . . . 473
Preparing for recovery . . . 473
Events that occur during recovery . . . 474
Maximizing data availability during backup and recovery . . . 477
How to find recovery information . . . 480
Preparing to recover to a prior point of consistency . . . 481
Preparing to recover the entire DB2 subsystem to a prior point in time . . . 482
Preparing for disaster recovery . . . 483
Ensuring more effective recovery from inconsistency . . . 486
Running the RECOVER utility in parallel . . . 488
Using fast log apply during RECOVER. . . . 489
Reading the log without RECOVER . . . 489
Copying page sets and data sets . . . 489

Backing up with DFSMS . . . 491
Backing up with RVA storage control or Enterprise Storage Server. . . . 491
Recovering page sets and data sets . . . 491
Recovering the work file database . . . 493
Problems with the user-defined work file data set . . . 493
Problems with DB2-managed work file data sets . . . 493
Recovering error ranges for a work file table space . . . 493
Recovering the catalog and directory . . . 494
Recovering data to a prior point of consistency . . . 495
Considerations for recovering to a prior point of consistency . . . 495
Using RECOVER to restore data to a previous point in time. . . . 500
Recovery of dropped objects . . . 505
Discarding SYSCOPY and SYSLGRNX records . . . 510
System-level point in time recovery . . . 511

Chapter 22. Recovery scenarios . . . 517
IRLM failure recovery . . . 517
z/OS or power failure recovery . . . 518
Disk failure recovery . . . 518
Application error recovery . . . 520
IMS-related failure recovery . . . 521
IMS control region (CTL) failure recovery . . . 522
Resolution of indoubt units of recovery. . . . 522
IMS application failure recovery . . . 524
CICS-related failure recovery . . . 525
CICS application failure recovery. . . . 525
Recovery when CICS is not operational . . . 525
Recovery when CICS cannot connect to DB2 . . . 526
Manually recovering CICS indoubt units of recovery . . . 527
CICS attachment facility failure recovery . . . 530
Subsystem termination recovery . . . 530
Resource failure recovery . . . 531
Active log failure recovery . . . 531
Archive log failure recovery . . . 535
Temporary resource failure recovery. . . . 537
BSDS failure recovery . . . 538
Recovering the BSDS from a backup copy . . . 539
DB2 database failures. . . . 542
Recovery from down-level page sets. . . . 543
Procedure for recovering invalid LOBs . . . 544
Table space input/output error recovery . . . 545
DB2 catalog or directory input/output errors. . . . 546
Integrated catalog facility failure recovery . . . 547
Recovery when the VSAM volume data set (VVDS) is destroyed . . . 547
Out of disk space or extent limit recovery . . . 548
Violation of referential constraint recovery. . . . 552
Distributed data facility failure recovery . . . 553
Conversation failure recovery . . . 553
Communications database failure recovery . . . 554
Database access thread failure recovery. . . . 555
VTAM failure recovery . . . 555
TCP/IP failure recovery . . . 556
Remote logical unit failure recovery . . . 556
Indefinite wait condition recovery . . . 557
Security failure recovery for database access threads . . . 557
Remote site recovery from a disaster at the local site . . . 558
Restoring data from image copies and archive logs. . . . 558
Using a tracker site for disaster recovery . . . 568
Using data mirroring . . . 576
Resolving indoubt threads . . . 582
Description of the recovery environment . . . 582

Communication failure recovery . . . 583
Making a heuristic decision . . . 584
Recovery from an IMS outage that results in an IMS cold start . . . 585
Recovery from a DB2 outage at a requester that results in a DB2 cold start . . . 586
Recovery from a DB2 outage at a server that results in a DB2 cold start . . . 589
Correcting a heuristic decision. . . . 589

Chapter 23. Recovery from BSDS or log failure during restart . . . 591
Log initialization or current status rebuild failure recovery . . . 593
Description of failure during log initialization . . . 594
Description of failure during current status rebuild . . . 595
Restart by truncating the log . . . 595
Failure during forward log recovery . . . 602
Understanding forward log recovery failure . . . 603
Starting DB2 by limiting restart processing . . . 603
Failure during backward log recovery . . . 607
Understanding backward log recovery failure . . . 608
Bypassing backout before restarting . . . 608
Failure during a log RBA read request . . . 609
Unresolvable BSDS or log data set problem during restart . . . 610
Preparing for recovery or restart . . . 611
Performing fall back to a prior shutdown point . . . 611
Failure resulting from total or excessive loss of log data . . . 613
Total loss of the log . . . 613
Excessive loss of data in the active log . . . 614
Resolving inconsistencies resulting from a conditional restart . . . 616
Inconsistencies in a distributed environment . . . 616
Procedures for resolving inconsistencies . . . 616
Method 1. Recover to a prior point of consistency . . . 617
Method 2. Re-create the table space . . . 618
Method 3. Use the REPAIR utility on the data . . . 618

Part 5. Performance monitoring and tuning . . . 621

Chapter 24. Planning your performance strategy . . . 627
Managing performance in general . . . 627
Setting reasonable performance objectives . . . 628
Defining the workload . . . 628
Initial performance planning . . . 629
Post-development review . . . 631
Planning for performance monitoring . . . 631
Continuous performance monitoring . . . 632
Periodic performance monitoring . . . 633
Detailed performance monitoring. . . . 633
Exception performance monitoring . . . 634
A performance monitoring strategy . . . 634
Reviewing performance data . . . 634
Typical review questions . . . 635
Are your performance objectives reasonable? . . . 636

Chapter 25. Analyzing performance data. . . . 639
Investigating the problem overall. . . . 639
Looking at the entire system . . . 639
Beginning to look at DB2 . . . 639
Reading accounting reports from OMEGAMON. . . . 640
The accounting report (short format) . . . 640
The accounting report (long format) . . . 641
A general approach to problem analysis in DB2 . . . 646

Chapter 26. Improving response time and throughput . . . 651
Reducing I/O operations . . . 651
Using RUNSTATS to keep access path statistics current . . . 651
Reserving free space in table spaces and indexes . . . 652
Making buffer pools large enough for the workload . . . 654
Reducing the time needed to perform I/O operations . . . 654
Distributing data sets efficiently . . . 654
Creating additional work file table spaces . . . 657
Managing space for I/O performance . . . 658
Reducing processor resource consumption . . . 659
Reusing threads for your high-volume transactions . . . 660
Minimizing the use of DB2 traces . . . 660
Using fixed-length records . . . 661
Response time reporting . . . 661

Chapter 27. Tuning DB2 buffer, EDM, RID, and sort pools . . . 665
Tuning database buffer pools . . . 665
Terminology: Types of buffer pool pages . . . 666
Read operations . . . 666
Write operations . . . 666
Assigning a table space or index to a buffer pool . . . 667
Buffer pool thresholds . . . 667
Determining size and number of buffer pools . . . 671
Choosing a page-stealing algorithm . . . 674
Long-term page fix option for buffer pools . . . 675
Monitoring and tuning buffer pools using online commands . . . 675
Using OMEGAMON to monitor buffer pool statistics . . . 677
Tuning EDM storage . . . 679
EDM storage space handling . . . 680
Tips for managing EDM storage . . . 682
Increasing RID pool size . . . 683
Controlling sort pool size and sort processing . . . 684
Estimating the maximum size of the sort pool . . . 684
How sort work files are allocated . . . 684
Improving the performance of sort processing . . . 685

Chapter 28. Improving resource utilization . . . 687
Managing the opening and closing of data sets . . . 687
Determining the maximum number of open data sets . . . 687
Understanding the CLOSE YES and CLOSE NO options . . . 690
Switching to read-only for infrequently updated and infrequently accessed page sets . . . 691
Planning the placement of DB2 data sets . . . 692
Estimating concurrent I/O requests . . . 692
Crucial DB2 data sets . . . 692
Changing catalog and directory size and location . . . 693
Monitoring I/O activity of data sets . . . 693
Work file data sets . . . 693
DB2 logging . . . 694
Logging performance issues and recommendations . . . 694
Log capacity . . . 697
Controlling the amount of log data . . . 699
Improving disk utilization: space and device utilization . . . 701
Allocating and extending data sets . . . 701
Compressing your data . . . 702
Evaluating your indexes . . . 705
Improving real storage utilization . . . 706
Performance and storage . . . 708
Real storage . . . 708
Storage devices . . . 709
z/OS performance options for DB2 . . . 711

Determining z/OS workload management velocity goals . . . 711
How DB2 assigns I/O priorities . . . 712
IBM System z9 Integrated Information Processor usage monitoring . . . 713
Controlling resource usage . . . 714
Prioritizing resources . . . 715
Limiting resources for each job . . . 715
Limiting resources for TSO sessions . . . 715
Limiting resources for IMS and CICS . . . 716
Limiting resources for a stored procedure . . . 716
Resource limit facility (governor) . . . 716
Using resource limit tables (RLSTs) . . . 717
Governing dynamic queries . . . 722
Restricting bind operations . . . 726
Restricting parallelism modes . . . 727

Chapter 29. Managing DB2 threads . . . 729
Setting thread limits . . . 729
Allied thread allocation . . . 730
Step 1: Thread creation . . . 730
Step 2: Resource allocation . . . 731
Step 3: SQL statement execution . . . 731
Step 4: Commit and thread termination. . . . 732
Variations on thread management . . . 733
Providing for thread reuse . . . 733
Database access threads . . . 735
Using allied threads and database access threads . . . 735
| Setting thread limits for database access threads. . . . 735
| Using threads in INACTIVE MODE for DRDA-only connections . . . 736
| Using threads with private-protocol connections. . . . 738
Reusing threads for remote connections . . . 739
Using z/OS workload management to set performance objectives . . . 739
CICS design options . . . 742
IMS design options . . . 742
TSO design options . . . 743
DB2 QMF design options . . . 744

Chapter 30. Tuning your queries . . . 745
General tips and questions . . . 745
Is the query coded as simply as possible? . . . 745
Are all predicates coded correctly? . . . 745
Are there subqueries in your query?. . . . 746
Does your query involve aggregate functions? . . . 747
Do you have an input variable in the predicate of an SQL query? . . . 748
Do you have a problem with column correlation? . . . 748
Can your query be written to use a noncolumn expression? . . . 748
| Can materialized query tables help your query performance? . . . 748
| Does the query contain encrypted data? . . . 749
Writing efficient predicates . . . 749
Properties of predicates . . . 749
Predicates in the ON clause . . . 752
General rules about predicate evaluation . . . 753
Order of evaluating predicates. . . . 753
Summary of predicate processing. . . . 754
Examples of predicate properties . . . 759
Predicate filter factors . . . 760
Column correlation . . . 766
DB2 predicate manipulation . . . 770
| Predicates with encrypted data . . . 773
Using host variables efficiently . . . 774
| Changing the access path at run time . . . 774

Rewriting queries to influence access path selection . . . 777
Writing efficient subqueries. . . . 780
Correlated subqueries . . . 780
Noncorrelated subqueries . . . 781
Conditions for DB2 to transform a subquery into a join . . . 782
Subquery tuning . . . 784
Using scrollable cursors efficiently . . . 785
| Writing efficient queries on tables with data-partitioned secondary indexes. . . . 786
Special techniques to influence access path selection . . . 788
Obtaining information about access paths . . . 789
Fetching a limited number of rows: FETCH FIRST *n* ROWS ONLY . . . 789
Minimizing overhead for retrieving few rows: OPTIMIZE FOR *n* ROWS . . . 790
| Favoring index access . . . 792
Using a subsystem parameter to control outer join processing . . . 793
| Using the CARDINALITY clause to improve the performance of queries with user-defined table function
| references . . . 793
Reducing the number of matching columns . . . 794
Creating indexes for efficient star-join processing . . . 796
Rearranging the order of tables in a FROM clause . . . 798
Updating catalog statistics . . . 799
Using a subsystem parameter . . . 800
Giving optimization hints to DB2. . . . 801

Chapter 31. Improving concurrency . . . 809
Definitions of concurrency and locks . . . 810
Effects of DB2 locks . . . 810
Suspension . . . 810
Timeout . . . 811
Deadlock . . . 811
Basic recommendations to promote concurrency. . . . 812
Recommendations for system and subsystem options . . . 812
Recommendations for database design . . . 813
Recommendations for application design . . . 814
Aspects of transaction locks . . . 817
The size of a lock . . . 817
The duration of a lock . . . 820
The mode of a lock . . . 821
The object of a lock . . . 823
DB2 choices of lock types . . . 826
Options for tuning locks. . . . 832
IRLM startup procedure options . . . 832
Installation options for wait times . . . 833
Other options that affect locking . . . 837
Bind options. . . . 843
Isolation overriding with SQL statements . . . 857
The LOCK TABLE statement . . . 859
LOB locks . . . 860
Relationship between transaction locks and LOB locks . . . 860
Hierarchy of LOB locks . . . 861
LOB and LOB table space lock modes . . . 861
LOB lock and LOB table space lock duration . . . 862
Instances when LOB table space locks are not taken . . . 863
Control of the number of LOB locks. . . . 863
The LOCK TABLE statement for LOBs . . . 864
LOCKSIZE clause for LOB table spaces. . . . 864
Claims and drains for concurrency control . . . 864
Objects that are subject to takeover . . . 864
Claims . . . 865
Drains. . . . 865
Usage of drain locks . . . 866
Utility locks on the catalog and directory . . . 866

Compatibility of utilities . . . 867
Concurrency during REORG . . . 868
Utility operations with nonpartitioned indexes . . . 869
Monitoring of DB2 locking . . . 869
Using EXPLAIN to tell which locks DB2 chooses . . . 870
Using the statistics and accounting traces to monitor locking . . . 871
Scenario for analyzing concurrency . . . 872
Deadlock detection scenarios . . . 877
Scenario 1: Two-way deadlock with two resources . . . 877
Scenario 2: Three-way deadlock with three resources . . . 879

Chapter 32. Using materialized query tables . . . 881
Introduction to materialized query tables and automatic query rewrite . . . 881
Example of automatic query rewrite using a materialized query table . . . 882
Steps for using automatic query rewrite . . . 882
Defining a materialized query table . . . 883
Creating a new materialized query table . . . 883
Registering an existing table as a materialized query table . . . 884
Altering a materialized query table . . . 886
Populating and maintaining materialized query tables . . . 886
Using the REFRESH TABLE statement . . . 887
Using INSERT, UPDATE, DELETE, and LOAD for materialized query tables . . . 887
Collecting statistics on materialized query tables . . . 888
Using multilevel security with materialized query tables . . . 888
Creating a materialized query table . . . 888
Altering a source table . . . 889
Refreshing a materialized query table . . . 889
Exploiting automatic query rewrite . . . 889
Making materialized query tables eligible . . . 889
Query requirements and the rewrite process . . . 891
Recommendations for materialized query table and base table design . . . 897
Materialized query table examples shipped with DB2 . . . 898

Chapter 33. Maintaining statistics in the catalog . . . 899
Statistics used for access path selection . . . 899
Filter factors and catalog statistics . . . 906
Statistics for partitioned table spaces . . . 908
Setting default statistics for created temporary tables . . . 908
History statistics . . . 909
Gathering monitor statistics and update statistics . . . 912
Updating the catalog . . . 913
Correlations in the catalog . . . 913
Recommendation for COLCARDF and FIRSTKEYCARDF . . . 914
Recommendation for HIGH2KEY and LOW2KEY . . . 915
Statistics for distributions . . . 915
Recommendation for using the TIMESTAMP column . . . 915
Querying the catalog for statistics . . . 916
Improving index and table space access . . . 916
How clustering affects access path selection . . . 917
What other statistics provide index costs . . . 919
When to reorganize indexes and table spaces . . . 920
Whether to rebind after gathering statistics . . . 923
Modeling your production system . . . 923

Chapter 34. Using EXPLAIN to improve SQL performance . . . 927
Obtaining PLAN_TABLE information from EXPLAIN . . . 928
Creating PLAN_TABLE . . . 929
Populating and maintaining a plan table . . . 936
Reordering rows from a plan table . . . 939
Asking questions about data access . . . 939

Is access through an index? (ACCESSTYPE is I, I1, N or MX) . . . 940
Is access through more than one index? (ACCESSTYPE=M) . . . 940
How many columns of the index are used in matching? (MATCHCOLS=n) . . . 941
Is the query satisfied using only the index? (INDEXONLY=Y) . . . 942
Is direct row access possible? (PRIMARY_ACCESSTYPE = D) . . . 942
Is a view or nested table expression materialized? . . . 944
Was a scan limited to certain partitions? (PAGE_RANGE=Y) . . . 944
What kind of prefetching is expected? (PREFETCH = L, S, D, or blank) . . . 945
Is data accessed or processed in parallel? (PARALLELISM_MODE is I, C, or X) . . . 945
Are sorts performed? . . . 945
Is a subquery transformed into a join? . . . 946
When are aggregate functions evaluated? (COLUMN_FN_EVAL) . . . 946
How many index screening columns are used? . . . 946
Is a complex trigger WHEN clause used? (QBLOCKTYPE=TRIGGR) . . . 946
Interpreting access to a single table . . . 947
Table space scans (ACCESSTYPE=R PREFETCH=S) . . . 947
Overview of index access . . . 948
Index access paths . . . 950
UPDATE using an index . . . 954
Interpreting access to two or more tables (join) . . . 955
Definitions and examples of join operations . . . 955
Nested loop join (METHOD=1) . . . 957
Merge scan join (METHOD=2) . . . 959
Hybrid join (METHOD=4) . . . 961
Star join (JOIN_TYPE='S') . . . 963
Interpreting data prefetch . . . 969
Sequential prefetch (PREFETCH=S) . . . 969
Dynamic prefetch (PREFETCH=D) . . . 970
List prefetch (PREFETCH=L) . . . 970
Sequential detection at execution time . . . 972
Determining sort activity . . . 973
Sorts of data . . . 973
Sorts of RIDs . . . 974
The effect of sorts on OPEN CURSOR . . . 974
Processing for views and nested table expressions . . . 975
Merge . . . 975
Materialization . . . 976
Using EXPLAIN to determine when materialization occurs . . . 978
Using EXPLAIN to determine UNION activity and query rewrite . . . 979
Performance of merge versus materialization . . . 981
Estimating a statement's cost . . . 981
Creating a statement table . . . 982
Populating and maintaining a statement table . . . 984
Retrieving rows from a statement table . . . 984
The implications of cost categories . . . 985

Chapter 35. Parallel operations and query performance . . . 987
Comparing the methods of parallelism . . . 988
Partitioning for optimal parallel performance . . . 990
Determining if a query is I/O- or processor-intensive . . . 991
Determining the number of partitions . . . 991
Working with a table space that is already partitioned . . . 992
Making the partitions the same size . . . 992
Working with partitioned indexes . . . 993
Enabling parallel processing . . . 993
When parallelism is not used . . . 994
Interpreting EXPLAIN output . . . 995
A method for examining PLAN_TABLE columns for parallelism . . . 995
PLAN_TABLE examples showing parallelism . . . 995
Monitoring parallel operations . . . 997
Using DISPLAY BUFFERPOOL . . . 998

Using DISPLAY THREAD . . . 998
Using DB2 trace . . . 998
Tuning parallel processing . . . 1000
Disabling query parallelism . . . 1001

Chapter 36. Tuning and monitoring in a distributed environment . . . 1003
Remote access types: DRDA and private protocol . . . 1003
Characteristics of DRDA . . . 1003
Characteristics of DB2 private protocol . . . 1003
Tuning distributed applications . . . 1004
Application and requesting systems . . . 1004
Serving system . . . 1012
Monitoring DB2 in a distributed environment . . . 1013
The DISPLAY command . . . 1013
Tracing distributed events . . . 1013
Reporting server-elapsed time . . . 1017
Monitoring distributed processing with RMF . . . 1017
Duration of an enclave . . . 1017
RMF records for enclaves . . . 1018

Chapter 37. Monitoring and tuning stored procedures and user-defined functions 1019
Controlling address space storage . . . 1019
Assigning procedures and functions to WLM application environments . . . 1020
Providing DB2 cost information for accessing user-defined table functions . . . 1021
Monitoring stored procedures with the accounting trace . . . 1022
Accounting for nested activities . . . 1024
Comparing the types of stored procedure address spaces . . . 1025

Part 6. Appendixes . . . 1029

Appendix A. DB2 sample tables . . . 1031
Activity table (DSN8810.ACT) . . . 1031
Department table (DSN8810.DEPT) . . . 1032
Employee table (DSN8810.EMP) . . . 1034
Employee photo and resume table (DSN8810.EMP_PHOTO_RESUME) . . . 1037
Project table (DSN8810.PROJ) . . . 1038
Project activity table (DSN8810.PROJACT) . . . 1039
Employee to project activity table (DSN8810.EMPPROJACT) . . . 1040
Unicode sample table (DSN8810.DEMO_UNICODE) . . . 1041
Relationships among the sample tables . . . 1042
Views on the sample tables . . . 1042
Storage of sample application tables . . . 1047
Storage group . . . 1048
Databases . . . 1048
Table spaces . . . 1048

Appendix B. Writing exit routines . . . 1051
Connection routines and sign-on routines . . . 1051
Specifying connection and sign-on routines . . . 1052
Sample connection and sign-on routines . . . 1052
When connection and sign-on routines are taken . . . 1053
EXPL for connection and sign-on routines . . . 1053
Exit parameter list for connection and sign-on routines . . . 1054
Authorization ID parameter list for connection and sign-on routines . . . 1054
Input values for connection routines . . . 1055
Input values for sign-on routines . . . 1056
Expected output for connection and sign-on routines . . . 1056
Processing in the sample connection and sign-on routines . . . 1057
Performance considerations for connection and sign-on routines . . . 1058

Debugging connection and sign-on routines . . . 1059
Session variables in connection and sign-on routines . . . 1060
Access control authorization exit routine . . . 1061
Specifying the access control authorization routine . . . 1062
The default access control authorization routine . . . 1063
When the access control authorization routine is taken . . . 1063
Considerations for the access control authorization routine . . . 1063
Parameter list for the access control authorization routine . . . 1066
Expected output for the access control authorization routine . . . 1073
Debugging the access control authorization routine . . . 1076
Determining whether the access control authorization routine is active . . . 1076
Edit routines . . . 1077
Specifying the edit routine routine . . . 1078
When the edit routine is taken . . . 1078
Parameter lists for the edit routine . . . 1078
Processing requirements for edit routines . . . 1079
Incomplete rows and edit routines . . . 1079
Expected output for edit routines . . . 1080
Validation routines . . . 1080
Specifying the validation routine . . . 1081
When the validation routine is taken . . . 1081
Parameter lists for the validation routine . . . 1081
Processing requirements for validation routines . . . 1082
Incomplete rows and validation routines . . . 1082
Expected output for validation routines . . . 1082
Date and time routines . . . 1083
Specifying the date and time routine . . . 1084
When date and time routines are taken . . . 1084
Parameter lists for date and time routines . . . 1085
Expected output for date and time routines . . . 1085
Conversion procedures . . . 1086
Specifying a conversion procedure . . . 1086
When conversion procedures are taken . . . 1087
Parameter lists for conversion procedures . . . 1087
Expected output for conversion procedures . . . 1088
Field procedures . . . 1089
Field definition for field procedures . . . 1090
Specifying the field procedure . . . 1090
When field procedures are taken . . . 1091
Control blocks for execution of field procedures . . . 1092
Field-definition (function code 8) . . . 1095
Field-encoding (function code 0) . . . 1098
Field-decoding (function code 4) . . . 1099
Log capture routines . . . 1101
Specifying the log capture routine . . . 1101
When log capture routines are taken . . . 1102
Parameter lists for log capture routines . . . 1102
Routines for dynamic plan selection in CICS . . . 1103
Routine for CICS transaction invocation stored procedure . . . 1104
General considerations for writing exit routines . . . 1104
Coding rules for exit routines . . . 1104
Modifying exit routines. . . . 1105
Execution environment for exit routines . . . 1105
Registers at invocation for exit routines . . . 1105
Parameter lists for exit routines . . . 1106
Row formats for edit and validation routines . . . 1106
Column boundaries for edit and validation routines . . . 1107
Null values for edit procedures, field procedures, and validation routines . . . 1107
Fixed-length rows for edit and validation routines. . . . 1107
Varying-length rows for edit and validation routines . . . 1107
Varying-length rows with nulls for edit and validation routines . . . 1108

Dates, times, and timestamps for edit and validation routines . . . 1108
Parameter list for row format descriptions . . . 1108
DB2 codes for numeric data in edit and validation routines. . . . 1110
RACF access control module . . . 1110

Appendix C. Reading log records . . . 1111
Contents of the log . . . 1111
Unit of recovery log records . . . 1112
Checkpoint log records . . . 1115
Database page set control records . . . 1116
Other exception information . . . 1116
The physical structure of the log. . . . 1116
Physical and logical log records . . . 1116
The log record header . . . 1117
The log control interval definition (LCID). . . . 1118
Log record type codes . . . 1120
Log record subtype codes . . . 1120
Interpreting data change log records . . . 1121
Reading log records with IFI . . . 1122
Reading log records into a buffer . . . 1122
Reading specific log records (IFCID 0129) . . . 1122
Reading complete log data (IFCID 0306) . . . 1123
Reading log records with OPEN, GET, and CLOSE . . . 1126
Data sharing members that participate in a read . . . 1128
Registers and return codes . . . 1129
Stand-alone log OPEN request . . . 1129
Stand-alone log GET request . . . 1131
Stand-alone log CLOSE request . . . 1132
Sample application that uses stand-alone log services . . . 1133
Reading log records with the log capture exit routine. . . . 1134

Appendix D. Interpreting DB2 trace output . . . 1135
The sections of the trace output . . . 1135
SMF writer header section. . . . 1136
GTF writer header section . . . 1137
Self-defining section . . . 1143
Product section . . . 1145
Trace field descriptions . . . 1150

Appendix E. Programming for the Instrumentation Facility Interface (IFI) . . . 1151
Submitting DB2 commands through IFI . . . 1151
Obtaining trace data through IFI . . . 1152
Passing data to DB2 through IFI. . . . 1152
IFI functions . . . 1152
Invoking IFI from your program . . . 1153
Using IFI from stored procedures . . . 1154
COMMAND: Syntax and usage with IFI . . . 1154
Authorization for DB2 commands through IFI . . . 1154
Syntax for DB2 commands through IFI . . . 1154
Example of DB2 command through IFI . . . 1156
READS: Syntax and usage with IFI . . . 1156
Authorization for READS requests through IFI . . . 1157
Syntax for READS requests through IFI . . . 1157
Which qualifications are used for READS requests issued through IFI? . . . 1166
Usage notes for READS requests through IFI . . . 1167
Synchronous data and READS requests through IFI . . . 1167
Using READS calls to monitor the dynamic statement cache . . . 1169
Controlling collection of dynamic statement cache statistics with IFCID 0318 . . . 1170
READA: Syntax and usage with IFI . . . 1171
Authorization for READA requests through IFI. . . . 1171

Syntax for READA requests through IFI . . . 1171
Usage notes for READA requests through IFI . . . 1171
Asynchronous data and READA requests through IFI . . . 1172
Example of READA requests through IFI . . . 1173
WRITE: Syntax and usage with IFI . . . 1173
Authorization for WRITE requests through IFI . . . 1173
Syntax for WRITE requests through IFI . . . 1173
Usage notes for WRITE requests through IFI . . . 1174
Common communication areas for IFI calls . . . 1174
Instrument facility communications area (IFCA) . . . 1174
Return area . . . 1178
IFCID area . . . 1179
Output area . . . 1179
Using IFI in a data sharing group . . . 1179
Interpreting records returned by IFI . . . 1181
Trace data record format . . . 1181
Command record format . . . 1182
Data integrity and IFI . . . 1183
Auditing data and IFI . . . 1183
Locking considerations for IFI . . . 1183
Recovery considerations for IFI . . . 1184
Errors and IFI . . . 1184

Appendix F. Using tools to monitor performance . . . 1185
Using z/OS, CICS, and IMS tools . . . 1187
Monitoring system resources . . . 1188
Monitoring transaction manager throughput . . . 1189
DB2 trace . . . 1189
Types of traces . . . 1190
Effect on DB2 performance . . . 1193
Recording SMF trace data . . . 1193
Activating SMF . . . 1194
Allocating additional SMF buffers . . . 1194
Reporting data in SMF . . . 1195
Recording GTF trace data . . . 1195
OMEGAMON . . . 1196
Tivoli Decision Support for OS/390 . . . 1197
Monitoring application plans and packages . . . 1197

Appendix G. EXPLAIN tables that are used by optimization tools . . . 1199
DSN_PREDICAT_TABLE . . . 1199
DSN_STRUCT_TABLE . . . 1203
DSN_PGROUP_TABLE . . . 1205
DSN_PTASK_TABLE . . . 1208
DSN_FILTER_TABLE . . . 1210
DSN_DETCOST_TABLE . . . 1211
DSN_SORT_TABLE . . . 1218
DSN_SORTKEY_TABLE . . . 1219
DSN_PGRANGE_TABLE . . . 1221
DSN_VIEWREF_TABLE . . . 1222
DSN_QUERY_TABLE . . . 1224

Appendix H. Profile tables that are used by optimization tools . . . 1227
SYSIBM.DSN_VIRTUAL_INDEXES . . . 1227

Appendix I. Real-time statistics tables . . . 1229
Setting up your system for real-time statistics . . . 1229
Creating and altering the real-time statistics objects . . . 1230
Setting the interval for writing real-time statistics . . . 1231
Starting the real-time statistics database . . . 1231

Establishing base values for real-time statistics . . . 1231
Contents of the real-time statistics tables . . . 1231
Operating with real-time statistics . . . 1243
When DB2 externalizes real-time statistics . . . 1243
How DB2 utilities affect the real-time statistics . . . 1244
How non-DB2 utilities affect real-time statistics. . . . 1251
Real-time statistics on objects in work file databases and the TEMP database . . . 1251
| Real-time statistics for DEFINE NO objects . . . 1251
Real-time statistics on read-only or nonmodified objects . . . 1251
How dropping objects affects real-time statistics . . . 1251
How SQL operations affect real-time statistics counters . . . 1252
Real-time statistics in data sharing . . . 1253
Improving concurrency with real-time statistics . . . 1253
Recovering the real-time statistics tables . . . 1253
Statistics accuracy . . . 1254

Appendix J. DB2-supplied stored procedures . . . 1255
The DB2 real-time statistics stored procedure . . . 1257
Environment for DSNACCOR . . . 1258
Authorization required for DSNACCOR . . . 1258
DSNACCOR syntax diagram . . . 1259
DSNACCOR option descriptions . . . 1259
DSNACCOR formulas for recommending actions . . . 1267
Using an exception table . . . 1269
Example of DSNACCOR invocation . . . 1270
DSNACCOR output . . . 1274
WLM environment refresh stored procedure (WLM_REFRESH) . . . 1278
Environment for WLM_REFRESH . . . 1278
Authorization required for WLM_REFRESH. . . . 1278
WLM_REFRESH syntax diagram . . . 1279
WLM_REFRESH option descriptions . . . 1279
Example of WLM_REFRESH invocation . . . 1280
The CICS transaction invocation stored procedure (DSNACICS) . . . 1280
Environment for DSNACICS . . . 1281
Authorization required for DSNACICS . . . 1281
DSNACICS syntax diagram . . . 1281
DSNACICS option descriptions . . . 1281
DSNACICX user exit routine . . . 1283
Example of DSNACICS invocation . . . 1285
DSNACICS output . . . 1287
DSNACICS restrictions . . . 1287
DSNACICS debugging . . . 1287
The SYSIBM.USERNAMES encryption stored procedure. . . . 1287
Environment for DSNLEUSR . . . 1288
Authorization required for DSNLEUSR . . . 1288
DSNLEUSR syntax diagram . . . 1288
DSNLEUSR option descriptions . . . 1288
Example of DSNLEUSR invocation . . . 1289
DSNLEUSR output . . . 1290
IMS transactions stored procedure (DSNAIMS) . . . 1291
Environment for DSNAIMS . . . 1291
Authorization required for DSNAIMS . . . 1291
DSNAIMS syntax diagram . . . 1291
DSNAIMS option descriptions . . . 1291
Examples of DSNAIMS invocation . . . 1294
Connecting to multiple IMS subsystems with DSNAIMS . . . 1294
The DB2 EXPLAIN stored procedure . . . 1295
Environment . . . 1295
Authorization required . . . 1295
DSNAEXP syntax diagram . . . 1295
DSNAEXP option descriptions . . . 1296

Example of DSNAEXP invocation 1297
DSNAEXP output 1297

Appendix K. How to use the DB2 library 1299

Notices 1301
Programming Interface Information 1302
Trademarks 1304

Glossary 1305

Bibliography 1339

Index X-1

About this book

This two-volume book provides guidance information that you can use to perform a variety of administrative tasks with DB2 Universal Database for z/OS (DB2).

Unless it is stated otherwise, this information assumes that DB2® is running in new-function mode (as opposed to compatibility mode or enabling-new-function mode).

Important

In this version of DB2 UDB for z/OS, the DB2 Utilities Suite is available as an optional product. You must separately order and purchase a license to such utilities, and discussion of those utility functions in this publication is not intended to otherwise imply that you have a license to them. See Part 1 of *DB2 Utility Guide and Reference* for packaging details.

The DB2 Utilities Suite is designed to work with the DFSORT program, which you are licensed to use in support of the DB2 utilities even if you do not otherwise license DFSORT for general use. If your primary sort product is not DFSORT, consider the following informational APARs mandatory reading:

- II14047/II14213: USE OF DFSORT BY DB2 UTILITIES
- II13495: HOW DFSORT TAKES ADVANTAGE OF 64-BIT REAL ARCHITECTURE

These informational APARs are periodically updated.

Who should read this book

This book is primarily intended for system and database administrators. It assumes that the user is familiar with:

- The basic concepts and facilities of DB2
- Time Sharing Option (TSO) and Interactive System Productivity Facility (ISPF)
- The basic concepts of Structured Query Language (SQL)
- The basic concepts of Customer Information Control System (CICS®)
- The basic concepts of Information Management System (IMS™)
- How to define and allocate z/OS data sets using job control language (JCL).

Certain tasks require additional skills, such as knowledge of Transmission Control Protocol/Internet Protocol (TCP/IP) or Virtual Telecommunications Access Method (VTAM®) to set up communication between DB2 subsystems, or knowledge of the IBM System Modification Program (SMP/E) to install IBM licensed programs.

Terminology and citations

In this information, DB2 Universal Database™ for z/OS® is referred to as "DB2 UDB for z/OS." In cases where the context makes the meaning clear, DB2 UDB for z/OS is referred to as "DB2." When this information refers to titles of books in this library, a short title is used. (For example, "See *DB2 SQL Reference*" is a citation to *IBM® DB2 Universal Database for z/OS SQL Reference*.)

© Copyright IBM Corp. 1982, 2008

When referring to a DB2 product other than DB2 UDB for z/OS, this information uses the product's full name to avoid ambiguity.

The following terms are used as indicated:

DB2 Represents either the DB2 licensed program or a particular DB2 subsystem.

OMEGAMON
Refers to any of the following products:
- IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS
- IBM Tivoli OMEGAMON XE for DB2 Performance Monitor on z/OS
- IBM DB2 Performance Expert for Multiplatforms and Workgroups
- IBM DB2 Buffer Pool Analyzer for z/OS

C, C++, and C language
Represent the C or C++ programming language.

CICS Represents CICS Transaction Server for z/OS or CICS Transaction Server for OS/390®.

IMS Represents the IMS Database Manager or IMS Transaction Manager.

MVS™ Represents the MVS element of the z/OS operating system, which is equivalent to the Base Control Program (BCP) component of the z/OS operating system.

RACF®
Represents the functions that are provided by the RACF component of the z/OS Security Server.

How to read the syntax diagrams

The following rules apply to the syntax diagrams that are used in this book:

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

  The ►►── symbol indicates the beginning of a statement.

  The ──► symbol indicates that the statement syntax is continued on the next line.

  The ►── symbol indicates that a statement is continued from the previous line.

  The ──►◄ symbol indicates the end of a statement.
- Required items appear on the horizontal line (the main path).

```
►►──required_item──────────────────────────────────────────────────►◄
```

- Optional items appear below the main path.

```
►►──required_item──┬──────────────┬──────────────────────────────►◄
                   └──optional_item──┘
```

  If an optional item appears above the main path, that item has no effect on the execution of the statement and is used only for readability.

```
                   ┌─optional_item─┐
►►──required_item──┴───────────────┴──────────────────────────────►◄
```

- If you can choose from two or more items, they appear vertically, in a stack.

If you *must* choose one of the items, one item of the stack appears on the main path.

```
►►─required_item──┬─required_choice1─┬──────────────────────────────────►◄
                  └─required_choice2─┘
```

If choosing one of the items is optional, the entire stack appears below the main path.

```
►►─required_item──┬──────────────────┬──────────────────────────────────►◄
                  ├─optional_choice1─┤
                  └─optional_choice2─┘
```

If one of the items is the default, it appears above the main path and the remaining choices are shown below.

```
                  ┌─default_choice───┐
►►─required_item──┤                  ├──────────────────────────────────►◄
                  ├─optional_choice──┤
                  └─optional_choice──┘
```

- An arrow returning to the left, above the main line, indicates an item that can be repeated.

```
                  ┌──────────────────┐
                  ▼                  │
►►─required_item──┴─repeatable_item──┴──────────────────────────────────►◄
```

If the repeat arrow contains a comma, you must separate repeated items with a comma.

```
                  ┌─,────────────────┐
                  ▼                  │
►►─required_item──┴─repeatable_item──┴──────────────────────────────────►◄
```

A repeat arrow above a stack indicates that you can repeat the items in the stack.

- Keywords appear in uppercase (for example, `FROM`). They must be spelled exactly as shown. Variables appear in all lowercase letters (for example, *column-name*). They represent user-supplied names or values.
- If punctuation marks, parentheses, arithmetic operators, or other such symbols are shown, you must enter them as part of the syntax.

Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products. The major accessibility features in z/OS products, including DB2 UDB for z/OS, enable users to:

- Use assistive technologies such as screen reader and screen magnifier software
- Operate specific or equivalent features by using only a keyboard
- Customize display attributes such as color, contrast, and font size

Assistive technology products, such as screen readers, function with the DB2 UDB for z/OS user interfaces. Consult the documentation for the assistive technology products for specific information when you use assistive technology to access these interfaces.

Online documentation for Version 8 of DB2 UDB for z/OS is available in the Information management software for z/OS solutions information center, which is an accessible format when used with assistive technologies such as screen reader or screen magnifier software. The Information management software for z/OS solutions information center is available at the following Web site: http://publib.boulder.ibm.com/infocenter/dzichelp

How to send your comments

Your feedback helps IBM to provide quality information. Please send any comments that you have about this book or other DB2 UDB for z/OS documentation. You can use the following methods to provide comments:

- Send your comments by e-mail to db2zinfo@us.ibm.com and include the name of the product, the version number of the product, and the number of the book. If you are commenting on specific text, please list the location of the text (for example, a chapter and section title or a help topic title).
- You can send comments from the Web. Visit the library Web site at:

  www.ibm.com/software/db2zos/library.html

  This Web site has a an online reader comment form that you can use to send comments.
- You can also send comments by using the feedback link at the footer of each page in the Information Management Software for z/OS Solutions Information Center at http://publib.boulder.ibm.com/infocenter/db2zhelp.

Summary of changes to this book

This section summarizes the major changes to this book for Version 8.

Part 2, “Designing a database: advanced topics” has changed as follows:

- Chapter 3, “Creating storage groups and managing DB2 data sets,” on page 27 describes how to define data sets for partitioned table spaces and partitioned indexes (when you manage your own DB2 data sets with VSAM Access Method Services)
- Chapter 4, “Implementing your database design,” on page 43 describes:
  - Table-controlled partitioning, which uses the PARTITION BY clause of the CREATE TABLE statement
  - New definitions for clustering, partitioning, and secondary indexes
  - The NOT PADDED option of the CREATE INDEX statement, which allows DB2 to use index-only access for varying-length columns within the index key
  - How DB2 can use the same index to avoid sorts with either a forward index-scan or a backward index-scan (for queries with an ORDER BY clause)
- Chapter 6, “Altering your database design,” on page 67 describes how to use the ALTER TABLE and ALTER INDEX statements.

Part 3, “Security and auditing” has changed as follows:

- “Multilevel security” on page 192 describes a new security policy that prevents unauthorized users from accessing information at a higher classification than their authorization and prevents users from declassifying information.
- “Data encryption through built-in functions” on page 207 describes built-in functions that you can use to encrypt and decrypt sensitive data.
- Chapter 11, “Controlling access to a DB2 subsystem,” on page 231 includes information about enhanced options for data security in TCP/IP networks.

Part 4, “Operation and recovery” has changed as follows:

- “Monitoring databases” on page 365 describes the OVERVIEW option for DISPLAY DATABASE command, which you can use to display all of the objects in a database.
- “Resolution of WebSphere Application Server indoubt units of recovery” on page 461 explains how indoubt units of recovery are resolved for WebSphere Application Server.
- “Considerations for recovering indexes” on page 473 describes how indexes on altered tables and indexes on tables in partitioned table spaces can restrict recovery.
- “System-level point in time recovery” on page 511 details how the system-level point-in-time (PIT) recovery capability of DB2 provides improved usability, more flexibility, and faster recovery times.
- Chapter 22, “Recovery scenarios,” on page 517 explains that with the introduction of table-based partitioning in Version 8 of DB2, you can redefine a table-based partition and an index-based partition when an out of disk space condition occurs or an extent limit is reached.

Part 5, “Performance monitoring and tuning” has changed as follows:

© Copyright IBM Corp. 1982, 2008

- “Allocating buffer pool storage” on page 673 provides recommendations on allocating buffer pool storage.
- “Long-term page fix option for buffer pools” on page 675 describes how to use the PGFIX keyword with the ALTER BUFFERPOOL command to fix a buffer pool in real storage for an extended period of time.
- “Evaluating your indexes” on page 705 provides information about evaluating the effectiveness of your indexes, avoiding unnecessary indexes, and using nonpadded indexes.
- “Database access threads” on page 735 introduces new terminology for database access threads. Type 1 inactive threads become inactive database access threads and type 2 inactive threads become inactive connections. Additionally, pooling behaviors are discussed.
- Chapter 30, “Tuning your queries ,” on page 745 includes discussion of the predicates IS DISTINCT FROM and IS NOT DISTINCT FROM.
- “Changing the access path at run time” on page 774 describes the bind options REOPT(ONCE) and REOPT(ALWAYS).
- “Favoring index access” on page 792 introduces the VOLATILE keyword and explains how to use it to minimize contention among certain applications.
- Chapter 32, “Using materialized query tables,” on page 881 describes how to use materialized query tables to improve the performance of queries that require expensive join and aggregation operations, such as some queries that are used in data warehousing applications.
- “Dedicated virtual memory pool for star join operations” on page 967 explains the advantages of a dedicated virtual memory pool for star join operations and demonstrates how to determine the size of the virtual memory pool.

Appendix B, “Writing exit routines” has changed as follows:

- “Session variables in connection and sign-on routines” on page 1060 explains how session variables work in connection and sign-on routines.
- “Creating materialized query tables” on page 1066 discusses the authorization issues involved with creating a materialized query table.
- “RACF access control module” on page 1110 describes the RACF access control module.

Appendix D, “Interpreting DB2 trace output” has changed as follows:

- “Self-defining section” on page 1143 describes the self-defining section for variable-length data items.

Part 1. Introduction

Chapter 1. System planning concepts 3
The structure of DB2. 3
Data structures 3
Databases 5
Storage groups 5
Table spaces 5
Tables 6
Indexes 6
Views. 7
System structures 7
DB2 catalog. 7
DB2 directory 8
Active and archive logs 8
Bootstrap data set (BSDS) 9
Buffer pools 9
Data definition control support database . . . 9
Resource limit facility database 10
Work file database 10
TEMP database 10
More information about data structures 10
Control and maintenance of DB2 11
Commands 11
Utilities 11
High availability. 12
Daily operations and tuning 12
Backup and recovery 12
Restart 13
More information about control and maintenance of DB2 13
The DB2 environment 13
Address spaces 13
DB2 lock manager 14
What IRLM does 14
Administering IRLM 14
DB2 attachment facilities 14
WebSphere 15
CICS 15
IMS 16
TSO 17
CAF 18
RRS 18
DB2 and distributed data 18
DB2 and z/OS 19
DB2 and the Parallel Sysplex 19
DB2 and the Security Server for z/OS 19
DB2 and DFSMS. 20
More information about the z/OS environment 20

© Copyright IBM Corp. 1982, 2008

Chapter 1. System planning concepts

This chapter introduces the DB2 UDB for z/OS system and explains the concepts that relate to system and database administration. It consists of the following sections:

- "The structure of DB2" describes the elements you deal with when using DB2.
- "Control and maintenance of DB2" on page 11 briefly describes commands and utility jobs.
- "The DB2 environment" on page 13 describes the main DB2 components and explains how DB2 operates with certain related IBM products.

Each section concludes with a list of citations to more detailed information about the topics that the sections introduces.

If you are new DB2 UDB for z/OS, begin with *The Official Introduction to DB2 UDB for z/OS* for extensive conceptual information.

General information about DB2 UDB for z/OS is available from the DB2 UDB for z/OS World Wide Web page:

`http://www.software.ibm.com/data/db2/os390/`

The structure of DB2

The elements that DB2 manages can be divided into two broad categories:

- Data structures, which are accessed under the user's direction and by which the user's data (and some system data) is organized.
- System structures, which are controlled and accessed by DB2.

Data structures

DB2 data structures described in this section include:

"Databases" on page 5
"Storage groups" on page 5
"Table spaces" on page 5
"Tables" on page 6
"Indexes" on page 6
"Views" on page 7

The brief descriptions here show how the structures fit into an overall view of DB2.

Figure 1 on page 4 shows how some DB2 structures contain others. To some extent, the notion of "containment" provides a hierarchy of structures. This section introduces those structures from the most to the least inclusive.

© Copyright IBM Corp. 1982, 2008

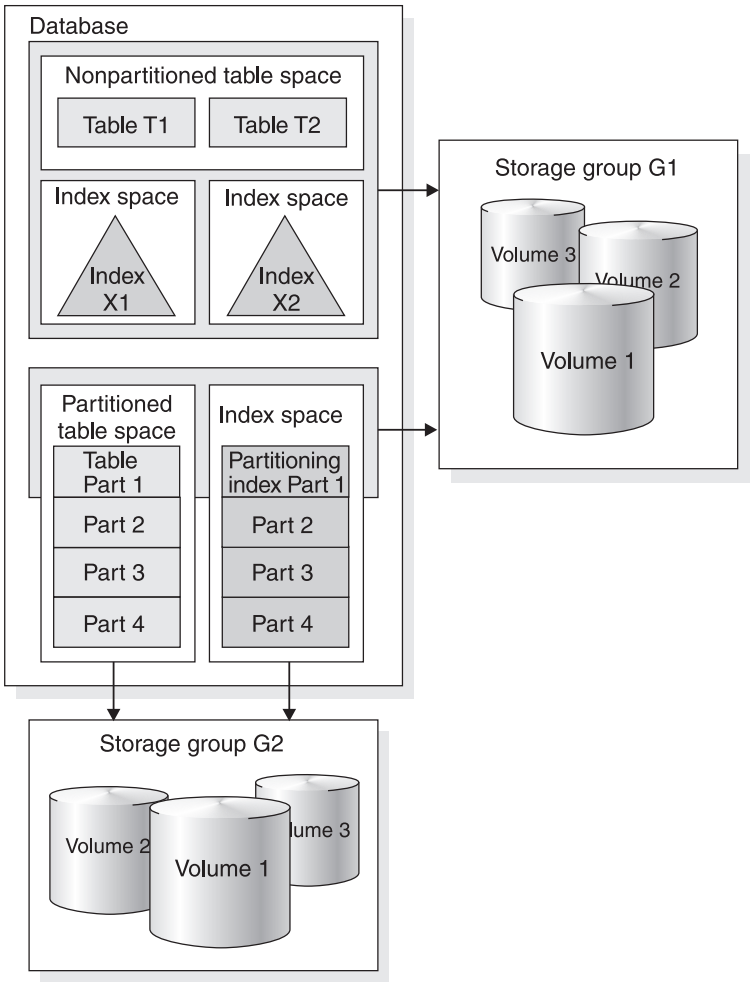


Figure 1. A hierarchy of DB2 structures

The DB2 objects that Figure 1 introduces are:

Databases
A set of DB2 structures that include a collection of tables, their associated indexes, and the table spaces in which they reside.

Storage groups
A set of volumes on disks that hold the data sets in which tables and indexes are actually stored.

Table spaces
A set of volumes on disks that hold the data sets in which tables and indexes are actually stored.

Tables All data in a DB2 database is presented in *tables*—collections of rows all having the same columns. A table that holds persistent user data is a *base table*. A table that stores data temporarily is a *temporary table*.

Indexes
An *index* is an ordered set of pointers to the data in a DB2 table. The index is stored separately from the table.

Views A *view* is an alternate way of representing data that exists in one or more tables. A view can include all or some of the columns from one or more base tables.

Databases

A single database can contain all the data associated with one application or with a group of related applications. Collecting that data into one database allows you to start or stop access to all the data in one operation and grant authorization for access to all the data as a single unit. Assuming that you are authorized to do so, you can access data stored in different databases.

If you create a table space or a table and do not specify a database, the table or table space is created in the default database, DSNDB04. DSNDB04 is defined for you at installation time. All users have the authority to create table spaces or tables in database DSNDB04. The system administrator can revoke those privileges and grant them only to particular users as necessary.

When you migrate to Version 8, DB2 adopts the default database and default storage group you used in Version 7. You have the same authority for Version 8 as you did in Version 7.

Storage groups

The description of a storage group names the group and identifies its volumes and the VSAM (virtual storage access method) catalog that records the data sets. The default storage group, SYSDEFLT, is created when you install DB2.

All volumes of a given storage group must have the same device type. But, as Figure 1 on page 4 suggests, parts of a single database can be stored in different storage groups.

Table spaces

A table space can consist of a number of VSAM data sets. Data sets are VSAM linear data sets (LDSs). Table spaces are divided into equal-sized units, called *pages*, which are written to or read from disk in one operation. You can specify page sizes for the data; the default page size is 4 KB.

When you create a table space, you can specify the database to which the table space belongs and the storage group it uses. If you do not specify the database and storage group, DB2 assigns the table space to the default database and the default storage group.

You also determine what kind of table spaces is created.

Partitioned
: Divides the available space into separate units of storage called *partitions*. Each partition contains one data set of one table. You assign the number of partitions (from 1 to 4096) and you can assign partitions independently to different storage groups.

Segmented
: Divides the available space into groups of pages called *segments*. Each segment is the same size. A segment contains rows from only one table.

Large object (LOB)
: Holds large object data such as graphics, video, or very large text strings. A LOB table space is always associated with the table space that contains the logical LOB column values. The table space that contains the table with the LOB columns is called, in this context, the *base table space*.

Simple
: Can contain more than one table. The rows of different tables are not kept separate (unlike segmented table spaces).

Tables

When you create a table in DB2, you define an ordered set of columns.

Sample tables: The examples in this book are based on the set of tables described in Appendix A, "DB2 sample tables," on page 1031. The sample tables are part of the DB2 licensed program and represent data related to the activities of an imaginary computer services company, the Spiffy Computer Services Company. Table 1 shows an example of a DB2 sample table.

Table 1. Example of a DB2 sample table (Department table)

DEPTNO	DEPTNAME	MGRNO	ADMRDEPT
A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00
B01	PLANNING	000020	A00
C01	INFORMATION CENTER	000030	A00
D01	DEVELOPMENT CENTER		A00
E01	SUPPORT SERVICES	000050	A00
D11	MANUFACTURING SYSTEMS	000060	D01
D21	ADMINISTRATION SYSTEMS	000070	D01
E11	OPERATIONS	000090	E01
E21	SOFTWARE SUPPORT	000100	E01

The department table contains:

- **Columns:** The ordered set of columns are DEPTNO, DEPTNAME, MGRNO, and ADMRDEPT. All the data in a given column must be of the same data type.
- **Row:** Each row contains data for a single department.
- **Value:** At the intersection of a column and row is a *value*. For example, PLANNING is the value of the DEPTNAME column in the row for department B01.
- **Referential constraints:** You can assign a *primary key* and *foreign keys* to tables. DB2 can automatically enforce the integrity of references from a foreign key to a primary key by guarding against insertions, updates, or deletions that violate the integrity.
  - **Primary key:** A column or set of columns whose values uniquely identify each row, for example, DEPTNO.
  - **Foreign key:** Columns of other tables, whose values must be equal to values of the primary key of the first table (in this case, the department table). In the sample employee table, the column that shows what department an employee works in is a foreign key; its values must be values of the department number column in the department table.

Indexes

Each index is based on the values of data in one or more columns of a table. After you create an index, DB2 maintains the index, but you can perform necessary maintenance such as reorganizing it or recovering the index.

Indexes take up physical storage in *index spaces*. Each index occupies its own index space.

The main purposes of indexes are:

- To improve performance. Access to data is often faster with an index than without.
- To ensure that a row is unique. For example, a unique index on the employee table ensures that no two employees have the same employee number.

Except for changes in performance, users of the table are unaware that an index is in use. DB2 decides whether to use the index to access the table. There are ways to influence how indexes affect performance when you calculate the storage size of an index and determine what type of index to use. An index can be partitioning, nonpartitioning, or clustered. For example, you can apportion data by last names, maybe using one partition for each letter of the alphabet. Your choice of a partitioning scheme is based on how an application accesses data, how much data you have, and how large you expect the total amount of data to grow.

Views

Views allow you to shield some table data from end users. A view can be based on other views or on a combination of views and tables.

When you define a view, DB2 stores the definition of the view in the DB2 catalog. However, DB2 does not store any data for the view itself, because the data already exists in the base table or tables.

System structures

DB2 system structures described in this section include:

"DB2 catalog"
"DB2 directory" on page 8
"Active and archive logs" on page 8
"Bootstrap data set (BSDS)" on page 9
"Buffer pools" on page 9
"Data definition control support database" on page 9
"Resource limit facility database" on page 10
"Work file database" on page 10
"TEMP database" on page 10

In addition, Parallel Sysplex® data sharing uses shared system structures.

DB2 catalog

The DB2 catalog consists of tables of data about everything defined to the DB2 system, including table spaces, indexes, tables, copies of table spaces and indexes, storage groups, and so forth. The system database DSNDB06 contains the DB2 catalog.

When you create, alter, or drop any structure, DB2 inserts, updates, or deletes rows of the catalog that describe the structure and tell how the structure relates to other structures. For example, SYSIBM.SYSTABLES is one catalog table that records information when a table is created. DB2 inserts a row into SYSIBM.SYSTABLES that includes the table name, its owner, its creator, and the name of its table space and its database.

Because the catalog consists of DB2 tables in a DB2 database, authorized users can use SQL statements to retrieve information from it.

The *communications database* (CDB) is part of the DB2 catalog. The CDB consists of a set of tables that establish conversations with remote database management systems (DBMSs). The distributed data facility (DDF) uses the CDB to send and receive distributed data requests.

DB2 directory

The DB2 directory contains information that DB2 uses during normal operation. You cannot access the directory using SQL, although much of the same information is contained in the DB2 catalog, for which you can submit queries. The structures in the directory are not described in the DB2 catalog.

The directory consists of a set of DB2 tables stored in five table spaces in system database DSNDB01. Each of the table spaces listed in Table 2 is contained in a VSAM linear data set.

Table 2. Directory table spaces

Table space name	Description
SCT02 Skeleton cursor (SKCT)	Contains the internal form of SQL statements contained in an application. When you bind a plan, DB2 creates a skeleton cursor table in SCT02.
SPT01 Skeleton package	Similar to SCT02 except that the skeleton package table is created when you bind a package.
SYSLGRNX Log range	Tracks the opening and closing of table spaces, indexes, or partitions. By tracking this information and associating it with relative byte addresses (RBAs) as contained in the DB2 log, DB2 can reduce recovery time by reducing the amount of log that must be scanned for a particular table space, index, or partition.
SYSUTILX System utilities	Contains a row for every utility job that is running. The row stays until the utility is finished. If the utility terminates without completing, DB2 uses the information in the row when you restart the utility.
DBD01 Database descriptor (DBD)	Contains internal information, called *database descriptors* (DBDs), about the databases that exist within DB2. Each database has exactly one corresponding DBD that describes the database, table spaces, tables, table check constraints, indexes, and referential relationships. A DBD also contains other information about accessing tables in the database. DB2 creates and updates DBDs whenever their corresponding databases are created or updated.

Active and archive logs

DB2 records all data changes and significant events in a log as they occur. In the case of failure, DB2 uses this data to recover.

DB2 writes each log record to a disk data set called the *active log*. When the active log is full, DB2 copies the contents of the active log to a disk or magnetic tape data set called the *archive log*.

You can choose either single logging or dual logging.

- A single active log contains as many as 31 or 93 active log data sets, depending on the format of the BSDS. If the BSDS has been installed with support for the larger BSDS or converted to support it, then the maximum number of active log data sets is 93. Otherwise, the maximum number is 31.
- With dual logging, the active log has twice the capacity for active log data sets, because two identical copies of the log records are kept.

Each active log data set is a single-volume, single-extent VSAM LDS.

Bootstrap data set (BSDS)

The *bootstrap data set* (BSDS) is a VSAM key-sequenced data set (KSDS) that contains information critical to DB2. Specifically, the BSDS contains:

- An inventory of all active and archive log data sets known to DB2. DB2 uses this information to track the active and archive log data sets. DB2 also uses this information to locate log records to satisfy log read requests during normal DB2 system activity and during restart and recovery processing.
- A wrap-around inventory of all recent DB2 checkpoint activity. DB2 uses this information during restart processing.
- The distributed data facility (DDF) communication record, which contains information necessary to use DB2 as a distributed server or requester.
- Information about buffer pools.

Because the BSDS is essential to recovery in the event of subsystem failure, during installation DB2 automatically creates two copies of the BSDS and, if space permits, places them on separate volumes.

Buffer pools

Buffer pools are areas of virtual storage in which DB2 temporarily stores pages of table spaces or indexes. When an application program accesses a row of a table, DB2 retrieves the page containing that row and places the page in a buffer. If the needed data is already in a buffer, the application program does not have to wait for it to be retrieved from disk, significantly reducing the cost of retrieving the page.

Buffer pools require monitoring and tuning. The size of buffer pools is critical to the performance characteristics of an application or group of applications that access data in those buffer pools.

When you use Parallel Sysplex data sharing, buffer pools map to structures called *group buffer pools*. These structures reside in a special PR/SM™ LPAR logical partition called a *coupling facility*, which enables several DB2s to share information and control the coherency of data.

Buffer pools reside in DB2's DBM1 primary address space. This option offers the best performance. In Version 8, the maximum size of a buffer pool increases to 1 TB.

Data definition control support database

The *data definition control support database* is automatically created during installation. This database is a user-maintained collection of tables used by data definition control support to restrict the submission of specific DB2 DDL (data definition language) statements to selected application identifiers (plans or collections of packages). After this database is created, you must populate the tables to use of this facility. The system name for this database is DSNRGFDB.

Resource limit facility database

The *resource limit facility database* (DSNRLST) is a facility that lets you control the amount of processor resources used by dynamic SELECT statements. For example, you might choose to disable bind operations during critical times of day to avoid contention with the DB2 catalog.

You can establish a single limit for all users, different limits for individual users, or both. You can choose to have these limits applied before the statement is executed (this is called *predictive* governing), or while a statement is running (sometimes called *reactive* governing). You can even use both modes of governing. You define these limits in one or more resource limit specification tables (RLST).

Work file database

The *work file database* is used as storage for processing SQL statements that require working space, such as that required for a sort. DB2 creates a work file database for you at installation time, and you can create additional work file table spaces at any time using CREATE TABLESPACE statements.

In a non-data-sharing environment, the work file database is called DSNDB07. In a data sharing environment, each DB2 member in the data sharing group has its own work file database.

TEMP database

The TEMP database is for declared temporary tables only. DB2 stores all declared temporary tables in this database. You can create one TEMP database for each DB2 subsystem or data sharing member.

More information about data structures

Table 3 lists additional information sources about topics that this section introduces.

Table 3. More information about DB2 structures

For more information about...	See...
Basic concepts for designing data structures, including: • Table spaces • Tables, views • Columns • Indexes	*The Official Introduction to DB2 UDB for z/OS*
Data structures	
Data structures, defining	• *The Official Introduction to DB2 UDB for z/OS* • Chapter 4, "Implementing your database design," on page 43
Table space size limits	Appendix A of *DB2 SQL Reference*
Table columns, data types	Volume 1 of *DB2 SQL Reference*
Referential integrity	Volume 1 of *DB2 Application Programming and SQL Guide*
System structures	
Shared system structures	*DB2 Data Sharing: Planning and Administration*
Catalog tables	Appendix F of *DB2 SQL Reference*
Catalog, data set naming conventions	*DB2 Installation Guide*

Table 3. More information about DB2 structures (continued)

For more information about...	See...
CDB	*DB2 Installation Guide*
Directory, data set naming conventions	*DB2 Installation Guide*
Logs	Chapter 18, "Managing the log and the bootstrap data set," on page 423
BSDS usage, functions	"Managing the bootstrap data set (BSDS)" on page 436
Buffer pools, tuning	• Chapter 27, "Tuning DB2 buffer, EDM, RID, and sort pools," on page 665 • *DB2 Command Reference*
Group buffer pools	*DB2 Data Sharing: Planning and Administration*
Data definition control support database	Chapter 10, "Controlling access through a closed application," on page 215
RLST	"Resource limit facility (governor)" on page 716
Work file and TEMP database, defining	Volume 2 of *DB2 SQL Reference*

Control and maintenance of DB2

You use commands and utilities to perform the tasks required to control and maintain DB2:

- Commands can be entered at a terminal, a z/OS console, or through an APF-authorized program or application that uses the instrumentation facility interface (IFI)
- Utility jobs run as standard batch jobs

Commands

The commands are divided into the following categories:

- DSN command and subcommands
- DB2 commands
- IMS commands
- CICS attachment facility commands
- IRLM commands
- TSO CLIST commands

To enter a DB2 command from an authorized z/OS consolee, you use a subsystem command prefix (composed of 1 to 8 characters) at the beginning of the command. The default subsystem command prefix is *-DSN1*, which you can change when you install or migrate DB2.

Example: The following command starts the DB2 subsystem that is associated with the command prefix -DSN1:

```
-DSN1 START DB2
```

Utilities

You use utilities to perform many of the tasks required to maintain DB2 data. Those tasks include loading a table, copying a table space, or recovering a database to a previous point in time.

The utilities run as batch jobs. *DB2 interactive* (DB2I) provides a simple way to prepare the job control language (JCL) for those jobs and to perform many other operations by entering values on panels. DB2I runs under TSO using ISPF services. A utility control statement tells a particular utility what task to perform.

High availability

It is not necessary to start or stop DB2 often. DB2 continually adds function to improve availability, especially in the following areas:

- "Daily operations and tuning"
- "Backup and recovery"
- "Restart" on page 13

Daily operations and tuning

Some of the high availability features related to normal DB2 operations include:

- You can bind application plans and packages online. Packages let you change and rebind smaller units. Using package *versions* permits binding while the applications continue to run.
- You can define and change databases and authorizations online.
- You can change buffer pool sizes online.
- You can use utilities to reorganize indexes, table spaces, or partitions of indexes or table spaces.
- DB2's data sharing function lets several DB2 subsystems process applications on shared data. While the different subsystems share data, they appear as a single DB2 to end users. Applications can be rerouted to avoid outages if one of the subsystems must be taken down for maintenance.

Backup and recovery

Unplanned outages are difficult to avoid entirely. However, a good backup strategy can reduce the elapsed time of an unplanned outage. To reduce the probability and duration of unplanned outages, you should periodically back up and reorganize your data.

A lot of factors affect the availability of the databases. Here are some key points to be aware of:

- You should limit your use of, and understand the options of, utilities such as COPY and REORG.
  - You can recover online such structures as table spaces, partitions, data sets, a range of pages, a single page, and indexes.
  - You can recover table spaces and indexes at the same time to reduce recovery time.
  - With some options on the COPY utility, you can read and update a table space while copying it.
- I/O errors have the following affects:
  - I/O errors on a range of data do not affect availability to the rest of the data.
  - If an I/O error occurs when DB2 is writing to the log, DB2 continues to operate.
  - If an I/O error is on the active log, DB2 moves to the next data set. If the error is on the archive log, DB2 dynamically allocates another data set.
- Documented disaster recovery methods are crucial in the case of disasters that might cause a complete shutdown of your local DB2 system.
- If DB2 is forced to a single mode of operations for the bootstrap data set or logs, you can usually restore dual operation while DB2 continues to run.

Restart

A key to the perception of high availability is getting the DB2 subsystem back up and running quickly after an unplanned outage.

- Some restart processing can occur concurrently with new work. Also, you can choose to postpone some processing.
- During a restart, DB2 applies data changes from the log. This technique ensures that data changes are not lost, even if some data was not written at the time of the failure. Some of the process of applying log changes can run in parallel
- You can register DB2 to the Automatic Restart Manager of z/OS. This facility automatically restarts DB2 should it go down as a result of a failure.

More information about control and maintenance of DB2

Table 4 lists additional information sources about topics that this section introduces.

Table 4. More information about DB2 control and maintenance

For more information about...	See...
Commands	• Part 4, "Operation and recovery," on page 311 • *DB2 Command Reference*
Utilities	*DB2 Utility Guide and Reference*
Data sharing	*DB2 Data Sharing: Planning and Administration*
Recovery	• "Recovering page sets and data sets" on page 491 • *DB2 Utility Guide and Reference*

The DB2 environment

This section provides an overview of DB2 components and environments that work together in the z/OS environment. DB2 operates as a formal subsystem of z/OS. DB2 utilities run in the batch environment, and applications that access DB2 resources can run in the batch, TSO, IMS, or CICS environments. IBM provides attachment facilities to connect DB2 to each of these environments.

Address spaces

DB2 uses several different address spaces for the following purposes:

Database services
: *ssnm*DBM1 manipulates most of the structures in user-created databases. In Version 8, storage areas such as buffer pools reside above the 2GB bar in the *ssnm*DBM1 address space. With 64-bit virtual addressing to access these storage areas, buffer pools to scale to extremely large sizes.

System services
: *ssnm*MSTR performs a variety of system-related functions.

Distributed data facility
: *ssnm*DIST provides support for remote requests.

IRLM (Internal resource lock manager)
: IRLMPROC controls DB2 locking.

DB2-established
: *ssnm*SPAS, for stored procedures, provides an isolated execution environment for user-written SQL programs at a DB2 server.

WLM-established
Zero to many address spaces for stored procedures and user-defined functions. WLM-established address spaces are handled in order of priority and isolated from other stored procedures or user-defined functions running in other address spaces

User address spaces
At least one, possibly several, of the following types of user address spaces:
- TSO
- Batch
- CICS
- IMS dependent region
- IMS control region

DB2 lock manager

DB2's internal resource lock manager (IRLM) is both a separate subsystem and an integral component of DB2. IRLM is shipped with DB2, and each DB2 subsystem must have its own instance of IRLM.

Recommendation: Always run with the latest level of IRLM.

You cannot share IRLM between DB2s or between DB2 and IMS. (IRLM is also shipped with IMS.) If you are running a DB2 data sharing group, there is a corresponding IRLM group.

What IRLM does

IRLM works with DB2 to serialize access to your data. DB2 requests locks from IRLM to ensure data integrity when applications, utilities, commands, and so forth, are all attempting to access the same data.

Administering IRLM

IRLM requires some control and monitoring. The external interfaces to the IRLM include:

- Installation

  Install IRLM when you install DB2. Consider that locks take up storage, and adequate storage for IRLM is crucial to the performance of your system.

  Another important performance item is to make the priority of the IRLM address space above all the DB2 address spaces.
- Commands

  Some z/OS commands specifically for IRLM let you modify parameters, display information about the status of the IRLM and its storage use, and start and stop IRLM.
- Tracing

  DB2's trace facility gives you the ability to trace lock interactions.

  You can use z/OS trace commands or IRLMPROC options to control diagnostic traces for IRLM. You normally use these traces under the direction of IBM Service.

DB2 attachment facilities

This section describes the attachment facilities that you can use in the z/OS environment to begin a DB2 session. You can also begin DB2 sessions from other environments on clients such as Windows® or UNIX® by using interfaces that include ODBC, JDBC, and SQLJ.

An *attachment facility* provides the interface between DB2 and another environment. Figure 2 shows the z/OS attachment facilities with interfaces to DB2.

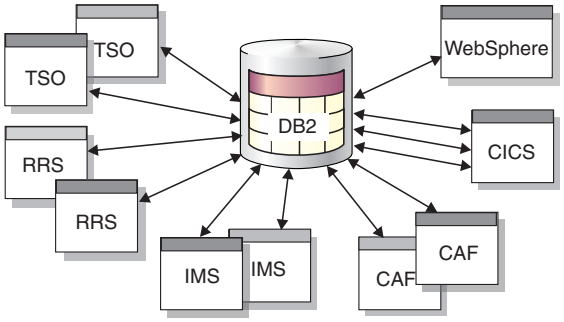


Figure 2. Attaching to DB2

The z/OS environments include:
- WebSphere®
- CICS (Customer Information Control System)
- IMS (Information Management System)
- TSO (Time Sharing Option)
- Batch

The z/OS attachment facilities include:
- CICS
- IMS
- TSO
- CAF (call attachment facility)
- RRS (Resource Recovery Services)

In the TSO and batch environments, you can use the TSO, CAF, and RRS attachment facilities to access DB2.

WebSphere

WebSphere products that are integrated with DB2 include WebSphere Application Server, WebSphere Studio, and Transaction Servers & Tools. In the WebSphere environment, you can use the RRS attachment facility.

CICS

The *Customer Information Control System* (CICS) attachment facility provided with the CICS transaction server lets you access DB2 from CICS. After you start DB2, you can operate DB2 from a CICS terminal. You can start and stop CICS and DB2 independently, and you can establish or terminate the connection between them at any time. You also have the option of allowing CICS to connect to DB2 automatically.

The CICS attachment facility also provides CICS applications with access to DB2 data while operating in the CICS environment. CICS applications, therefore, can access both DB2 data and CICS data. In case of system failure, CICS coordinates recovery of both DB2 and CICS data.

CICS operations: The CICS attachment facility uses standard CICS command-level services where needed.

Examples:

```
EXEC CICS WAIT
EXEC CICS ABEND
```

A portion of the CICS attachment facility executes under the control of the transaction issuing the SQL requests. Therefore these calls for CICS services appear to be issued by the application transaction.

With proper planning, you can include DB2 in a CICS XRF recovery scenario.

Application programming with CICS: Programmers writing CICS command-level programs can use the same data communication coding techniques to write the data communication portions of application programs that access DB2 data. Only the database portion of the programming changes. For the database portions, programmers use SQL statements to retrieve or modify data in DB2 tables.

To a CICS terminal user, application programs that access both CICS and DB2 data appear identical to application programs that access only CICS data.

DB2 supports this cross-product programming by coordinating recovery resources with those of CICS. CICS applications can therefore access CICS-controlled resources as well as DB2 databases.

Function shipping of SQL requests is not supported. In a CICS multi-region operation (MRO) environment, each CICS address space can have its own attachment to the DB2 subsystem. A single CICS region can be connected to only one DB2 subsystem at a time.

System administration and operation with CICS: An authorized CICS terminal operator can issue DB2 commands to control and monitor both the attachment facility and DB2 itself. Authorized terminal operators can also start and stop DB2 databases.

Even though you perform DB2 functions through CICS, you need to have the TSO attachment facility and ISPF to take advantage of the online functions supplied with DB2 to install and customize your system. You also need the TSO attachment to bind application plans and packages.

IMS

The *Information Management System* (IMS) attachment facility allows you to access DB2 from IMS. The IMS attachment facility receives and interprets requests for access to DB2 databases using exits provided by IMS subsystems. Usually, IMS connects to DB2 automatically with no operator intervention.

In addition to Data Language I (DL/I) and Fast Path calls, IMS applications can make calls to DB2 using embedded SQL statements. In case of system failure, IMS coordinates recovery of both DB2 and IMS data.

With proper planning, you can include DB2 in an IMS XRF recovery scenario.

Application programming with IMS: With the IMS attachment facility, DB2 provides database services for IMS dependent regions. DL/I batch support allows users to access both IMS data (DL/I) and DB2 data in the IMS batch environment, which includes:

- Access to DB2 and DL/I data from application programs.
- Coordinated recovery through a two-phase commit process.
- Use of the IMS extended restart (XRST) and symbolic checkpoint (CHKP) calls by application programs to coordinate recovery with IMS, DB2, and generalized sequential access method (GSAM) files.

IMS programmers writing the data communication portion of application programs do not need to alter their coding technique to write the data communication portion when accessing DB2; only the database portions of the application programs change. For the database portions, programmers code SQL statements to retrieve or modify data in DB2 tables.

To an IMS terminal user, IMS application programs that access DB2 appear identical to IMS.

DB2 supports this cross-product programming by coordinating database recovery services with those of IMS. Any IMS program uses the same synchronization and rollback calls in application programs that access DB2 data as they use in IMS DB/DC application programs that access DL/I data.

Another aid for cross-product programming is the DataPropagator™ NonRelational (DPropNR) licensed program. DPropNR allows automatic updates to DB2 tables when corresponding information in an IMS database is updated, and it allows automatic updates to an IMS database when a DB2 table is updated.

System administration and operation with IMS: An authorized IMS terminal operator can issue DB2 commands to control and monitor DB2. The terminal operator can also start and stop DB2 databases.

Even though you perform DB2 functions through IMS, you need the TSO attachment facility and ISPF to take advantage of the online functions supplied with DB2 to install and customize your system. You also need the TSO attachment facility to bind application plans and packages.

TSO

The *Time Sharing Option* (TSO) attachment facility is required for binding application plans and packages and for executing several online functions that are provided with DB2.

Using the TSO attachment facility, you can access DB2 by running in either foreground or batch. You gain foreground access through a TSO terminal; you gain batch access by invoking the TSO terminal monitor program (TMP) from a batch job.

The following two command processors are available:

- DSN command processor — Runs as a TSO command processor and uses the TSO attachment facility.
- DB2 Interactive (DB2I) — Consists of Interactive System Productivity Facility (ISPF) panels. ISPF has an interactive connection to DB2, which invokes the DSN command processor. Using DB2I panels, you can perform most DB2 tasks interactively, such as running SQL statements, commands, and utilities.

Whether you access DB2 in foreground or batch, attaching through the TSO attachment facility and the DSN command processor makes access easier. DB2 subcommands that execute under DSN are subject to the command size limitations as defined by TSO. TSO allows authorized DB2 users or jobs to create, modify, and maintain databases and application programs. You invoke the DSN processor from the foreground by issuing a command at a TSO terminal. From batch, first invoke TMP from within a batch job, and then pass commands to TMP in the SYSTSIN data set.

After DSN is running, you can issue DB2 commands or DSN subcommands. You cannot issue a -START DB2 command from within DSN. If DB2 is not running, DSN cannot establish a connection to it; a connection is required so that DSN can transfer commands to DB2 for processing.

CAF

Most TSO applications must use the TSO attachment facility, which invokes the DSN command processor. Together, DSN and TSO provide services such as automatic connection to DB2, attention key support, and translation of return codes into error messages. However, when using DSN services, your application must run under the control of DSN.

The *call attachment facility* (CAF) provides an alternative connection for TSO and batch applications needing tight control over the session environment. Applications using CAF can *explicitly* control the state of their connections to DB2 by using connection functions that CAF supplies.

RRS

z/OS Resource Recovery Services is a newer implementation of CAF with additional capabilities. RRS is a feature of z/OS that coordinates commit processing of recoverable resources in a z/OS system. DB2 supports use of these services for DB2 applications that use the RRS attachment facility provided with DB2. Use the RRS attachment to access resources such as SQL tables, DL/I databases, MQSeries® messages, and recoverable VSAM files within a single transaction scope.

The RRS attachment is required for stored procedures that run in a WLM-established address space.

DB2 and distributed data

In a distributed data environment, DB2 applications can access data at many different DB2 sites and at remote relational database systems.

Example: Assume a company needs to satisfy customer requests at hundreds of locations and the company representatives who answer those requests work at locations that span a wide geographic area. You can document requests on workstations that have DB2 Connect™ Personal Edition. This information is uploaded to DB2 UDB for z/OS. The representatives can then use Java™ applications to access the customer request information in DB2 from their local offices.

The company's distributed environment relies on the distributed data facility (DDF), which is part of DB2 UDB for z/OS. DB2 applications can use DDF to access data at other DB2 sites and at remote relational database systems that support *Distributed Relational Database Architecture*™ (DRDA®). *DRDA* is a standard for distributed connectivity. All IBM DB2 servers support this DRDA standard.

DDF also enables applications that run in a remote environment that supports DRDA. These applications can use DDF to access data in DB2 servers. Examples of application requesters include IBM DB2 Connect and other DRDA-compliant client products.

With DDF, you can have up to 150 000 distributed *threads* connect to a DB2 server at the same time. A thread is a DB2 structure that describes an application's connection and traces its progress.

Use *stored procedures* to reduce processor and elapsed time costs of distributed access. A stored procedure is user-written SQL program that a requester can invoke at the server. By encapsulating the SQL, many fewer messages flow across the wire.

Local DB2 applications can use stored procedures as well to take advantage of the ability to encapsulate SQL that is shared among different applications.

The decision to access distributed data has implications for many DB2 activities: application programming, data recovery, authorization, and so on.

DB2 and z/OS

z/OS is the next generation of the OS/390 operating system. z/OS and the IBM zSeries® 800, 900, 990, or the equivalent, offer architecture that provides qualities of service that are critical for e-business. The z/OS operating system is based on 64-bit z/Architecture™. The operating system is highly secure, scalable, and high performing. With these characteristics, z/OS provides a strong base for Internet and Java-enabled applications and a comprehensive and diverse environment for running your applications.

DB2 and the Parallel Sysplex

The Parallel Sysplex is a key example of the synergy of DB2 and zSeries. DB2 takes advantage of the zSeries Parallel Sysplex, with its superior processing capabilities. By allowing two or more processors to share the same data, you can maximize performance while minimizing cost; improve system availability and concurrency; expand system capacity; and configure your system environment more flexibly. With data sharing, applications running on more than one DB2 subsystem can read from and write to the same set of data concurrently.

Sharing DB2s must belong to a DB2 data sharing *group*. A data sharing group is a collection of one or more DB2 subsystems accessing shared DB2 data. Each DB2 subsystem belonging to a particular data sharing group is a *member* of that group. All members of a group use the same shared DB2 catalog and directory.

With data sharing, you can grow your system incrementally by adding additional central processor complexes and DB2s to the data sharing group. You don't have to move part of the workload onto another system, alleviating the need to manage copies of the data or to use distributed processing to access the data.

You can configure your environment flexibly. For example, you can tailor each z/OS image to meet the requirements for the user set on that image. For processing that occurs during peak workload periods, you can bring up a dormant DB2 to help process the work.

DB2 and the Security Server for z/OS

You can use the Resource Access Control Facility (RACF) component of the SecureWay® Security Server for z/OS, or an equivalent product, to control access to your z/OS system. When users begin sessions with TSO, IMS, or CICS, their identities are checked to prevent unauthorized access to the system.

Recommendation: Use the Security Server check the security of DB2 users and to protect DB2 resources. The Security Server provides effective protection for DB2 data by permitting only DB2-mediated access to DB2 data sets.

Much authorization to DB2 objects can be controlled directly from the Security Server. An *exit routine* (a program that runs as an extension of DB2) that is shipped with the z/OS Security Server lets you centralize access control.

DB2 and DFSMS

The DFSMSdfp™ *storage management subsystem* (SMS) can be used to manage DB2 disk data sets. The purpose of SMS is to automate as much as possible the management of physical storage by centralizing control, automating tasks, and providing interactive controls for system administrators. SMS can reduce users' needs to be concerned about physical details of performance, space, and device management.

Consult with your site's storage administrator about using SMS for DB2 private data, image copies, and archive logs. For data that is especially performance-sensitive, there might need to be more manual control over data set placement.

Table spaces or indexes with data sets larger than 4 gigabytes require SMS-managed data sets.

Extended partitioned data sets (PDSE), a feature of DFSMSdfp, are useful for managing stored procedures that run in a stored procedures address space. PDSE enables extent information for the load libraries to be dynamically updated, reducing the need to start and stop the stored procedures address space.

More information about the z/OS environment

Table 5 lists additional information sources about topics that this section introduces.

Table 5. More information about the z/OS environment

For more information about...	See...
z/OS	www.ibm.com/servers/eserver/zseries/zos/
IRLM installation	*DB2 Installation Guide*
IRLM commands	*DB2 Command Reference*
IRLM lock tracing	"Using the statistics and accounting traces to monitor locking" on page 871
Exit routines	Appendix B, "Writing exit routines," on page 1051
Security methods	Part 3, "Security and auditing," on page 123
PDSE data sets	*z/OS DFSMS: Using Data Sets*
SMS	"Using SMS to manage archive log data sets" on page 430
DFSMShsm™	"Managing DB2 data sets with DFSMShsm" on page 35
Attachment facilities, programming	Volume 2 of *DB2 Application Programming and SQL Guide*
CICS XRF	• "Extended recovery facility (XRF) toleration" on page 472 • *CICS Transaction Server for z/OS Operations and Utilities Guide*

Table 5. More information about the z/OS environment (continued)

For more information about...	See...
CICS connections	Chapter 17, "Monitoring and controlling DB2 and its connections," on page 363
CICS administration	*DB2 Installation Guide*
IMS XRF	• "Extended recovery facility (XRF) toleration" on page 472 • *IMS Administration Guide: System*
DL/I batch	Volume 2 of *DB2 Application Programming and SQL Guide*
DataPropagator NonRelational	*IMS DataPropagator: An Introduction*
ISPF	Volume 2 of *DB2 Application Programming and SQL Guide*
Distributed data	Volume 1 of *DB2 Application Programming and SQL Guide*
Parallel Sysplex data sharing	*DB2 Data Sharing: Planning and Administration*

Part 2. Designing a database: advanced topics

Chapter 2. Introduction to designing a database: advanced topics 25

Chapter 3. Creating storage groups and managing DB2 data sets 27
Managing data sets with DB2 storage groups . . . 27
Creating DB2 storage groups 29
Using SMS to manage DB2 storage groups . . . 30
Deferring allocation of DB2-managed data sets 30
Extending DB2-managed data sets. 31
Primary space allocation 32
Secondary space allocation 32
Example of primary and secondary space allocation 34
Managing DB2 data sets with DFSMShsm 35
Migrating to DFSMShsm 35
Recalling archive logs 36
Using DFSMShsm with the RECOVER utility . . 36
Using DFSMShsm with the BACKUP SYSTEM utility 37
Managing your own data sets 37
Requirements for your own data sets. 38
Using the DEFINE CLUSTER command 40
Extending user-managed data sets. 41
Defining index space storage 41
Creating EA-enabled table spaces and index spaces 42

Chapter 4. Implementing your database design 43
Implementing databases 43
Implementing table spaces 44
Creating a table space explicitly 44
Creating a table space implicitly 45
Choosing a page size 45
Choosing a page size for LOBs 46
Implementing tables 48
Distinctions between DB2 base tables and temporary tables 48
Implementing table-controlled partitioning . . . 51
Automatic conversion 52
Using nullable partitioning columns 53
Implementing indexes 54
Types of indexes. 54
Unique indexes 54
Clustering indexes 55
Partitioning indexes 55
Secondary indexes 56
Using the NOT PADDED clause for indexes with varying-length columns 57
Using indexes to avoid sorts. 58
Using schemas 58
Creating a schema 58
Authorization to process schema definitions . . 59
Processing schema definitions 59

Chapter 5. Loading data into DB2 tables 61
Loading tables with the LOAD utility 61
Making corrections after LOAD 63
Loading data using the SQL INSERT statement . . 63
Inserting a single row 64
Inserting multiple rows 64
Special considerations when using INSERT statement to load tables 64
Loading data from DL/I 65

Chapter 6. Altering your database design . . . 67
Altering DB2 storage groups. 67
Letting SMS manage your DB2 storage groups . 67
Adding or removing volumes from a DB2 storage group. 67
Altering DB2 databases 68
Altering table spaces 69
Changing the space allocation for user-managed data sets 69
Dropping, re-creating, or converting a table space 69
Altering tables 71
Adding a new column to a table 71
Altering the data type of a column 73
What happens to the column 73
What happens to an index on the column . . 74
Table space versions 76
Altering a table for referential integrity 77
Adding referential constraints to existing tables 77
Adding parent keys and foreign keys. . . . 78
Dropping parent keys and foreign keys . . . 79
Adding or dropping table check constraints . . 80
Adding a partition 80
Altering partitions 82
Changing the boundary between partitions. . 82
Rotating partitions 83
Extending the boundary of the last partition 84
Registering or changing materialized query tables 85
Registering an existing table as a materialized query table 85
Changing a materialized query table to a base table. 86
Changing the attributes of a materialized query table 86
Changing the definition of a materialized query table 86
Altering the assignment of a validation routine 86
Altering a table for capture of changed data . . 87
Changing an edit procedure or a field procedure 87
Altering the subtype of a string column 88
Altering the attributes of an identity column . . 88
Changing data types by dropping and re-creating the table 88
Implications of dropping a table 89
Check objects that depend on the table . . . 90
Re-creating a table 90

© Copyright IBM Corp. 1982, 2008

Moving a table to a table space of a different page size . . . 91
Altering indexes . . . 92
Adding a new column to an index . . . 92
Altering how varying-length index columns are stored . . . 93
Altering the clustering index . . . 93
Rebalancing data in partitioned table spaces . . . 93
Dropping and redefining an index . . . 94
Index versions . . . 94
Altering views . . . 96
Altering stored procedures . . . 96
Altering user-defined functions . . . 96
Moving from index-controlled to table-controlled partitioning . . . 97
Changing the high-level qualifier for DB2 data sets 98
Defining a new integrated catalog alias . . . 99
Changing the qualifier for system data sets . . . 99
Step 1: Change the load module to reflect the new qualifier . . . 99
Step 2: Stop DB2 with no outstanding activity . . . 100
Step 3: Rename system data sets with the new qualifier . . . 101
Step 4: Update the BSDS with the new qualifier . . . 102
Step 5: Establish a new xxxxMSTR cataloged procedure . . . 102
Step 6: Start DB2 with the new xxxxMSTR and load module . . . 102
Changing qualifiers for other databases and user data sets . . . 102
Changing your work database to use the new high-level qualifier . . . 103
Changing user-managed objects to use the new qualifier . . . 104
Changing DB2-managed objects to use the new qualifier . . . 104
Moving DB2 data . . . 105
Tools for moving DB2 data . . . 106
Moving a DB2 data set . . . 108
Copying a relational database . . . 109
Copying an entire DB2 subsystem . . . 109

Chapter 7. Estimating disk storage for user data 111
Factors that affect storage . . . 111
Calculating the space required for a table . . . 112
Calculating record lengths and pages . . . 113
Saving space with data compression . . . 114
Estimating storage for LOBs . . . 114
Estimating storage when using the LOAD utility 114
Calculating the space required for a dictionary . . . 116
Disk requirements . . . 116
Virtual storage requirements . . . 117
Calculating the space required for an index . . . 117
Levels of index pages . . . 117
Estimating storage from number of index pages 118

Advanced topics include:

- Creating storage groups and managing DB2 data sets, which explores your options for allocating and managing data storage for table spaces and indexes
- Implementing your database design, which covers creating table spaces, tables, indexes, referential constraints, and views
- Loading data into DB2 tables, which provides an overview of the methods that you can use to load data into your DB2 tables
- Altering your database design, which addresses altering DB2 storage groups, databases, table spaces, tables, indexes, views, stored procedures, and user-defined functions
- Estimating disk storage for user data, which includes a discussion of the factors that affect storage and calculations for the space requirements of tables, dictionaries, and indexes

Chapter 2. Introduction to designing a database: advanced topics

Part 2, "Designing a database: advanced topics," on page 23 presents information about the current release of DB2 UDB for z/OS and selected advanced topics. *The Official Introduction to DB2 UDB for z/OS* covers basic information about designing and implementing a database.

Table 6 shows where you can find more information about topics related to designing a database.

Table 6. More information about designing a database

For more information about...	See...
Basic database design concepts for DB2 Universal Database for z/OS, including: • Designing tables and views • Designing columns • Designing indexes • Designing table spaces	*The Official Introduction to DB2 UDB for z/OS*
Maintaining data integrity, including: • Maintaining referential constraints • Defining table check constraints • Planning to use triggers	Part 2 of *DB2 Application Programming and SQL Guide*
Maintaining data integrity, including implications for the following SQL statements: INSERT, UPDATE, DELETE, and DROP	Chapter 5 of *DB2 SQL Reference*
Maintaining data integrity, including implications for the following utilities: COPY, QUIESCE, RECOVER, and REPORT	Part 2 of *DB2 Utility Guide and Reference*
Compressing data in a table space or a partition	Part 5 (Volume 2) of *DB2 Administration Guide*
Designing and using materialized query tables	Part 5 (Volume 2) of *DB2 Administration Guide*

© Copyright IBM Corp. 1982, 2008

Chapter 3. Creating storage groups and managing DB2 data sets

A *DB2 storage group* is a named set of disk volumes, in which DB2:
- Allocates storage for table spaces and indexes
- Defines the necessary VSAM data sets
- Extends and deletes VSAM data sets
- Alters VSAM data sets

DB2 can manage the auxiliary storage requirements of a database by using DB2 storage groups. Data sets in these DB2 storage groups are called *DB2-managed data sets*.

Note: These DB2 storage groups are not the same as storage groups that are defined by the DFSMS storage management subsystem (DFSMSsms).

You have several options for managing DB2 data sets:
- Let DB2 manage the data sets. This option means less work for DB2 database administrators. See "Managing data sets with DB2 storage groups" for more information.
- Let SMS manage some or all of the data sets, either when you use DB2 storage groups or when you use data sets that you have defined yourself. This option offers a reduced workload for DB2 database administrators and storage administrators. See "Managing DB2 data sets with DFSMShsm" on page 35 for complete information.
- Define and manage your own data sets using VSAM Access Method Services. This option gives you the most control over the physical storage of tables and indexes. See "Managing your own data sets" on page 37 for detailed information.

Recommendation: Use DB2 storage groups whenever you can, either specifically or by default.

This chapter also includes the following additional topics:
- "Defining index space storage" on page 41
- "Creating EA-enabled table spaces and index spaces" on page 42

Managing data sets with DB2 storage groups

This section describes how to create DB2 storage groups, how DB2 manages the data sets in your DB2 storage groups, and what you should consider when using storage groups :
- "Creating DB2 storage groups" on page 29
- "Using SMS to manage DB2 storage groups" on page 30
- "Deferring allocation of DB2-managed data sets" on page 30
- "Extending DB2-managed data sets" on page 31

Here are some of the things that DB2 does for you in managing your auxiliary storage requirements:
- When a table space is created, DB2 defines the necessary VSAM data sets using VSAM Access Method Services. After the data sets are created, you can process

© Copyright IBM Corp. 1982, 2008

them with access method service commands that support VSAM control-interval (CI) processing (for example, IMPORT and EXPORT).

Exception: You can defer the allocation of data sets for table spaces and index spaces by specifying the DEFINE NO clause on the associated statement (CREATE TABLESPACE and CREATE INDEX), which also must specify the USING STOGROUP clause. For more information about deferring data set allocation, see either "Deferring allocation of DB2-managed data sets" on page 30 or Chapter 5 of *DB2 SQL Reference*.

- When a table space is dropped, DB2 automatically deletes the associated data sets.
- When a data set in a segmented or simple table space reaches its maximum size of 2 GB, DB2 might automatically create a new data set. The primary data set allocation is obtained for each new data set.
- When needed, DB2 can extend individual data sets. For more information, see "Extending DB2-managed data sets" on page 31.
- When you create or reorganize a table space that has associated data sets, DB2 deletes and then redefines them, reclaiming fragmented space. However, when you run REORG with the REUSE option and SHRLEVEL NONE, REORG resets and reuses DB2-managed data sets without deleting and redefining them. If the size of your table space is not changing, using the REUSE parameter could be more efficient.

  Exception: When reorganizing a LOB table space, DB2 does not delete and redefine the first data set that was allocated for the table space. If the REORG results in empty data sets beyond the first data set, DB2 deletes those empty data sets.
- When you want to move data sets to a new volume, you can alter the volumes list in your storage group. DB2 automatically relocates your data sets during the utility operations that build or rebuild a data set (LOAD REPLACE, REORG, REBUILD, and RECOVER). Note that if you use the REUSE option, DB2 does not delete and redefine the data sets and therefore does not move them.

  For a LOB table space, you can alter the volumes list in your storage group, and DB2 automatically relocates your data sets during the utility operations that build or rebuild a data set (LOAD REPLACE and RECOVER).

  To move user-defined data sets, you must delete and redefine the data sets in another location. For information about defining your own data sets, see "Managing your own data sets" on page 37.

Control interval sizing: A *control interval* is a fixed-length area or disk in which VSAM stores records and creates distributed free space. It is the unit of information that VSAM transmits to or from disk.

DB2 page sets are defined as VSAM linear data sets. Prior to Version 8, DB2 defined all data sets with VSAM control intervals that were 4 KB in size. Beginning in Version 8, DB2 can define data sets with variable VSAM control intervals. One of the biggest benefits of this change is an improvement in query processing performance.

The VARY DS CONTROL INTERVAL parameter on installation panel DSNTIP7 allows you to control whether DB2–managed data sets have variable VSAM control intervals:

- A value of YES indicates that a DB2–managed data set is created with a VSAM control interval that corresponds to the size of the buffer pool that is used for the table space. This is the default value.

- A value of NO indicates that a DB2–managed data set is created with a fixed VSAM control interval of 4 KB, regardless of the size of the buffer pool that is used for the table space.

Table 7 shows the default and compatible control interval sizes for each table space page size. For example, a table space with pages 16 KB in size can have a VSAM control interval of 4 KB or 16 KB. Control interval sizing has no impact on indexes; index pages are always 4 KB in size.

Table 7. Default and compatible control interval sizes

Table space page size	Default control interval size	Compatible control interval sizes
4 KB	4 KB	4 KB
8 KB	8 KB	4 KB, 8 KB
16 KB	16 KB	4 KB, 16 KB
32 KB	32 KB	4 KB, 32 KB

Creating DB2 storage groups

To create a DB2 storage group, use the SQL statement CREATE STOGROUP. For detailed information about CREATE STOGROUP, see Chapter 5 of *DB2 SQL Reference*.

DB2 storage group names are unqualified identifiers of up to 128 characters. A DB2 storage group name cannot be the same as any other storage group name in the DB2 catalog.

After you define a storage group, DB2 stores information about it in the DB2 catalog. (This catalog is not the same as the integrated catalog facility catalog that describes DB2 VSAM data sets). The catalog table SYSIBM.SYSSTOGROUP has a row for each storage group, and SYSIBM.SYSVOLUMES has a row for each volume. With the proper authorization, you can retrieve the catalog information about DB2 storage groups by using SQL statements. See Appendix F of *DB2 SQL Reference* for more information about using SQL statements to retrieve catalog information about DB2 storage groups.

When you create table spaces and indexes, you name the storage group from which space is to be allocated. You can also assign an entire database to a storage group. Try to assign frequently accessed objects (indexes, for example) to fast devices, and assign seldom-used tables to slower devices. This approach to choosing storage groups improves performance.

A *default storage group*, SYSDEFLT, is defined when DB2 is installed. If you are authorized and do not take specific steps to manage your own storage, you can still define tables, indexes, table spaces, and databases; DB2 uses SYSDEFLT to allocate the necessary auxiliary storage. Information about SYSDEFLT, as with any other storage group, is kept in the catalog tables SYSIBM.SYSSTOGROUP and SYSIBM.SYSVOLUMES.

For both user-managed and DB2-managed data sets, you need at least one integrated catalog facility (ICF) catalog—either user or master—that is created with the ICF. You must identify the catalog of the ICF when you create a storage group or when you create a table space that does not use storage groups.

Using SMS to manage DB2 storage groups

When defining DB2 storage groups, use the VOLUMES('*') attribute on the CREATE STOGROUP statement to let SMS control the selection of volumes during allocation. See "Managing DB2 data sets with DFSMShsm" on page 35 for more information.

If you use DB2 to allocate data to specific volumes, you must assign an SMS storage class with guaranteed space, and you must manage free space for each volume to prevent failures during the initial allocation and extension. Using guaranteed space reduces the benefits of SMS allocation, requires more time for space management, and can result in more space shortages. You should only use guaranteed space when space needs are relatively small and do not change.

Recommendation: Let SMS manage your DB2 storage groups; to do this, use asterisks (nonspecific volume IDs) in the VOLUMES clause.

Deferring allocation of DB2-managed data sets

When you execute a CREATE TABLESPACE statement with the USING STOGROUP clause, DB2 generally defines the necessary VSAM data sets for the table space. In some cases, however, you might want to define a table space without immediately allocating the associated data sets.

For example, you might be installing a software program that requires that many table spaces be created, but your company might not need to use some of those table spaces; you might prefer not to allocate data sets for the table spaces you will not be using.

To defer the physical allocation of DB2-managed data sets, use the DEFINE NO clause of the CREATE TABLESPACE statement. When you specify the DEFINE NO clause, the table space is created, but DB2 does not allocate (that is, define) the associated data sets until a row is inserted or loaded into a table in that table space. The DB2 catalog table SYSIBM.SYSTABLESPART contains a record of the created table space and an indication that the data sets are not yet allocated.

The DEFINE NO clause is not allowed for LOB table spaces, for table spaces in a work file database or a TEMP database, or for user-defined data sets. (In the case of user-defined data sets, the table space is created with the USING VCAT clause of the CREATE TABLESPACE statement).

Using the DEFINE NO clause is recommended when:
- Performance of the CREATE TABLESPACE statement is important.
- Disk resource is constrained.

Do not use the DEFINE NO clause on a table space if you plan to use a tool outside of DB2 to propagate data into a data set in the table space. When you use DEFINE NO, the DB2 catalog indicates that the data sets have not yet been allocated for that table space. Then, if data is propagated from a tool outside of DB2 into a data set in the table space, the DB2 catalog information does not reflect the fact that the data set has been allocated. The resulting inconsistency causes DB2 to deny application programs access to the data until the inconsistency is resolved.

Extending DB2-managed data sets

When a data set is created, DB2 allocates a primary allocation space on a volume that has space available and that is specified in the DB2 storage group. Any extension to a data set always gets a secondary allocation space.

If new extensions reach the end of the volume, DB2 accesses all candidate volumes from the DB2 storage group and issues the Access Method Services command ALTER ADDVOLUMES to add these volumes to the integrated catalog facility (ICF) catalog as candidate volumes for the data set. DB2 then makes a request to allocate a secondary extent on any one of the candidate volumes that has space available. After the allocation is successful, DB2 issues the command ALTER REMOVEVOLUMES to remove all candidate volumes from the ICF catalog for the data set.

DB2 extends data sets when either of the following conditions occurs:

- The requested space exceeds the remaining space in the data set.
- 10% of the secondary allocation space (but not over 10 allocation units, based on either tracks or cylinders) exceeds the remaining space.

If DB2 fails to extend a data set with a secondary allocation space because of insufficient available space on any single candidate volume of a DB2 storage group, DB2 tries again to extend with the requested space if the requested space is smaller than the secondary allocation space. Typically, DB2 requests only one additional page. In this case, a small amount of two units (tracks or cylinders, as determined by dfSMS based on the SECQTY value) is allocated. To monitor data set extension activity, use IFCID 258 in statistics class 3.

Extending nonpartitioned spaces: For a nonpartitioned table space or a nonpartitioned index space, DB2 defines the first piece of the page set starting with a primary allocation space, and extends that piece by using secondary allocation spaces. When the end of the first piece is reached, DB2 defines a new piece (which is a new data set) and extends that new piece starting with a primary allocation space.

Exception: In the case where the OPTIMIZE EXTENT SIZING parameter (MGEXTSZ) on installation panel DSNTIP7 is set to YES and the SECQTY value for the table space or index space is greater than zero, the primary space allocation of each subsequent data set is the larger of the SECQTY setting and the value that is derived from a sliding scale algorithm. See “Secondary space allocation” on page 32 for information about the sliding scale algorithm.

Extending partitioned spaces: For a partitioned table space or a partitioned index space, each partition is a data set. Therefore, DB2 defines each partition with the primary allocation space and extends each partition’s data set by using a secondary allocation space, as needed.

When data extension fails: If a data set uses all possible extents, DB2 cannot extend that data set. For a partitioned page set, the extension fails only for the particular partition that DB2 is trying to extend. For nonpartitioned page sets, DB2 cannot extend to a new data set piece, which means that the extension for the entire page set fails.

To avoid extension failures, allow DB2 to use the default value for primary space allocation and to use a sliding scale algorithm for secondary extent allocations. These concepts are discussed in the topics that follow.

Primary space allocation

By default, DB2 uses the following values for primary space allocation of DB2–managed data sets:

- 1 cylinder (720 KB) for non-LOB table spaces
- 10 cylinders for LOB table spaces
- 1 cylinder for indexes

To indicate that you want DB2 to use the default values for primary space allocation of table spaces and indexes, specify a value of 0 for the following parameters on installation panel DSNTIP7, as shown in Table 8.

Table 8. DSNTIP7 parameter values for managing space allocations

Installation panel DSNTIP7 parameter	Recommended value
TABLE SPACE ALLOCATION	0
INDEX SPACE ALLOCATION	0

Thereafter:

- On CREATE TABLESPACE and CREATE INDEX statements, do not specify a value for the PRIQTY option.
- On ALTER TABLESPACE and ALTER INDEX statements, specify a value of -1 for the PRIQTY option.

Primary space allocation quantities do not exceed DSSIZE or PIECESIZE clause values.

Exception: If the OPTIMIZE EXTENT SIZING parameter (MGEXTSZ) on installation panel DSNTIP7 is set to YES and the table space or index space has a SECQTY setting of greater than zero, the primary space allocation of each subsequent data set is the larger of the SECQTY setting and the value that is derived from a sliding scale algorithm. See “Secondary space allocation” for information about the sliding scale algorithm.

For those situations in which the default primary quantity value is not large enough, you can specify a larger value for the PRIQTY option when creating or altering table spaces and indexes. DB2 always uses a PRIQTY value if one is explicitly specified.

If you want to prevent DB2 from using the default value for primary space allocation of table spaces and indexes, specify a non-zero value for the TABLE SPACE ALLOCATION and INDEX SPACE ALLOCATION parameters on installation panel DSNTIP7.

Secondary space allocation

DB2 can calculate the amount of space to allocate to secondary extents by using a sliding scale algorithm. The first 127 extents are allocated in increasing size, and the remaining extents are allocated based on the initial size of the data set:

- For 32 GB and 64 GB data sets, each extent is allocated with a size of 559 cylinders.
- For data sets that range in size from less than 1 GB to 16 GB, each extent is allocated with a size of 127 cylinders.

This approach has several advantages:

- It minimizes the potential for wasted space by increasing the size of secondary extents slowly at first.

- It prevents very large allocations for the remaining extents, which would likely cause fragmentation.
- It does not require users to specify SECQTY values when creating and altering table spaces and index spaces.
- It is theoretically possible to reach maximum data set size without running out of secondary extents.

In the case of severe DASD fragmentation, it can take up to 5 extents to satisfy a logical extent request. In this situation, the data set does not reach the theoretical data set size.

If you installed DB2 on the operating system z/OS Version 1 Release 7, or later, then you can modify the Extent Constraint Removal option. By setting the Extent Constraint Removal option to YES in the SMS data class, the maximum number of extents can be up to 7257. However, the limits of 123 extents per volume and a maximum volume count of 59 per data set remain valid. For more information, see "Using VSAM extent constraint removal" in the z/OS V1R7 guide "DFSMS™: Using the New Functions" (order number SC26-7473-02).

Maximum allocation is shown in Table 9. This table assumes that the initial extent that is allocated is one cylinder in size.

Table 9. Maximum allocation of secondary extents

Maximum data set size, in GB	Maximum allocation, in cylinders	Extents required to reach full size
1	127	54
2	127	75
4	127	107
8	127	154
16	127	246
32	559	172
64	559	255

DB2 uses a sliding scale for secondary extent allocations of table spaces and indexes when:

- You do not specify a value for the SECQTY option of a CREATE TABLESPACE or CREATE INDEX statement
- You specify a value of -1 for the SECQTY option of an ALTER TABLESPACE or ALTER INDEX statement.

Otherwise, DB2 always uses a SECQTY value for secondary extent allocations, if one is explicitly specified.

Exception: For those situations in which the calculated secondary quantity value is not large enough, you can specify a larger value for the SECQTY option when creating or altering table spaces and indexes. However, in the case where the OPTIMIZE EXTENT SIZING parameter is set to YES and you specify a value for the SECQTY option, DB2 uses the value of the SECQTY option to allocate a secondary extent only if the value of the option is larger than the value that is derived from the sliding scale algorithm. The calculation that DB2 uses to make this determination is:

```
Actual secondary extent size = max ( min ( ss_extent, MaxAlloc ), SECQTY )
```

In this calculation, *ss_extent* represents the value that is derived from the sliding scale algorithm, and *MaxAlloc* is either 127 or 559 cylinders, depending on the maximum potential data set size. This approach allows you to reach the maximum page set size faster. Otherwise, DB2 uses the value that is derived from the sliding scale algorithm.

If you do not provide a value for the secondary space allocation quantity, DB2 calculates a secondary space allocation value equal to 10% of the primary space allocation value and subject to the following conditions:

- The value cannot be less than 127 cylinders for data sets that range in initial size from less than 1 GB to 16 GB, and cannot be less than 559 cylinders for 32 GB and 64 GB data sets.
- The value cannot be more than the value that is derived from the sliding scale algorithm.

The calculation that DB2 uses for the secondary space allocation value is:

```
Actual secondary extent size = max ( 0.1 × PRIQTY, min ( ss_extent, MaxAlloc ) )
```

In this calculation, *ss_extent* represents the value that is derived from the sliding scale algorithm, and *MaxAlloc* is either 127 or 559 cylinders, depending on the maximum potential data set size.

Secondary space allocation quantities do not exceed DSSIZE or PIECESIZE clause values.

If you do not want DB2 to extend a data set, you can specify a value of 0 for the SECQTY option. Specifying 0 is a useful way to prevent DSNDB07 work files from growing out of proportion.

If you want to prevent DB2 from using the sliding scale for secondary extent allocations of table spaces and indexes, specify a value of NO for the OPTIMIZE EXTENT SIZING parameter on installation panel DSNTIP7.

Example of primary and secondary space allocation

Table 10 shows how primary and secondary quantities are affected by a CREATE statement and two subsequent ALTER statements. This example assumes a maximum data set size of less than 32 GB, and the following parameter values on installation panel DSNTIP7:

TABLE SPACE ALLOCATION = 0

INDEX SPACE ALLOCATION = 0

OPTIMIZE EXTENT SIZING = YES

Table 10. Example of specified and actual space allocations

Action	Specified PRIQTY	Actual primary quantity allocated	Specified SECQTY	Actual secondary quantity allocated
CREATE TABLESPACE	100 KB	100 KB	1000 KB	2 cylinders
ALTER TABLESPACE	-1	1 cylinder	2000 KB	3 cylinders
ALTER TABLESPACE		1 cylinder	-1	1 cylinder

Managing DB2 data sets with DFSMShsm

This section describes how you can use the Hierarchical Storage Management functional component (DFSMShsm) of DFSMS to manage space and data availability among the storage devices in your system:

- "Migrating to DFSMShsm"
- "Recalling archive logs" on page 36
- "Using DFSMShsm with the RECOVER utility" on page 36
- "Using DFSMShsm with the BACKUP SYSTEM utility" on page 37

You can use DFSMShsm to move data sets that have not been recently used to slower, less expensive storage devices. Moving the data sets helps to ensure that disk space is managed efficiently. For more information about using DFSMShsm to manage DB2 data sets, see *MVS Storage Management Library: Storage Management Subsystem Migration Planning Guide* and *z/OS DFSMShsm Managing Your Own Data*.

Migrating to DFSMShsm

If you decide to use DFSMShsm for your DB2 data sets, you should develop a migration plan with your system administrator. With user-managed data sets, you can specify DFSMShsm classes on the Access Method Services DEFINE command. With DB2 storage groups, you need to develop automatic class selection routines.

General-use Programming Interface

To allow DFSMShsm to manage your DB2 storage groups, you can use one or more asterisks as volume IDs in your CREATE STOGROUP or ALTER STOGROUP statement, as shown here:

```
CREATE STOGROUP G202
VOLUMES ('*')
VCAT DB2SMST;
```

End of General-use Programming Interface

This example causes all database data set allocations and definitions to use nonspecific selection through DFSMShsm filtering services.

When you use DFSMShsm and DB2 storage groups, you can use the system parameters SMSDCFL and SMSDCIX to assign table spaces and indexes to different DFSMShsm data classes.

- SMSDCFL specifies a DFSMShsm data class for table spaces. If you assign a value to SMSDCFL, DB2 specifies that value when it uses Access Method Services to define a data set for a table space.
- SMSDCIX specifies a DFSMShsm data class for indexes. If you assign a value to SMSDCIX, DB2 specifies that value when it uses Access Method Services to define a data set for an index.

Before you set the data class system parameters, you need to do two things:

- Define the data classes for your table space data sets and index data sets.
- Code the SMS automatic class selection (ACS) routines to assign indexes to one SMS storage class and to assign table spaces to a different SMS storage class.

For more information about creating data classes, see *z/OS DFSMS: Implementing System-Managed Storage*.

Recalling archive logs

DFSMShsm can automatically migrate and recall archive log and image copy data sets. If DB2 needs an archive log data set or an image copy data set that DFSMShsm has migrated, a recall begins automatically and DB2 waits for the recall to complete before continuing.

For processes that read more than one archive log data set, such as the RECOVER utility, DB2 anticipates a DFSMShsm recall of migrated archive log data sets. When a DB2 process finishes reading one data set, it can continue with the next data set without delay, because the data set might already have been recalled by DFSMShsm.

If you accept the default value YES for the RECALL DATABASE parameter on the Operator Functions panel (DSNTIPO), DB2 also recalls migrated table spaces and index spaces. At data set open time, DB2 waits for DFSMShsm to perform the recall. You can specify the amount of time DB2 waits while the recall is being performed with the RECALL DELAY parameter, which is also on panel DSNTIPO. If RECALL DELAY is set to zero, DB2 does not wait, and the recall is performed asynchronously.

You can use System Managed Storage (SMS) to archive DB2 subsystem data sets, including the DB2 catalog, DB2 directory, active logs, and work file databases (DSNDB07 in a non-data-sharing environment). However, before starting DB2, you should recall these data sets by using DFSMShsm. Alternatively, you can avoid migrating these data sets by assigning them to a management class that prevents migration. Considerations for using DFSMShsm for archive log data sets are discussed in “Archive log data sets” on page 428.

If a volume has a STOGROUP specified, you must recall that volume only to volumes of the same device type as others in the STOGROUP.

In addition, you must coordinate the DFSMShsm automatic purge period, the DB2 log retention period, and MODIFY utility usage. Otherwise, the image copies or logs that you might need during a recovery could already have been deleted.

Using DFSMShsm with the RECOVER utility

The RECOVER utility can execute the DFSMSdss™ RESTORE command, which generally uses extensions larger than the data set’s primary and secondary space allocation values. RECOVER executes this command if the point of recovery is defined by an image copy that was taken by using the CONCURRENT option of the COPY utility.

The DFSMSdss RESTORE command extends a data set differently than DB2, so you must alter the page set to contain extents defined by DB2. To do this, use ALTER TABLESPACE to enlarge the primary and secondary space allocation values for DB2–managed data sets. After you use ALTER TABLESPACE, the new values take effect only when you use REORG or LOAD REPLACE. Using RECOVER again does not resolve the extent definition.

For user-defined data sets, define the data sets with larger primary and secondary space allocation values (see “Managing your own data sets” on page 37).

Using DFSMShsm with the BACKUP SYSTEM utility

If you plan to use the DB2 BACKUP SYSTEM online utility to take volume-level copies of the data and logs of a non-data-sharing DB2 subsystem or a DB2 data sharing group, all of the DB2 data sets must reside on volumes that are managed by DFSMShms.

The BACKUP SYSTEM utility uses copy pools, which are new constructs in z/OS DFSMShsm Version 1 Release 5. A *copy pool* is a named set of storage groups that can be backed up and restored as a unit; DFSMShsm processes the storage groups collectively for fast replication. Each DB2 subsystem has up to two copy pools, one for databases and one for logs.

Copy pools are also referred to as source storage groups. Each source storage group contains the name of an associated copy-pool backup storage group, which contains eligible volumes for the backups. The storage administrator must define both the source and target storage groups, and use the following DB2 naming convention:

DSN$locn-name$cp-type

The variables that are used in this naming convention are described in Table 11:

Table 11. Naming convention variables

Variable	Meaning
DSN	The unique DB2 product identifier
$	A delimiter. You must use the dollar sign ($) character.
locn-name	The DB2 location name
cp-type	The copy pool type. Use DB for database and LG for log

For detailed instructions on how to create storage groups, see the *z/OS DFSMSdss Storage Administration Reference*.

The DB2 BACKUP SYSTEM and RESTORE SYSTEM utilities invoke DFSMShsm to back up and restore the copy pools. DFSMShsm interacts with DFSMSsms to determine the volumes that belong to a given copy pool so that the volume-level backup and restore functions can be invoked.

For information about the BACKUP SYSTEM and RESTORE SYSTEM utilities, see the *DB2 Utility Guide and Reference*. For information about recovery procedures that use these utilities, see "System-level point in time recovery" on page 511.

Managing your own data sets

This section describes how to manage your own data sets by using VSAM Access Method Services and what you should consider:

- "Requirements for your own data sets" on page 38
- "Using the DEFINE CLUSTER command" on page 40
- "Extending user-managed data sets" on page 41

You might choose to manage your own VSAM data sets for reasons such as these:

- You have a large linear table space on several data sets. If you manage your own data sets, you can better control the placement of individual data sets on the volumes (although you can keep a similar type of control by using single-volume DB2 storage groups).
- You want to prevent deleting a data set within a specified time period, by using the TO and FOR options of the Access Method Services DEFINE and ALTER commands. You can create and manage the data set yourself, or you can create the data set with DB2 and use the ALTER command of Access Method Services to change the TO and FOR options.
- You are concerned about recovering dropped table spaces. Your own data set is not automatically deleted when a table space is dropped, making it easier to reclaim the data.

To define the required data sets, use DEFINE CLUSTER; to add secondary volumes to expanding data sets, use ALTER ADDVOLUMES; and to delete data sets, use DELETE CLUSTER.

You must define a data set for each of these items:
- A simple or segmented table space
- A partition of a partitioned table space
- A partition of a partitioned index

Furthermore, as table spaces and index spaces expand, you might need to provide additional data sets. To take advantage of parallel I/O streams when doing certain read-only queries, consider spreading large table spaces over different disk volumes that are attached on separate channel paths.

Requirements for your own data sets

DB2 checks whether you have defined your data sets correctly. If you plan to define and manage VSAM data sets yourself, you must perform these steps:

1. Define the data sets **before** you issue the CREATE TABLESPACE, the CREATE INDEX statement, or the ALTER TABLE ADD PARTITION statement.

   If you create a partitioned table space, you must create a separate data set for each partition, or you must allocate space for each partition by using the PARTITION option of the NUMPARTS clause in the CREATE TABLESPACE statement.

   If you create a partitioned secondary index, you must create a separate data set for each partition. Alternatively, for DB2 to manage your data sets, you must allocate space for each partition by using the PARTITIONED option of the CREATE INDEX statement. For information about the performance of partitioned secondary indexes, see “Recommendations for database design” on page 813.

   If you create a partitioning index that is partitioned, you must create a separate data set for each partition. Alternatively, for DB2 to manage your data sets, you must allocate space for each partition by using the PARTITIONED option or the PARTITION ENDING AT clause of the CREATE INDEX statement in the case of index-controlled partitioning.
2. Give each data set a name with this format:

   `catname.DSNDBx.dbname.psname.y0001.znnn`

   catname
   Integrated catalog name or alias (up to eight characters). Use the same name or alias here as in the USING VCAT clause of the CREATE TABLESPACE and CREATE INDEX statements.

x C (for VSAM clusters) or D (for VSAM data components).

dbname

DB2 database name. If the data set is for a table space, *dbname* must be the name given in the CREATE TABLESPACE statement. If the data set is for an index, *dbname* must be the name of the database containing the base table. If you are using the default database, *dbname* must be DSNDB04.

psname

Table space name or index name. This name must be unique within the database.

You use this name on the CREATE TABLESPACE or CREATE INDEX statement. (You can use a name longer than eight characters on the CREATE INDEX statement, but the first eight characters of that name must be the same as in the data set's *psname*.)

y0001 Instance qualifier for the data set.

Define one data set for the table space or index with a value of I for *y* if one of the following conditions is true:

- You plan to run REORG with SHRLEVEL CHANGE or SHRLEVEL REFERENCE without the FASTSWITCH YES option.
- You *do not* plan to run REORG with SHRLEVEL CHANGE or SHRLEVEL REFERENCE.

Define two data sets if you plan to run REORG, using the FASTSWITCH YES option, with SHRLEVEL CHANGE or SHRLEVEL REFERENCE. Define one data set with a value of I for *y*, and one with a value of J for *y*.

For more information about defining data sets for REORG, see Part 2 of *DB2 Utility Guide and Reference*.

znnn Data set number. The first digit *z* of the data set number is represented by the letter A, B, C, D, or E, which corresponds to the value 0, 1, 2, 3, or 4 as the first digit of the partition number.

For partitioned table spaces, if the partition number is less than 1000, the data set number is A*nnn* in the data set name (for example, A999 represents partition 999). For partitions 1000 to 1999, the data set number is B*nnn* (for example, B000 represents partition 1000). For partitions 2000 to 2999, the data set number is C*nnn*. For partitions 3000 to 3999, the data set number is D*nnn*. For partitions 4000 up to a maximum of 4096, the data set number is E*nnn*.

The naming convention for data sets that you define for a partitioned index is the same as the naming convention for other partitioned objects.

For simple or segmented table spaces, the number is 001 (preceded by A) for the first data set. When little space is available, DB2 issues a warning message. If the size of the data set for a simple or a segmented table space approaches the maximum limit, define another data set with the same name as the first data set and the number 002. The next data set will be 003, and so on.

You can reach the VSAM extent limit for a data set before you reach the size limit for a partitioned or a nonpartitioned table space. If this happens, DB2 does not extend the data set.

3. Use the DEFINE CLUSTER command to define the size of the primary and secondary extents of the VSAM cluster. If you specify zero for the secondary extent size, data set extension does not occur.
4. Define the data sets as LINEAR. Do not use RECORDSIZE or CONTROLINTERVALSIZE; these attributes are invalid.
5. Use the REUSE option. You must define the data set as REUSE before running the DSN1COPY utility.
6. Use SHAREOPTIONS(3,3).

The DEFINE CLUSTER command has many optional parameters that do not apply when DB2 uses the data set. If you use the parameters SPANNED, EXCEPTIONEXIT, SPEED, BUFFERSPACE, or WRITECHECK, VSAM applies them to your data set, but DB2 ignores them when it accesses the data set.

The value of the OWNER parameter for clusters that are defined for storage groups is the first SYSADM authorization ID specified at installation.

When you drop indexes or table spaces for which you defined the data sets, you must delete the data sets unless you want to reuse them. To reuse a data set, first commit, and then create a new table space or index with the same name. When DB2 uses the new object, it overwrites the old information with new information, which destroys the old data.

Likewise, if you delete data sets, you must drop the corresponding table spaces and indexes; DB2 does not drop these objects automatically.

Using the DEFINE CLUSTER command

Figure 3 shows an example of the DEFINE CLUSTER command, which defines a VSAM data set for the SYSUSER table space in database DSNDB06. Assume that an integrated catalog facility catalog named DSNCAT is already defined.

```
DEFINE CLUSTER -
       (NAME(DSNCAT.DSNDBC.DSNDB06.SYSUSER.I0001.A001) -
        LINEAR                                         -
        REUSE                                          -
        VOLUMES(DSNV01)                                -
        RECORDS(100 100)                               -
        SHAREOPTIONS(3 3) )                            -
      DATA                                             -
       (NAME(DSNCAT.DSNDBD.DSNDB06.SYSUSER.I0001.A001) -
    CATALOG(DSNCAT)
```

Figure 3. Defining a VSAM data set for the SYSUSER table space

For user-managed data sets, you must pre-allocate shadow data sets prior to running REORG with SHRLEVEL CHANGE, REORG with SHRLEVEL REFERENCE, or CHECK INDEX with SHRLEVEL CHANGE against the table space. You can specify the MODEL option for the DEFINE CLUSTER command so that the shadow is created like the original data set, as shown in Figure 4 on page 41.

```
DEFINE CLUSTER -
       (NAME('DSNCAT.DSNDBC.DSNDB06.SYSUSER.x0001.A001') -
       MODEL('DSNCAT.DSNDBC.DSNDB06.SYSUSER.y0001.A001')) -
      DATA                                                -
       (NAME('DSNCAT.DSNDBD.DSNDB06.SYSUSER.x0001.A001') -
       MODEL('DSNCAT.DSNDBD.DSNDB06.SYSUSER.y0001.A001')) -
```

Figure 4. Defining shadow data sets

In Figure 4, the instance qualifiers *x* and *y* are distinct and are equal to either **I** or **J**. You must determine the correct instance qualifier to use for a shadow data set by querying the DB2 catalog for the database and table space.

For more information about defining data sets for REORG, see Chapter 2 of *DB2 Utility Guide and Reference*. For more information about defining and managing VSAM data sets, see *DFSMS/MVS: Access Method Services for the Integrated Catalog*.

Extending user-managed data sets

A user-managed data set is allocated by using only volumes defined for that data set in the ICF catalog. Before the current volume runs out of space, you must issue the Access Method Services commands ALTER ADDVOLUMES or ALTER REMOVEVOLUMES for candidate volumes.

Defining index space storage

Generally, the CREATE INDEX statement creates an index space in the same DB2 database that contains the table on which the index is defined. This is true even if you defer building the index.

Exceptions:

- If you specify the USING VCAT clause, you create and manage the data sets yourself.
- If you specify the DEFINE NO clause on a CREATE INDEX statement that uses the USING STOGROUP clause, DB2 defers the allocation of the data sets for the index space.

When you use CREATE INDEX, always specify a USING clause. When you specify USING, you declare whether you want DB2-managed or user-managed data sets. For DB2-managed data sets, you specify the primary and secondary space allocation parameters on the CREATE INDEX statement. If you do not specify USING, DB2 assigns the index data sets to the default storage groups using default space attributes. For information about how space allocation can affect the performance of mass inserts, see “Formatting early and speed up formatting” on page 658.

You can specify the USING clause to allocate space for the entire index, or if the index is a partitioned index, you can allocate space for each partition. Information about space allocation for the index is kept in the SYSIBM.SYSINDEXPART table of the DB2 catalog. Other information about the index is in SYSIBM.SYSINDEXES. For more information about determining the space required for an index, see “Calculating the space required for an index” on page 117. For more information about CREATE INDEX clauses, see Chapter 5 of *DB2 SQL Reference*.

Creating EA-enabled table spaces and index spaces

DFSMS has an extended-addressability function, which is necessary to create data sets that are larger than 4 GB. Therefore, the term for page sets that are enabled for extended addressability is *EA-enabled*. You must use EA-enabled table spaces or index spaces if you specify a DSSIZE that is larger than 4 GB in the CREATE TABLESPACE statement.

To create EA-enabled page sets, you must:

1. Use SMS to manage the data sets that are associated with the EA-enabled page sets.
2. Associate the data sets with a *data class* (an SMS construct) that specifies the extended format and extended addressability options.

   To make this association between data sets and the data class, use an automatic class selection (ACS) routine to assign the DB2 data sets to the relevant SMS data class. The ACS routine does the assignment based on the data set name. No performance penalty occurs for having non-EA-enabled DB2 page sets assigned to this data class, too, if you would rather not have two separate data classes for DB2.

   For user-managed data sets, you can use ACS routines or specify the appropriate data class on the DEFINE CLUSTER command when you create the data set.
3. Create the partitioned or LOB table space with a DSSIZE of 8 GB or greater. The partitioning index for the partitioned table space takes on the EA-enabled attribute from its associated table space. See *DB2 SQL Reference* for more information about the correct syntax.

   After a page set is created, you cannot use the ALTER TABLESPACE statement to change the DSSIZE. You must drop and re-create the table space.

   Also, you cannot change the data sets of the page set to turn off the extended addressability or extended format attributes. If someone modifies the data class to turn off the extended addressability or extended format attributes, DB2 issues an error message the next time that it opens the page set.

Chapter 4. Implementing your database design

The information in this chapter is General-use Programming Interface and Associated Guidance Information, as defined in "Notices" on page 1301.

This chapter provides information about topics related to implementing a database.

Table 12 shows where you can find more information about topics that are related to implementing a database design.

Table 12. More information about implementing a database design

For more information about...	See...
Basic concepts in implementing a database design for DB2 Universal Database for z/OS, including: • Choosing names for DB2 objects • Implementing databases • Implementing table spaces, including reorganizing data • Implementing tables • Implementing indexes • Implementing referential constraints • Implementing views	*The Official Introduction to DB2 UDB for z/OS*
Details about SQL statements used to implement a database design (CREATE and DECLARE, for example)	*DB2 SQL Reference*
Loading tables with referential constraints	*DB2 Utility Guide and Reference*
Using the catalog in database design	Appendix G of *DB2 SQL Reference*

The following topics provide additional information:

- "Implementing databases"
- "Implementing table spaces" on page 44
- "Implementing tables" on page 48
- "Implementing indexes" on page 54
- "Using schemas" on page 58

Implementing databases

In DB2 UDB for z/OS, a database is a logical collection of table spaces and index spaces. Consider the following factors when deciding whether to define a new database for a new set of objects:

- You can start and stop an entire database as a unit; you can display the statuses of all its objects by using a single command that names only the database. Therefore, place a set of tables that are used together into the same database. (The same database holds all indexes on those tables.)
- Some operations lock an entire database. For example, some phases of the LOAD utility prevent some SQL statements (CREATE, ALTER, and DROP) from using the same database concurrently. Therefore, placing many unrelated tables in a single database is often inconvenient.

© Copyright IBM Corp. 1982, 2008

When one user is executing a CREATE, ALTER, or DROP statement for a table, no other user can access the database that contains that table. QMF™ users, especially, might do a great deal of data definition; the QMF operations SAVE DATA and ERASE *data-object* are accomplished by creating and dropping DB2 tables. For maximum concurrency, create a separate database for each QMF user.

- The internal database descriptors (DBDs) might become inconveniently large; Part 2 of *DB2 Installation Guide* contains some calculations showing how the size depends on the number of columns in a table. DBDs grow as new objects are defined, but they do not immediately shrink when objects are dropped—the DBD space for a dropped object is not reclaimed until the MODIFY RECOVERY utility is used to delete records of obsolete copies from SYSIBM.SYSCOPY. DBDs occupy storage and are the objects of occasional input and output operations. Therefore, limiting the size of DBDs is another reason to define new databases. The MODIFY utility is described in Part 2 of *DB2 Utility Guide and Reference*.

If you use declared temporary tables, you must define a database that is defined AS TEMP (the TEMP database). DB2 stores all declared temporary tables in the TEMP database. The majority of the factors described in this section do not apply to the TEMP database. For details about declared temporary tables, see "Distinctions between DB2 base tables and temporary tables" on page 48.

Implementing table spaces

Table spaces are the physical spaces that hold tables. A table space can have one or more tables. Simple and segmented table spaces hold a maximum of 64 GB of data and might use one or more VSAM data sets. Partitioned table spaces that are created with the DSSIZE or LARGE option, and LOB table spaces can be larger.
| When you create a partitioned table space, DB2 puts the last partition of the table
| space into a REORG-pending (REORP) state.

Table spaces are divided into units called pages that are 4 KB, 8 KB, 16 KB, or 32
KB in size. As a general rule, you should have no more than 50 to 100 table spaces
in one DB2 database. Following this guideline helps to minimize maintenance,
increase concurrency, and decrease log volume.If you are using declared temporary
tables, you must create at least one table space in your TEMP database with pages
that are 8 KB in size.

You need to create additional table spaces if your database contains LOB data. For more information about creating table spaces for LOB data, see Chapter 5 of *DB2 SQL Reference*.

Data in most table spaces can be compressed, which can allow you to store more data on each data page. For more information, see "Compressing your data" on page 702.

Creating a table space explicitly

Use the CREATE TABLESPACE statement to create a table space explicitly. The statement allows you to specify the attributes of the table space.

Generally, when you use the CREATE TABLESPACE statement with the USING STOGROUP clause, DB2 allocates data sets for the table space. However, if you also specify the DEFINE NO clause, you can defer the allocation of data sets until data is inserted or loaded into a table in the table space. For more information about deferring data set allocation, see "Deferring allocation of DB2-managed data sets" on page 30.

You can create simple, segmented, partitioned, and LOB table spaces. For detailed information about CREATE TABLESPACE, see Chapter 5 of *DB2 SQL Reference*.

Creating a table space implicitly

As with DB2 storage groups and databases, you do not need to create a table space before you create a table unless you are defining a declared temporary table or managing all your own data sets. When you use CREATE TABLE, DB2 generates a table space for you. However, DB2 generates a table space only if you use CREATE TABLE without specifying an existing table space name. If the table contains a LOB column and SQLRULES are STD, DB2 also creates the LOB table space, the auxiliary table, and auxiliary index. If you do not specify a database name in the CREATE TABLE statement, DB2 uses the default database, DSNDB04, and the default DB2 storage group, SYSDEFLT.

If you create a table space implicitly, DB2 uses defaults for the space allocation attributes. The default values of PRIQTY and SECQTY specify the space allocation for the table space. If the value of the TSQTY subsystem parameter is nonzero, it determines the default values for PRIQTY and SECQTY. If the value of TSQTY is zero, the default values for PRIQTY and SECQTY are determined as described in the CREATE TABLESPACE statement in Chapter 5 of *DB2 SQL Reference*.

When you do not specify a table space name in a CREATE TABLE statement (and the table space is created implicitly), DB2 derives the table space name from the name of your table according to these rules:

- The table space name is the same as the table name if these conditions apply:
  - No other table space or index space in the database already has that name.
  - The table name has no more than eight characters.
  - The characters are all alphanumeric, and the first character is not a digit.
- If some other table space in the database already has the same name as the table, DB2 assigns a name of the form *xxxxnyyy*, where *xxxx* is the first four characters of the table name, and *nyyy* is a single digit and three letters that guarantees uniqueness.

DB2 stores this name in the DB2 catalog in the SYSIBM.SYSTABLESPACE table along with all your other table space names. The rules for LOB table spaces are explained in Chapter 5 of *DB2 SQL Reference*.

Choosing a page size

DB2 provides many options for data page sizes. The size of the data page is determined by the buffer pool in which you define the table space. For example, a table space that is defined in a 4-KB buffer pool has 4-KB page sizes, and one that is defined in an 8-KB buffer pool has 8-KB page sizes. (Indexes must be defined in a 4-KB buffer pool.)

Data in table spaces is stored and allocated in 4-KB record segments. Thus, an 8-KB page size means two 4-KB records, and a 32-KB page size means eight 4-KB records. A good starting point is to use the default of 4-KB page sizes when access to the data is random and only a few rows per page are needed. If row sizes are very small, using the 4-KB page size is recommended.

However, there are situations in which larger page sizes are needed or recommended:

- **When the size of individual rows is greater than 4 KB.** In this case, you must use a larger page size. When considering the size of work file table spaces,

remember that some SQL operations, such as joins, can create a result row that does not fit in a 4-KB page. Therefore, having at least one work file that has 32-KB pages is recommended. (Work files cannot use 8-KB or 16-KB pages.)

- **When you can achieve higher density on disk by choosing a larger page size.** For example, only one 2100-byte record can be stored in a 4-KB page, which wastes almost half of the space. However, storing the record in a 32-KB page can significantly reduce this waste. The downside with this approach is the potential of incurring higher buffer pool storage costs or higher I/O costs—if you only touch a small number of rows, you are bringing a bigger chunk of data from disk into the buffer pool.

  Using 8-KB or 16-KB page sizes can let you store more data on your disk with less impact on I/O and buffer pool storage costs. If you use a larger page size and access is random, you might need to go back and increase the size of the buffer pool to achieve the same read-hit ratio you do with the smaller page size.
- **When a larger page size can reduce data sharing overhead.** One way to reduce the cost of data sharing is to reduce the number of times the coupling facility must be accessed. Particularly for sequential processing, larger page sizes can reduce this number. More data can be returned on each access of the coupling facility, and fewer locks must be taken on the larger page size, further reducing coupling facility interactions.

  If data is returned from the coupling facility, each access that returns more data is more costly than those that return smaller amounts of data, but, because the total number of accesses is reduced, coupling facility overhead is reduced.

  For random processing, using an 8-KB or 16-KB page size instead of a 32-KB page size might improve the read-hit ratio to the buffer pool and reduce I/O resource consumption.

The maximum number of partitions for a table space depends on the page size and on the DSSIZE. The size of the table space depends on how many partitions are in the table space and on the DSSIZE. For specific information about the maximum number of partitions and the total size of the table space, given the page size and the DSSIZE, see the CREATE TABLESPACE statement in Chapter 5 of *DB2 SQL Reference*.

Choosing a page size for LOBs

Choosing a page size for LOBs (in the LOB table space) is a tradeoff between minimizing the number of getpages (maximizing performance) and not wasting space. With LOB table spaces, no more than one LOB value is ever stored in a given page in a LOB table space. Space that is not used by the LOB value in the last page that is occupied by the LOB remains unused. DB2 also uses additional space for control information. The smaller the LOB, the greater the proportion of space for this “non-data” is used.

For example, if you have a 17-KB LOB, the 4-KB page size is the most efficient for storage. A 17-KB LOB requires five 4-KB pages for a total of 20 KB of storage space. Pages that are 8 KB, 16 KB, and 32 KB in size waste more space, because they require 24 KB, 32 KB, and 32 KB, respectively, for the LOB.

Table 13 on page 47 shows that the number of data pages is lower for larger page sizes, but larger page sizes might have more unused space.

Table 13. Relationship between LOB size and data pages based on page size

LOB size	Page size	LOB data pages	% Non-LOB data or unused space
262 144 bytes	4 KB	64	1.6
	8 KB	32	3.0
	16 KB	16	5.6
	32 KB	8	11.1
4 MB	4 KB	1029	0.78
	8 KB	513	0.39
	16 KB	256	0.39
	32 KB	128	0.78
33 MB	4 KB	8234	0.76
	8 KB	4106	0.39
	16 KB	2050	0.19
	32 KB	1024	0.10

Choosing a page size based on average LOB size: If you know that all of your LOBs are not the same size, you can still make an estimate of what page size to choose. To estimate the average size of a LOB, you need to add a percentage to account for unused space and control information. To estimate the average size of a LOB value, use the following formula:

```
LOB size = (average LOB length) × 1.05
```

Table 14 contains some suggested page sizes for LOBs with the intent to reduce the amount of I/O (getpages).

Table 14. Suggested page sizes based on average LOB length

Average LOB size (n)	Suggested page size
n ≤ 4 KB	4 KB
4 KB < n ≤ 8 KB	8 KB
8 KB < n ≤ 16 KB	16 KB
16 KB < n	32 KB

The estimates in Table 14 mean that a LOB value of 17 KB can mean 15 KB of unused space. Again, you must analyze your data to determine what is best.

General guidelines for LOBs of same size: If your LOBs are all the same size, you can fairly easily choose a page size that uses space efficiently without sacrificing performance. For LOBs that are all the same size, consider the alternative in Table 15 to maximize your space savings.

Table 15. Suggested page sizes when LOBs are the same size

LOB size (y)	Suggested page size
y ≤ 4 KB	4 KB
4 KB < y ≤ 8 KB	8 KB
8 KB < y ≤ 12 KB	4 KB
12 KB < y ≤ 16 KB	16 KB

Table 15. Suggested page sizes when LOBs are the same size (continued)

LOB size (y)	Suggested page size
16 KB < y ≤ 24 KB	8 KB
24 KB < y ≤ 32 KB	32 KB
32 KB < y ≤ 48 KB	16 KB
48 KB < y	32 KB

Implementing tables

This section discusses the following topics:

- "Distinctions between DB2 base tables and temporary tables"
- "Implementing table-controlled partitioning" on page 51

Distinctions between DB2 base tables and temporary tables

Table 16 on page 49 summarizes important distinctions between base tables and the two types of temporary tables. Additional examples of implementing temporary tables and information about restrictions and extensions of temporary tables can be found in Part 1 of *DB2 Application Programming and SQL Guide* and in Chapter 5 of *DB2 SQL Reference*. For information about temporary tables and their impact on DB2 resources, see "Work file data sets" on page 693.

Table 16. Important distinctions between DB2 base tables and DB2 temporary tables

Area of distinction	Base tables	Created temporary tables	Declared temporary tables
Creation, persistence, and ability to share table descriptions	CREATE TABLE statement puts a description of the table in catalog table SYSTABLES. The table description is persistent and is shareable across application processes. The name of the table in the CREATE statement can be a two-part or three-part name. If the table name is not qualified, DB2 implicitly qualifies the name using the standard DB2 qualification rules applied to the SQL statements.	CREATE GLOBAL TEMPORARY TABLE statement puts a description of the table in catalog table SYSTABLES. The table description is persistent and is shareable across application processes. The name of the table in the CREATE statement can be a two-part or three-part name. If the table name is not qualified, DB2 implicitly qualifies the name using the standard DB2 qualification rules applied to the SQL statements. The table space used by created temporary tables is reset by the following commands: START DB2, START DATABASE, and START DATABASE(*dbname*) SPACNAM(*tsname*), where *dbname* is the name of the database and *tsname* is the name of the table space.	DECLARE GLOBAL TEMPORARY TABLE statement does *not* put a description of the table in catalog table SYSTABLES. The table description is not persistent beyond the life of the application process that issued the DECLARE statement and the description is known only to that application process. Thus, each application process could have its own possibly unique description of the same table. The name of the table in the DECLARE statement can be a two-part or three-part name. If the table name is qualified, SESSION must be used as the qualifier for the owner (the second part in a three-part name). If the table name is not qualified, DB2 implicitly uses SESSION as the qualifier. The table space used by declared temporary tables is reset by the following commands: START DB2, START DATABASE, and START DATABASE(*dbname*) SPACNAM(*tsname*), where *dbname* is the name of the database and *tsname* is the name of the table space.
Table instantiation and ability to share data	CREATE TABLE statement creates one empty instance of the table, and all application processes use that one instance of the table. The table and data are persistent.	CREATE GLOBAL TEMPORARY TABLE statement does not create an instance of the table. The first implicit or explicit reference to the table in an OPEN, SELECT, INSERT, or DELETE operation executed by any program in the application process creates an empty instance of the given table. Each application process has its own unique instance of the table, and the instance is not persistent beyond the life of the application process.	DECLARE GLOBAL TEMPORARY TABLE statement creates an empty instance of the table for the application process. Each application process has its own unique instance of the table, and the instance is not persistent beyond the life of the application process.

Table 16. Important distinctions between DB2 base tables and DB2 temporary tables (continued)

Area of distinction	Base tables	Created temporary tables	Declared temporary tables
References to the table in application processes	References to the table name in multiple application processes refer to the same single persistent table description and same instance at the current server. If the table name being referenced is not qualified, DB2 implicitly qualifies the name using the standard DB2 qualification rules applied to the SQL statements. The name can be a two-part or three-part name.	References to the table name in multiple application processes refer to the same single persistent table description but to a distinct instance of the table for each application process at the current server. If the table name being referenced is not qualified, DB2 implicitly qualifies the name using the standard DB2 qualification rules applied to the SQL statements. The name can be a two-part or three-part name.	References to that table name in multiple application processes refer to a distinct description and instance of the table for each application process at the current server. References to the table name in an SQL statement (other than the DECLARE GLOBAL TEMPORARY TABLE statement) must include SESSION as the qualifier (the first part in a two-part table name or the second part in a three-part name). If the table name is not qualified with SESSION, DB2 assumes the reference is to a base table.
Table privileges and authorization	The owner implicitly has all table privileges on the table and the authority to drop the table. The owner's table privileges can be granted and revoked, either individually or with the ALL clause. Another authorization ID can access the table only if it has been granted appropriate privileges for the table.	The owner implicitly has all table privileges on the table and the authority to drop the table. The owner's table privileges can be granted and revoked, but only with the ALL clause; individual table privileges cannot be granted or revoked. Another authorization ID can access the table only if it has been granted ALL privileges for the table.	PUBLIC implicitly has all table privileges on the table without GRANT authority and has the authority to drop the table. These table privileges cannot be granted or revoked. Any authorization ID can access the table without a grant of any privileges for the table.
Indexes and other SQL statement support	Indexes and SQL statements that modify data (INSERT, UPDATE, DELETE, and so on) are supported.	Indexes, UPDATE (searched or positioned), and DELETE (positioned only) are not supported.	Indexes and SQL statements that modify data (INSERT, UPDATE, DELETE, and so on) are supported
Locking, logging, and recovery	Locking, logging, and recovery do apply.	Locking, logging, and recovery do not apply. Work files are used as the space for the table.	Some locking, logging, and limited recovery do apply. No row or table locks are acquired. Share-level locks on the table space and DBD are acquired. A segmented table lock is acquired when all the rows are deleted from the table or the table is dropped. Undo recovery (rolling back changes to a savepoint or the most recent commit point) is supported, but redo recovery (forward log recovery) is not supported.

Table 16. Important distinctions between DB2 base tables and DB2 temporary tables (continued)

Area of distinction	Base tables	Created temporary tables	Declared temporary tables
Table space and database operations	Table space and database operations do apply.	Table space and database operations do not apply.	Table space and database operations do apply.
Table space requirements and table size limitations	The table can be stored in simple table spaces in default database DSNDB04 or user-defined table spaces (simple, segmented, or partitioned) in user-defined databases. The table cannot span table spaces. Therefore, the size of the table is limited by the table space size (as determined by the primary and secondary space allocation values specified for the table space's data sets) and the shared usage of the table space among multiple users. When the table space is full, an error occurs for the SQL operation.	The table is stored in table spaces in the work file database. The table can span work file table spaces. Therefore, the size of the table is limited by the number of available work file table spaces, the size of each table space, and the number of data set extents that are allowed for the table spaces. Unlike the other types of tables, created temporary tables do not reach size limitations as easily.	The table is stored in segmented table spaces in the TEMP database (a database that is defined AS TEMP). The table cannot span table spaces. Therefore, the size of the table is limited by the table space size (as determined by the primary and secondary space allocation values specified for the table space's data sets) and the shared usage of the table space among multiple users. When the table space is full, an error occurs for the SQL operation.

Implementing table-controlled partitioning

Before DB2 Version 8, when you defined a partitioning index on a table in a partitioned table space, you specified the partitioning key and the limit key values in the PART VALUES clause of the CREATE INDEX statement. This type of partitioning is called *index-controlled partitioning*.

In DB2 Version 8, the partitioning index is no longer required. You can now specify the partitioning key and the limit key values for a table in a partitioned table space by using the PARTITION BY clause and the PARTITION ENDING AT clause of the CREATE TABLE statement. This type of partitioning is called *table-controlled partitioning*.

Example: Assume that you need to create a large transaction table that includes the date of the transaction in a column named POSTED. You want the transactions for each month in a separate partition. To create the table, issue the following statement:

```
CREATE TABLE TRANS
   (ACCTID ...,
    STATE ...,
    POSTED ...,
    ... , ...)
   PARTITION BY (POSTED)
   (PARTITION 1  ENDING AT ('01/31/2003'),
    PARTITION 2  ENDING AT ('02/28/2003'),
    ...
    PARTITION 13 ENDING AT ('01/31/2004'));
```

Restriction: If you use table-controlled partitioning, you cannot specify the partitioning key and the limit key values by using the PART VALUES clause of the

CREATE INDEX statement. (Note that in Version 8, the preferred syntax changed from PART VALUES to PARTITION ENDING AT.)

Recommendation: Use table-controlled partitioning instead of index-controlled partitioning. Table-controlled partitioning is a replacement for index-controlled partitioning. Table 17 lists the differences between the two partitioning methods.

Table 17. Differences between table-controlled and index-controlled partitioning

Table-controlled partitioning	Index-controlled partitioning
A partitioning index is not required; clustering index is not required.	A partitioning index is required; clustering index is required.
Multiple partitioned indexes can be created in a table space.	Only one partitioned index can be created in a table space.
A table space partition is identified by both a physical partition number and a logical partition number.	A table space partition is identified by a physical partition number.
The high-limit key is always enforced.	The high-limit key is not enforced if the table space is non-large.

See "Moving from index-controlled to table-controlled partitioning" on page 97 for detailed information about converting to table-controlled partitioning.

Automatic conversion

DB2 automatically converts an index-controlled partitioned table space to a table-controlled partitioned table space if you perform any of the following operations:

- Use CREATE INDEX with the PARTITIONED clause to create a partitioned index on an index-controlled partitioned table space.
- Use CREATE INDEX with a PART VALUES clause and without a CLUSTER clause to create a partitioning index.

  DB2 stores the specified high limit key value instead of the default high limit key value.
- Use ALTER INDEX with the NOT CLUSTER clause on a partitioning index that is on an index-controlled partitioned table space.
- Use DROP INDEX to drop a partitioning index on an index-controlled partitioned table space.
- Use ALTER TABLE to add a new partition, change a partition boundary, or rotate a partition from first to last on an index-controlled partitioned table space.

  In these cases, DB2 automatically converts to table-controlled partitioning but does not automatically drop any indexes. DB2 assumes that any existing indexes are useful.

After the conversion to table-controlled partitioning, DB2 changes the existing high-limit key value for non-large table spaces to the highest value for the key. Beginning in Version 8, DB2 enforces the high-limit key value. By default, DB2 does not put the last partition of the table space into a REORG-pending (REORP) state. Exceptions to this rule are:

- When adding a new partition, DB2 stores the original high-limit key value instead of the default high-limit key value. If this value was not previously enforced, DB2 puts the last partition into a REORP state.

- When rotating a new partition, DB2 stores the original high-limit key value instead of the default high-limit key value. DB2 puts the last partition into a REORP state.

After the conversion to table-controlled partitioning, the SQL statements that you used to create the tables and indexes are no longer valid. For example, after dropping a partitioning index on an index-controlled partitioned table space, an attempt to recreate the index by issuing the same CREATE INDEX statement that you originally used would fail because the boundary partitions are now under the control of the table.

Using nullable partitioning columns

DB2 lets you use nullable columns as partitioning columns. The use of nullable columns has different implications for table-controlled partitioning than for index-controlled partitioning.

With table-controlled partitioning, DB2 can restrict the insertion of null values into a table with nullable partitioning columns, depending on the order of the partitioning key:

- If the partitioning key is ascending and the highest value of the key column is not MAXVALUE, DB2 prevents the INSERT of a row with a null value for the key column If the highest value of the key column is MAXVALUE, the row is inserted into the last partition.
- If the partitioning key is descending, DB2 allows the INSERT of a row with a null value for the key column. The row is inserted into the first partition.

Example: Assume that a partitioned table space is created with the following SQL statements:

```
CREATE TABLESPACE TS IN DB
 USING STOGROUP SG
 NUMPARTS 4 BUFFERPOOL BP0;

CREATE TABLE TB (C01 CHAR(5),
      C02 CHAR(5) NOT NULL,
      C03 CHAR(5) NOT NULL)
 IN DB.TS
  PARTITION BY (C01)
   PARTITION 1 ENDING AT ('10000'),
   PARTITION 2 ENDING AT ('20000'),
   PARTITION 3 ENDING AT ('30000'),
   PARTITION 4 ENDING AT ('40000'));
```

Because the CREATE TABLE statement does not specify the order in which to put entries, DB2 puts them in ascending order by default. DB2 subsequently prevents any INSERT into the TB table of a row with a null value for partitioning column C01. If the CREATE TABLE statement had specified the key as descending, DB2 would subsequently have allowed an INSERT into the TB table of a row with a null value for partitioning column C01. DB2 would have inserted the row into partition 1.

With index-controlled partitioning, DB2 does not restrict the insertion of null values into a value with nullable partitioning columns.

Example: Assume that a partitioned table space is created with the following SQL statements:

```
CREATE TABLESPACE TS IN DB
 USING STOGROUP SG
 NUMPARTS 4 BUFFERPOOL BP0;
```

```
CREATE TABLE TB (C01 CHAR(5),
      C02 CHAR(5) NOT NULL,
      C03 CHAR(5) NOT NULL)
 IN DB.TS;

CREATE INDEX PI ON TB(C01) CLUSTER
 (PARTITION 1 ENDING AT ('10000'),
 PARTITION 2 ENDING AT ('20000'),
 PARTITION 3 ENDING AT ('30000'),
 PARTITION 4 ENDING AT ('40000'));
```

Regardless of the entry order, DB2 allows an INSERT into the TB table of a row with a null value for partitioning column C01. If the entry order is ascending, DB2 inserts the row into partition 4; if the entry order is descending, DB2 inserts the row into partition 1. Only if the table space is created with the LARGE keyword does DB2 prevent the insertion of a null value into the C01 column.

Implementing indexes

DB2 uses indexes not only to enforce uniqueness on column values, as for parent keys, but also to cluster data, to partition tables, to provide access paths to data for queries, and to order retrieved data without a sort.

This section discusses the following topics:
- "Types of indexes"
- "Using the NOT PADDED clause for indexes with varying-length columns" on page 57
- "Using indexes to avoid sorts" on page 58

Types of indexes

This section describes the various types of indexes and summarizes what you should consider when creating a particular type:
- "Unique indexes"
- "Clustering indexes" on page 55
- "Partitioning indexes" on page 55
- "Secondary indexes" on page 56

This section uses a transaction table named TRANS to illustrate the various types of indexes. Assume that the table has many columns, but you are interested in only the following columns:
- ACCTID, which is the customer account ID
- STATE, which is the state of the customer location
- POSTED, which holds the date of the transaction

Unique indexes

When you define a *unique index* on a DB2 table, you ensure that no duplicate values of the index key exist in the table. For example, creating a unique index on the ACCTID column ensures that no duplicate values of the customer account ID are in the TRANS table:

```
CREATE UNIQUE INDEX IX1
  ON TRANS (ACCTID);
```

If an index key allows nulls for some of its column values, you can use the WHERE NOT NULL clause to ensure that the nonnull values of the index key are unique.

Unique indexes are an important part of implementing referential constraints among the tables in your DB2 database. You cannot define a foreign key unless the corresponding primary key already exists and has a unique index defined on it. For more information, see "Altering a table for referential integrity" on page 77 and *The Official Introduction to DB2 UDB for z/OS*.

Clustering indexes

When you define a *clustering index* on a DB2 table, you direct DB2 to insert rows into the table in the order of the clustering key values. The first index that you define on the table serves implicitly as the clustering index unless you explicitly specify CLUSTER when you create or alter another index. For example, if you first define a unique index on the ACCTID column of the TRANS table, DB2 inserts rows into the TRANS table in the order of the customer account number unless you explicitly define another index to be the clustering index.

You can specify CLUSTER for any index, whether or not it is a partitioning index. For example, suppose that you want the rows of the TRANS table to be ordered by the POSTED column. Issue the statement:

```
CREATE INDEX IX2
  ON TRANS (POSTED)
  CLUSTER;
```

For more information, see "Altering the clustering index" on page 93 and *The Official Introduction to DB2 UDB for z/OS*.

Partitioning indexes

Before DB2 Version 8, when you defined a partitioning index on a table in a partitioned table space, you specified the partitioning key and the limit key values in the PART VALUES clause of the CREATE INDEX statement. This type of partitioning is referred to as index-controlled partitioning.

Beginning with DB2 Version 8, you can define table-controlled partitioning with the CREATE TABLE statement (which is described in "Implementing table-controlled partitioning" on page 51). A *partitioning index* is then defined to be an index where the leftmost columns are the partitioning columns of the table; the index can, but need not, be partitioned. If it is partitioned, a partitioning index is partitioned in the same way as the underlying data in the table.

For example, assume that the partitioning scheme for the TRANS table is defined by the CREATE TABLE statement in "Implementing table-controlled partitioning" on page 51. The rows in the table are partitioned by the transaction date in the POSTED column. The following statement defines an index where the column of the index is the same as the partitioning column of the table:

```
CREATE INDEX IX3
  ON TRANS (POSTED);
```

When you create a partitioning index, DB2 puts the last partition of the table space into a REORG-pending (REORP) state.

A partitioning index is optional; table-controlled partitioning is a replacement for index-controlled partitioning. For more information, see "Moving from index-controlled to table-controlled partitioning" on page 97.

Secondary indexes

A *secondary index* is any index that is not a partitioning index. You can create an index on a table to enforce a uniqueness constraint, to cluster data, or most typically to provide access paths to data for queries.

The usefulness of an index depends on the columns in its key and on the cardinality of the key. Columns that you use frequently in performing selection, join, grouping, and ordering operations are good candidates for keys. In addition, the number of distinct values in an index key for a large table must be sufficient for DB2 to use the index for data retrieval; otherwise, DB2 could choose to perform a table space scan.

A secondary index can be partitioned or not. This section discusses the two types of secondary indexes:

- "Nonpartitioned secondary index (NPSI)"
- "Data-partitioned secondary index (DPSI)"

Nonpartitioned secondary index (NPSI): A *nonpartitioned secondary index* is any index that is not defined as a partitioning index or a partitioned index. You can create a nonpartitioned secondary index on a table that resides in a partitioned table space or a nonpartitioned table space. For example, assume that the transaction date in the POSTED column is the partitioning key of the TRANS table and that the rows are ordered by the transaction date. To create an index on the STATE column, issue the following statement:

```
CREATE INDEX IX4 ON TRANS(STATE);
```

DB2 can use this index to access data with a particular value for STATE. However, if the query includes a predicate that references only a single partition of the table, the keys for that partition are scattered throughout the index. A better solution to accessing single partitions is data-partitioned secondary indexes (DPSIs).

Data-partitioned secondary index (DPSI): A *data-partitioned secondary index* is any index that is not defined as a partitioning index but is defined as a partitioned index. You can create a partitioned secondary index only on a table that resides in a partitioned table space. The partitioning scheme is the same as that of the data in the underlying table. That is, the index entries that reference data in physical partition 1 of a table reside in physical partition 1 of the index, and so on.

DB2 puts a data-partitioned secondary index into a REBUILD-pending (RBDP) state if you create the index after performing any of the following actions:

- Create a partitioned table space
- Create a partitioning index
- Insert a row into a table

In this situation, the last partition of the table space is set to REORG-pending (REORP) restrictive status.

Advantages and disadvantages of DPSIs: The use of data-partitioned secondary indexes promotes partition independence and therefore provides the following performance advantages, among others:

- Eliminates contention between parallel LOAD PART jobs that target different partitions of a table space
- Facilitates partition-level operations such as adding a new partition or rotating the first partition to be the last partition
- Improves the recovery time of secondary indexes on partitioned table spaces

However, the use of data-partitioned secondary indexes does not always improve the performance of queries. For example, for queries with predicates that reference only the columns in the key of the DPSI, DB2 must probe each partition of the index for values that satisfy the predicate.

Example: Assume that the transaction date in the POSTED column is the partitioning key of the table and that the rows are ordered by the transaction date. You want an index on the STATE column that is partitioned the same as the data in the table. Issue the following statement:

```
CREATE INDEX IX5 ON TRANS(STATE)
  PARTITIONED;
```

DB2 can use this index to access data with a particular value for STATE within partitions that are specified in a predicate of a query.

Example: Assume that the transaction date in the POSTED column is the partitioning key of the table. You want a clustering index on the ACCTID column that is partitioned the same as the data in the table. Issue the following statement:

```
CREATE INDEX IX6 ON TRANS(ACCTID)
  PARTITIONED CLUSTER;
```

DB2 orders the rows of the table by the values of the columns in the clustering key and partitions the rows by the values of the limit key that is defined for the underlying table. The data rows are clustered within each partition by the key of the clustering index instead of by the partitioning key.

Using the NOT PADDED clause for indexes with varying-length columns

If you specify the NOT PADDED clause on a CREATE INDEX statement, any varying-length columns in the index key are not padded to their maximum length. If an existing index key includes varying-length columns, you can consider altering the index to use the NOT PADDED clause (see "Altering how varying-length index columns are stored" on page 93).

Using the NOT PADDED clause has several advantages:

- DB2 can use index-only access for the varying-length columns within the index key, which enhances performance.
- DB2 stores only actual data, which reduces the storage requirements for the index key.

However, using the NOT PADDED clause might also have several disadvantages:

- Index key comparisons are slower because DB2 must compare each pair of corresponding varying-length columns individually instead of comparing the entire key when the columns are padded to their maximum length.
- DB2 stores an additional 2-byte length field for each varying-length column. Therefore, if the length of the padding (to the maximum length) is less than or equal to 2 bytes, the storage requirements could actually be greater for varying-length columns that are not padded.

Recommendation: Use the NOT PADDED clause to implement index-only access if your application typically accesses varying-length columns.

Tip: Use the PAD INDEXES BY DEFAULT option on installation panel DSNTIPE to control whether varying length columns are padded by default.

Using indexes to avoid sorts

DB2 can use indexes to avoid sorts when processing queries with the ORDER BY clause. When a query contains an ORDER BY clause, DB2 looks for indexes that satisfy the order in the query. For DB2 to be able to use an index to access ordered data, you must define an index on the same columns as specified in the ORDER BY clause. For DB2 to use a forward index scan, the ordering must be exactly the same as in the ORDER BY clause. For DB2 to use a backward index scan, the ordering must be exactly the opposite of what is requested in the ORDER BY clause.

Example: For example, if you define an index by specifying DATE DESC, TIME ASC as the column names and order, DB2 can use this same index for both of the following ORDER BY clauses:

- Forward scan for ORDER BY DATE DESC, TIME ASC
- Backward scan for ORDER BY DATE ASC, TIME DESC

You do not need to create two indexes for the two ORDER BY clauses. DB2 can use the same index for both forward index scan and backward index scan.

Optimization: Suppose that the query includes a WHERE clause with a predicate of the form COL=*constant*. For example:

```
...
WHERE CODE = 'A'
ORDER BY CODE, DATE DESC, TIME ASC
```

DB2 can use any of the following index keys to satisfy the ordering:

- CODE, DATE DESC, TIME ASC
- CODE, DATE ASC, TIME DESC
- DATE DESC, TIME ASC
- DATE ASC, TIME DESC

DB2 can ignore the CODE column in the ORDER BY clause and the index because the value of the CODE column in the result table of the query has no effect on the order of the data. If the CODE column is included, it can be in any position in the ORDER BY clause and in the index.

Using schemas

A *schema* is a collection of named objects. The objects that a schema can contain include tables, indexes, table spaces, distinct types, functions, stored procedures, and triggers. An object is assigned to a schema when it is created.

When a table, index, table space, distinct type, function, stored procedure, or trigger is created, it is given a qualified two-part name. The first part is the schema name (or the qualifier), which is either implicitly or explicitly specified. The default schema is the authorization ID of the owner of the plan or package. The second part is the name of the object.

Creating a schema

You can create a schema with the schema processor by using the CREATE SCHEMA statement. CREATE SCHEMA cannot be embedded in a host program or executed interactively. To process the CREATE SCHEMA statement, you must use the schema processor, as described in “Processing schema definitions” on page 59. The ability to process schema definitions is provided for conformance to ISO/ANSI standards. The result of processing a schema definition is identical to the result of executing the SQL statements without a schema definition.

Outside of the schema processor, the order of statements is important. They must be arranged so that all referenced objects have been previously created. This restriction is relaxed when the statements are processed by the schema processor if the object table is created within the same CREATE SCHEMA. The requirement that all referenced objects have been previously created is not checked until all of the statements have been processed. For example, within the context of the schema processor, you can define a constraint that references a table that does not exist yet or GRANT an authorization on a table that does not exist yet.

Figure 5 is an example of schema processor input that includes the definition of a schema.

```
CREATE SCHEMA AUTHORIZATION SMITH

CREATE TABLE TESTSTUFF
 (TESTNO   CHAR(4),
  RESULT   CHAR(4),
  TESTTYPE  CHAR(3))

CREATE TABLE STAFF
 (EMPNUM   CHAR(3) NOT NULL,
  EMPNAME  CHAR(20),
  GRADE    DECIMAL(4),
  CITY     CHAR(15))

CREATE VIEW STAFFV1
         AS SELECT * FROM STAFF
            WHERE  GRADE >= 12

GRANT INSERT ON TESTSTUFF TO PUBLIC

GRANT ALL PRIVILEGES ON STAFF
        TO PUBLIC
```

Figure 5. Example of schema processor input

Authorization to process schema definitions

The schema processor sets the current SQLID to the value of the schema authorization ID before executing any of the statements in the schema definition. Therefore, that ID must have SYSADM or SYSCTRL authority, or it must be the primary or one of the secondary authorization IDs of the process that executes the schema processor. The same ID must have all the privileges that are needed to execute all the statements in the schema definition.

Processing schema definitions

Run the schema processor (DSNHSP) as a batch job; use the sample JCL provided in member DSNTEJ1S of the SDSNSAMP library. The schema processor accepts only one schema definition in a single job. No statements that are outside the schema definition are accepted. Only SQL comments can precede the CREATE SCHEMA statement; the end of input ends the schema definition. SQL comments can also be used within and between SQL statements.

The processor takes the SQL from CREATE SCHEMA (the SYSIN data set), dynamically executes it, and prints the results in the SYSPRINT data set.

If a statement in the schema definition has an error, the schema processor processes the remaining statements but rolls back all the work at the end. In this case, you need to fix the statement in error and resubmit the entire schema definition.

Chapter 5. Loading data into DB2 tables

This chapter provides an overview of how to load data into DB2 tables:

- "Loading tables with the LOAD utility"
- "Making corrections after LOAD" on page 63
- "Loading data using the SQL INSERT statement" on page 63
- "Loading data from DL/I" on page 65

You can use several methods to fill DB2 tables, but you will probably load most of your tables by using the LOAD utility. This utility loads data into DB2 persistent tables from sequential data sets by using BSAM. You can also use a cursor that is declared with an EXEC SQL utility control statement to load data from another SQL table with the DB2 UDB family cross-loader function. The LOAD utility cannot be used to load data into DB2 temporary tables or system-maintained materialized query tables. For information about the DB2 UDB family cross-loader function, see Chapter 15, Load of the *DB2 Utility Guide and Reference*.

When loading tables with indexes, referential constraints, or table check constraints, LOAD can perform several checks on the validity of data. If errors are found, the table space that is being loaded, its index spaces, and even other table spaces might be left in a restricted status. LOAD does not check the validity of informational referential constraints. Plan to make necessary corrections and remove restrictions after any LOAD job. For instructions, see "Making corrections after LOAD" on page 63.

You can also use an SQL INSERT statement to copy all or selected rows of another table, either by using the INSERT statement in an application program or interactively through SPUFI.

To reformat data from IMS DL/I databases and VSAM and SAM loading for the LOAD utility, use DB2 DataPropagator. See "Loading data from DL/I" on page 65.

For general guidance about running DB2 utility jobs, see *DB2 Utility Guide and Reference*. For information about DB2 DataPropagator, see *DB2 Universal Database Replication Guide and Reference*.

Loading tables with the LOAD utility

Use LOAD to load one or more persistent tables of a table space, or one or more partitions of a table space. LOAD operates on a table space, so you must have authority for all tables in the table space when you run LOAD.

The LOAD utility loads records into the tables and builds or extends any indexes defined on them. If the table space already contains data, you can choose whether you want to add the new data to the existing data or replace the existing data.

Additionally, LOAD can be used to:
- Compress data and build a compression dictionary
- Convert data between compatible data types and between encoding schemes
- Load multiple tables in a single table space

© Copyright IBM Corp. 1982, 2008

Using delimited input and output files: The LOAD and UNLOAD utilities can accept or produce a delimited file, which is a sequential BSAM file with row delimiters and column delimiters. You can unload data from other systems into one or more files that use a delimited file format and then use these delimited files as input for the LOAD utility. You can also unload DB2 data into delimited files by using the UNLOAD utility and then use these files as input into another DB2 database.

Using the INCURSOR option: The INCURSOR option of the LOAD utility specifies a cursor for the input data set. Use the EXEC SQL utility control statement to declare the cursor before running the LOAD utility. You define the cursor so that it selects data from another DB2 table. The column names in the SELECT statement must be identical to the column names of the table that is being loaded. The INCURSOR option uses the DB2 UDB family cross-loader function.

Using the CCSID option: You can load input data into ASCII, EBCDIC, or Unicode tables. The ASCII, EBCDIC, and UNICODE options on the LOAD utility statement let you specify whether the format of the data in the input file is ASCII, EBCDIC, or Unicode. The CCSID option of the LOAD utility statement lets you specify the CCSIDs of the data in the input file. If the CCSID of the input data does not match the CCSID of the table space, the input fields are converted to the CCSID of the table space before they are loaded.

Availability during load: For nonpartitioned table spaces, data in the table space that is being loaded is unavailable to other application programs during the load operation with the exception of LOAD SHRLEVEL CHANGE. In addition, some SQL statements, such as CREATE, DROP, and ALTER, might experience contention when they run against another table space in the same DB2 database while the table is being loaded.

Default values for columns: When you load a table and do not supply a value for one or more of the columns, the action DB2 takes depends on the circumstances.

- If the column is not a ROWID or identity column, DB2 loads the default value of the column, which is specified by the DEFAULT clause of the CREATE or ALTER TABLE statement.
- If the column is a ROWID column that uses the GENERATED BY DEFAULT option, DB2 generates a unique value.
- If the column is an identity column that uses the GENERATED BY DEFAULT option, DB2 provides a specified value.

For ROWID or identity columns that use the GENERATED ALWAYS option, you cannot supply a value because this option means that DB2 always provides a value.

LOB columns: The LOAD utility treats LOB columns as varying-length data. The length value for a LOB column must be 4 bytes. The LOAD utility can be used to load LOB data if the length of the row, including the length of the LOB data, does not exceed 32 KB. The auxiliary tables are loaded when the base table is loaded. You cannot specify the name of the auxiliary table to load.

Replacing or adding data: You can use LOAD REPLACE to replace data in a single-table table space or in a multiple-table table space. You can replace all the data in a tale space (using the REPLACE option), or you can load new records into a table space without destroying the rows that are already there (using the RESUME option).

Making corrections after LOAD

LOAD can place a table space or index space into one of several kinds of restricted status. Your use of a table space in restricted status is severely limited. In general, you cannot access its data through SQL; you can only drop the table space or one of its tables, or perform some operation that resets the status.

To discover what spaces are in restricted status, use the command:

```
-DISPLAY DATABASE (*) SPACENAM (*) RESTRICT
```

COPY-pending status: LOAD places a table space in the COPY-pending state if you load with LOG NO, which you might do to save space in the log. Immediately after that operation, DB2 cannot recover the table space. However, you can recover the table space by loading it again. Prepare for recovery, and remove the restriction, by making a full image copy using SHRLEVEL REFERENCE. (If you end the COPY job before it is finished, the table space is still in COPY-pending status.)

When you use REORG or LOAD REPLACE with the COPYDDN keyword, a full image copy data set (SHRLEVEL REF) is created during the execution of the REORG or LOAD utility. This full image copy is known as an *inline copy*. The table space is not left in COPY-pending state regardless of which LOG option is specified for the utility.

The inline copy is valid only if you replace the entire table space or partition. If you request an inline copy by specifying COPYDDN in a LOAD utility statement, an error message is issued, and the LOAD terminates if you specify LOAD RESUME YES or LOAD RESUME NO without REPLACE.

REBUILD-pending status: LOAD places all the index spaces for a table space in the REBUILD-pending status if you end the job (using -TERM UTILITY) before it completes the INDEXVAL phase. It places the table space itself in RECOVER-pending status if you end the job before it completes the RELOAD phase.

CHECK-pending status: LOAD places a table space in the CHECK-pending status if its referential or check integrity is in doubt. Because of this restriction, use of the CHECK DATA utility is recommended. That utility locates and, optionally, removes invalid data. If the CHECK DATA utility removes invalid data, the remaining data satisfies all referential and table check constraints, and the CHECK-pending restriction is lifted. LOAD does not set the CHECK-pending status for informational referential constraints.

Loading data using the SQL INSERT statement

The information in this section, up to “Loading data from DL/I” on page 65, is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page 1301:

- “Inserting a single row” on page 64
- “Inserting multiple rows” on page 64
- “Special considerations when using INSERT statement to load tables” on page 64

Another way to load data into tables is with the SQL INSERT statement. You can issue the statement interactively or embed it in an application program.

Inserting a single row

The simplest form of INSERT inserts a single row of data. In this form of the statement, you specify the table name, the columns into which the data is to be inserted, and the data itself.

Example: Suppose that you create a test table, TEMPDEPT, with the same characteristics as the department table:

```
CREATE TABLE SMITH.TEMPDEPT
  (DEPTNO  CHAR(3)  NOT NULL,
  DEPTNAME VARCHAR(36)   NOT NULL,
  MGRNO    CHAR(6)  NOT NULL,
  ADMRDEPT CHAR(3)  NOT NULL)
  IN DSN8D81A.DSN8S81D;
```

To add a row to table TEMPDEPT, you can enter:

```
INSERT INTO SMITH.TEMPDEPT
  VALUES ('X05','EDUCATION','000631','A01');
```

If you write an application program to load data into tables, you use that form of INSERT, probably with host variables instead of the actual values shown in this example.

Inserting multiple rows

You can use a form of INSERT that copies rows from another table. For example, you can load TEMPDEPT with the following statement:

```
INSERT INTO SMITH.TEMPDEPT
  SELECT DEPTNO,DEPTNAME,MGRNO,ADMRDEPT
  FROM DSN8810.DEPT
  WHERE ADMRDEPT='D01';
```

The statement loads TEMPDEPT with data from the department table about all departments that report to department D01.

If you embed the INSERT statement in an application program, you can use a form of the statement that inserts multiple rows into a table from the values that are provided in host variable arrays. In this form, you specify the table name, the columns into which the data is to be inserted, and the arrays that contain the data. Each array corresponds to a column. For example, you can load TEMPDEPT with the number of rows in the host variable *num-rows* by using the following embedded INSERT statement:

```
EXEC SQL
  INSERT INTO SMITH.TEMPDEPT
  FOR :num-rows ROWS
  VALUES (:hva1, :hva2, :hva3, :hva4);
```

Assume that the host variable arrays *hva1*, *hva2*, *hva3*, and *hva4* are populated with the values that are to be inserted. The number of rows to insert must be less than or equal to the dimension of each host variable array.

Special considerations when using INSERT statement to load tables

If you plan to use the INSERT statement to load tables, consider the following topics of special importance:

- If you are inserting a large number of rows, you can use the LOAD utility. Alternatively, use multiple INSERT statements with predicates that isolate the data that is to be loaded, and then commit after each insert operation.
- When a table, whose indexes are already defined, is populated by using the INSERT statement, both the FREEPAGE and the PCTFREE parameters are ignored. FREEPAGE and PCTFREE are in effect only during a LOAD or REORG operation.
- You can load a value for a ROWID column with an INSERT and fullselect only if the ROWID column is defined as GENERATED BY DEFAULT. If you have a table with a column that is defined as ROWID GENERATED ALWAYS, you can propagate non-ROWID columns from a table with the same definition.
- You cannot use an INSERT statement on system-maintained materialized query tables. For information about materialized query tables, see "Registering or changing materialized query tables" on page 85.
- REBUILD-pending (RBDP) status is set on a data-partitioned secondary index if you create the index after you insert a row into a table. In addition, the last partition of the table space is set to REORG-pending (REORP) restrictive status.
- When you insert a row into a table that resides in a partitioned table space and the value of the first column of the limit key is null, the result of the INSERT depends on whether DB2 enforces the limit key of the last partition:
  - When DB2 enforces the limit key of the last partition, the INSERT fails if the first column is ascending and MAXVALUE was not specified as the highest value of the key in the last partition. If the first column is ascending and MAXVALUE was specified as the highest value of the key in the last partition, DB2 places NULLs into the last partition.
  - When DB2 enforces the limit key of the last partition, the rows are inserted into the first partition (if the first column is descending).
  - When DB2 does **not** enforce the limit key of the last partition, the rows are inserted into the last partition (if the first column is ascending) or the first partition (if the first column is descending).

  DB2 enforces the limit key of the last partition for the following table spaces:
  - Table spaces using table-controlled or index-controlled partitioning that are large (DSSIZE greater than, or equal to, 4 GB)
  - Tables spaces using table-controlled partitioning that are large or non-large (any DSSIZE)

For the complete syntax of the INSERT statement, see Chapter 5 of *DB2 SQL Reference*.

Loading data from DL/I

To convert data in IMS DL/I databases from a hierarchical structure to a relational structure so that it can be loaded into DB2 tables, you can use the DataRefresher™ licensed program.

Chapter 6. Altering your database design

The information in this chapter is General-use Programming Interface and Associated Guidance Information, as defined in "Notices" on page 1301.

In addition, this chapter includes techniques for making the following changes:
- "Moving from index-controlled to table-controlled partitioning" on page 97
- "Changing the high-level qualifier for DB2 data sets" on page 98
- "Moving DB2 data" on page 105

Recommendation: If possible, use the SQL ALTER statement to change the definitions of DB2 objects. When you cannot make changes with the ALTER statement, you typically must:
1. Use the DROP statement to remove the object.
2. Use the COMMIT statement to commit the changes to the object.
3. Use the CREATE statement to re-create the object.

The DROP statement has a cascading effect; objects that are dependent on the dropped object are also dropped. For example, all authorities for those objects disappear, and plans or packages that reference deleted objects are marked invalid by DB2. For a more detailed description of this method, see "Changing data types by dropping and re-creating the table" on page 88.

Altering DB2 storage groups

You can use the ALTER STOGROUP statement to specify whether you want SMS to manage your DB2 storage groups, or to add or remove volumes from a storage group:
- "Letting SMS manage your DB2 storage groups"
- "Adding or removing volumes from a DB2 storage group"

If you want to migrate to another device type or change the catalog name of the integrated catalog facility, you need to move the data (see "Moving DB2 data" on page 105).

Letting SMS manage your DB2 storage groups

To let SMS manage the storage needed for the objects that the storage group supports, specify ADD VOLUMES ('*') and REMOVE VOLUMES (*current-vols*) in the ALTER STOGROUP statement, where *current-vols* is the list of the volumes that are currently assigned to the storage group:

```
ALTER STOGROUP DSN8G810
  REMOVE VOLUMES (VOL1)
  ADD VOLUMES ('*');
```

SMS manages every new data set that is created after the ALTER STOGROUP statement is executed; SMS does not manage data sets that are created before the execution of the statement. See "Migrating to DFSMShsm" on page 35 for more considerations for using SMS to manage data sets.

Adding or removing volumes from a DB2 storage group

When you add or remove volumes from a storage group, all the volumes in a storage group must be of the same type; and, when a storage group is used to

© Copyright IBM Corp. 1982, 2008

extend a data set, the volumes must have the same device type as the volumes that were used when the data set was defined.

The changes you make to the volume list by ALTER STOGROUP have no effect on existing storage. Changes take effect when new objects are defined or when the REORG, RECOVER, or LOAD REPLACE utilities are used on those objects. For example, if you use ALTER STOGROUP to remove volume 22222 from storage group DSN8G810, the DB2 data on that volume remains intact. However, when a new table space is defined using DSN8G810, volume 22222 is not available for space allocation.

To force a volume off and add a new volume, follow these steps:

1. Use the SYSIBM.SYSTABLEPART catalog table to determine which table spaces are associated with the storage group. For example, the following query indicates which table spaces use storage group DSN8G810:

   ```
   SELECT TSNAME, DBNAME
     FROM SYSIBM.SYSTABLEPART
     WHERE STORNAME ='DSN8G810' AND STORTYPE = 'I';
   ```

2. Make an image copy of each table space; for example, COPY TABLESPACE *dbname.tsname* DEVT SYSDA.
3. Ensure that the table space is not being updated in such a way that the data set might need to be extended. For example, you can stop the table space with the DB2 command STOP DATABASE (*dbname*) SPACENAM (*tsname*).
4. Use the ALTER STOGROUP statement to remove the volume that is associated with the old storage group and to add the new volume:

   ```
   ALTER STOGROUP DSN8G810
     REMOVE VOLUMES (VOL1)
     ADD VOLUMES (VOL2);
   ```

   Important: When a new volume is added, or when a storage group is used to extend a data set, the volumes must have the same device type as the volumes that were used when the data set was defined.
5. Start the table space with utility-only processing by using the DB2 command START DATABASE (*dbname*) SPACENAM (*tsname*) ACCESS(UT).
6. Use the RECOVER or REORG utility to move the data in each table space; for example, RECOVER *dbname.tsname*. (You can use only the RECOVER utility to move a LOB table space.)
7. Start the table space with START DATABASE (*dbname*) SPACENAM (*tsname*).

Altering DB2 databases

The ALTER DATABASE statement allows you to change the following clauses that are used to create a database:

STOGROUP
: Lets you change the name of the default storage group to support disk space requirements for table spaces and indexes within the database. The new default DB2 storage group is used only for new table spaces and indexes; existing definitions do not change.

BUFFERPOOL
: Lets you change the name of the default buffer pool for table spaces and indexes within the database. Again, it applies only to new table spaces and indexes; existing definitions do not change.

INDEXBP
Lets you change the name of the default buffer pool for the indexes within the database. The new default buffer pool is used only for new indexes; existing definitions do not change.

Altering table spaces

If possible, use the ALTER TABLESPACE statement to modify the description of a table space. This statement can be embedded in an application program or issued interactively.

You cannot use ALTER TABLESPACE to change some attributes of your table space. For example, you must use other methods to change SEGSIZE or the number of partitions or to convert it to a large table space. The following topics describe these methods:
- "Changing the space allocation for user-managed data sets"
- "Dropping, re-creating, or converting a table space"

Changing the space allocation for user-managed data sets

If the table space is supported by user-managed data sets, use this procedure to change the space allocation:

1. Run the REORG TABLESPACE utility, and specify the UNLOAD PAUSE option.
2. Make the table space unavailable with the STOP DATABASE command and the SPACENAM option after the utility completes the unload and stops.
3. Delete and redefine the data sets.
4. Resubmit the utility job with the RESTART(PHASE) parameter specified on the EXEC statement.

   The job now uses the new data sets when reloading.

Use of the REORG utility to extend data sets causes the newly acquired free space to be distributed throughout the table space rather than to be clustered at the end.

Dropping, re-creating, or converting a table space

To make changes to a table space, such as changing SEGSIZE or the number of partitions or to convert it to a large table space, you must first drop the table space and then re-create it. You must commit the DROP TABLESPACE statement before creating a table space or index with the same name. When you drop a table space, all entries for that table space are dropped from SYSIBM.SYSCOPY. This makes recovery for that table space impossible from previous image copies. You can change or convert your table spaces with the following steps:

1. Locate the original CREATE TABLE statement and all authorization statements for all tables in the table space (for example, TA1, TA2, TA3, ... in TS1). If you cannot find these statements, query the DB2 catalog to determine the table's description, the description of all indexes and views on it, and all users with privileges on the table.
2. In another table space (TS2, for example), create tables TB1, TB2, TB3, ... identical to TA1, TA2, TA3, ... For example, use statements like:

   ```
   CREATE TABLE TB1 LIKE TA1 IN TS2;
   ```
3. If necessary, unload the data using a statement such as:

   ```
   REORG TABLESPACE DSN8D81A.TS1 LOG NO SORTDATA UNLOAD EXTERNAL;
   ```

Another way of unloading data from your old tables and loading into the new tables is by using the INCURSOR option of the LOAD utility. This option uses the DB2 UDB family cross-loader function.

4. Alternatively, instead of unloading the data, you can insert the data from your old tables into the new tables by executing an INSERT statement for each table. For example:

```
INSERT INTO TB1
  SELECT * FROM TA1;
```

   If a table contains a ROWID column or an identity column and you want to keep the existing column values, you must define that column as GENERATED BY DEFAULT. If the ROWID column or identity column is defined with GENERATED ALWAYS, and you want DB2 to generate new values for that column, specify OVERRIDING USER VALUE on the INSERT statement with the subselect.

5. Drop the table space by executing the statement:

```
DROP TABLESPACE TS1;
```

   The compression dictionary for the table space is dropped, if one exists. All tables in TS1 are dropped automatically.

6. Commit the DROP statement.
7. Create the new table space, TS1, and grant the appropriate user privileges. You can also create a partitioned table space. You could use the following statements:

```
CREATE TABLESPACE TS1
   IN DSN8D81A
   USING STOGROUP DSN8G810
     PRIQTY 4000
     SECQTY 130
     ERASE NO
 NUMPARTS 95
  (PARTITION 45 USING STOGROUP DSN8G810
     PRIQTY 4000
     SECQTY 130
     COMPRESS YES,
   PARTITION 62 USING STOGROUP DSN8G810
     PRIQTY 4000
     SECQTY 130
     COMPRESS NO)
 LOCKSIZE PAGE
 BUFFERPOOL BP1
 CLOSE NO;
```

8. Create new tables TA1, TA2, TA3, ...
9. Re-create indexes on the tables, and re-grant user privileges on those tables. See “Implications of dropping a table” on page 89 for more information.
10. Execute an INSERT statement for each table. For example:

```
INSERT INTO TA1
  SELECT * FROM TB1;
```

   If a table contains a ROWID column or an identity column and you want to keep the existing column values, you must define that column as GENERATED BY DEFAULT. If the ROWID column or identity column is defined with GENERATED ALWAYS, and you want DB2 to generate new values for that column, specify OVERRIDING USER VALUE on the INSERT statement with the subselect.

11. Drop table space TS2.

If a table in the table space has been created with RESTRICT ON DROP, you must alter that table to remove the restriction before you can drop the table space.

12. Notify users to re-create any synonyms they had on TA1, TA2, TA3, ...
13. REBIND plans and packages that were invalidated as a result of dropping the table space.

Altering tables

When you alter a table, you do not change the data in the table; you merely change the specifications that you used in creating the table.

With the ALTER TABLE statement, you can:

- Add a new column to a table; see "Adding a new column to a table."
- Change the data type of a column, with certain restrictions; see "Altering the data type of a column" on page 73.
- Add or drop a parent key or a foreign key; see "Altering a table for referential integrity" on page 77.
- Add or drop a table check constraint; see "Adding or dropping table check constraints" on page 80.
- Add a new partition to a table space; see "Adding a partition" on page 80.
- Change the boundary between partitions, extend the boundary of the last partition, or rotate partitions; see "Altering partitions" on page 82.
- Register an existing table as a materialized query table, change the attributes of a materialized query table, or change a materialized query table to a base table; see "Registering or changing materialized query tables" on page 85.
- Change the VALIDPROC clause; see "Altering the assignment of a validation routine" on page 86.
- Change the DATA CAPTURE clause; see "Altering a table for capture of changed data" on page 87.
- Change the AUDIT clause, using the options ALL, CHANGES, or NONE. For the effects of the AUDIT value, see Chapter 12, "Protecting data sets through RACF," on page 281.
- Add or drop the restriction on dropping the table and the database and table space that contain the table; see *DB2 SQL Reference*.
- Alter the length of a VARCHAR column using the SET DATA TYPE VARCHAR clause; see *DB2 SQL Reference*.

In addition, this section includes techniques for making the following changes:

- "Changing an edit procedure or a field procedure" on page 87
- "Altering the subtype of a string column" on page 88
- "Altering the attributes of an identity column" on page 88
- "Changing data types by dropping and re-creating the table" on page 88
- "Moving a table to a table space of a different page size" on page 91

Adding a new column to a table

When you use ALTER TABLE to add a new column to a table, the new column becomes the rightmost column of the table and the table space is placed in an advisory REORG-pending (AREO*) state. The physical records are not actually changed until values are inserted in the new column. Plans and packages are not invalidated unless the new column is a TIME, TIMESTAMP, or DATE. However, to use the new column in a program, you need to modify and recompile the program

and bind the plan or package again. You might also need to modify any program containing a static SQL statement SELECT *, which returns the new column after the plan or package is rebound. You must also modify any INSERT statement not containing a column list.

Access time to the table is not affected immediately, unless the record was previously fixed length. If the record was fixed length, the addition of a new column causes DB2 to treat the record as variable length and then access time is affected immediately. To change the records to fixed length, follow these steps:

1. Run REORG with COPY on the table space, using the inline copy.
2. Run the MODIFY utility with the DELETE option to delete records of all image copies that were made before the REORG you ran in step 1.
3. Create a unique index if you add a column that specifies PRIMARY KEY.

Inserting values in the new column might also degrade performance by forcing rows onto another physical page. You can avoid this situation by creating the table space with enough free space to accommodate normal expansion. If you already have this problem, run REORG on the table space to fix it.

You can define the new column as NOT NULL by using the DEFAULT clause unless the column has a ROWID data type or is an identity column. If the column has a ROWID data type or is an identity column, you must specify NOT NULL without the DEFAULT clause. You can let DB2 choose the default value, or you can specify a constant or the value of the CURRENT SQLID or USER special register as the value to be used as the default. When you retrieve an existing row from the table, a default value is provided for the new column. Except in the following cases, the value for retrieval is the same as the value for insert:

- For columns of data type DATE, TIME, and TIMESTAMP, the retrieval defaults are:

Data type	**Default for retrieval**
DATE	0001-01-01
TIME	00.00.00
TIMESTAMP	0001-01-01-00.00.00.000000

- For DEFAULT USER and DEFAULT CURRENT SQLID, the retrieved value for rows that existed before the column was added is the value of the special register when the column was added.

If the new column is a ROWID column, DB2 returns the same, unique row ID value for a row each time you access that row. Reorganizing a table space does not affect the values on a ROWID column. You cannot use the DEFAULT clause for ROWID columns.

If the new column is an identity column (a column that is defined with the AS IDENTITY clause), DB2 places the table space in REORG-pending (REORP) status, and access to the table space is restricted until the table space is reorganized. When the REORG utility is run, DB2

- Generates a unique value for the identity column of each existing row
- Physically stores these values in the database
- Removes the REORP status

You cannot use the DEFAULT clause for identity columns. For more information about identity columns, see *DB2 SQL Reference*.

If the new column is a short string column, you can specify a field procedure for it; see "Field procedures" on page 1089. If you do specify a field procedure, you cannot also specify NOT NULL.

The following example adds a new column to the table DSN8810.DEPT, which contains a location code for the department. The column name is LOCATION_CODE, and its data type is CHAR (4).

```
ALTER TABLE DSN8810.DEPT
  ADD LOCATION_CODE CHAR (4);
```

Altering the data type of a column

You can use the ALTER TABLE statement to change the data types of columns in existing tables in the following ways:

- Altering the length of fixed-length or varying-length character data types, and the length of fixed-length or varying-length graphic data types
- Switching between fixed-length and varying-length types for character and graphic data
- Switching between compatible numeric data types

In general, DB2 can alter a data type if the data can be converted from the old type to the new type without truncation or without losing arithmetic precision.

Restrictions: The column that you alter cannot be a part of a referential constraint, have a field procedure, be defined as an identity column, or be defined as a column of a materialized query table.

When you alter the data type of a column in a table, DB2 creates a new version for the table space that contains the data rows of the table, as described in "Table space versions" on page 76.

For specific information about valid conversions of data types, see the ALTER TABLE statement in Chapter 5 of *DB2 SQL Reference*. For information about other changes to data types, see "Changing data types by dropping and re-creating the table" on page 88.

What happens to the column

When you change the data type of a column by using the ALTER TABLE statement, the new definition of the column is stored in the catalog. When you retrieve table rows, the columns are materialized in the format that is indicated by the catalog, but the data is not saved in that format. When you change or insert a row, the entire row is saved in the format that is indicated by the catalog. When you reorganize the table space (or perform a load replace), DB2 reloads the data in the table space according to the format of the current definitions in the catalog.

Example: Assume that a table contains basic account information for a small bank. The initial account table was created many years ago in the following manner:

```
CREATE TABLE ACCOUNTS (
   ACCTID       DECIMAL(4,0)   NOT NULL,
   NAME         CHAR(20)       NOT NULL,
   ADDRESS      CHAR(30)       NOT NULL,
   BALANCE      DECIMAL(10,2)  NOT NULL)
IN dbname.tsname;
```

The columns, as currently defined, have the following problems:

- The ACCTID column allows for only 9999 customers.

- The NAME and ADDRESS columns were defined as fixed-length columns, which means that some of the longer values are truncated and some of the shorter values are padded with blanks.
- The BALANCE column allows for amounts up to 99 999 999.99, but inflation rates demand that this column hold larger numbers.

By altering the column data types in the following ways, you can make the columns more appropriate for the data that they contain. The INSERT statement that follows shows the kinds of values that you can now store in the ACCOUNTS table.

```
ALTER TABLE ACCOUNTS ALTER COLUMN NAME    SET DATA TYPE VARCHAR(40);
ALTER TABLE ACCOUNTS ALTER COLUMN ADDRESS SET DATA TYPE VARCHAR(60);
ALTER TABLE ACCOUNTS ALTER COLUMN BALANCE SET DATA TYPE DECIMAL(15,2);
ALTER TABLE ACCOUNTS ALTER COLUMN ACCTID  SET DATA TYPE INTEGER;
COMMIT;

INSERT INTO ACCOUNTS (ACCTID, NAME, ADDRESS, BALANCE)
  VALUES (123456, 'LAGOMARSINO, MAGDALENA',
          '1275 WINTERGREEN ST, SAN FRANCISCO, CA, 95060', 0);
COMMIT;
```

The NAME and ADDRESS columns can now handle longer values without truncation, and the shorter values are no longer padded. The BALANCE column is extended to allow for larger dollar amounts. DB2 saves these new formats in the catalog and stores the inserted row in the new formats.

Recommendation: If you change both the length and the type of a column from fixed-length to varying-length by using one or more ALTER statements, issue the ALTER statements within the same unit of work. Reorganize immediately so that the format is consistent for all of the data rows in the table.

What happens to an index on the column

Altering the data type of a column that is contained in an index has implications for the index.

Example: Assume that the following indexes are defined on the ACCOUNTS table.

```
CREATE INDEX IX1 ON ACCOUNTS(ACCTID);
CREATE INDEX IX2 ON ACCOUNTS(NAME);
```

When the data type of the ACCTID column is altered from DECIMAL(4,0) to INTEGER, the IX1 index is placed in a REBUILD-pending (RBDP) state. Similarly, when the data type of the NAME column is altered from CHAR(20) to VARCHAR(40), the IX2 index is placed in an RBDP state. These indexes cannot be accessed until they are rebuilt from the data.

Index inaccessibility and data availability. Whenever possible, DB2 tries to avoid using inaccessible indexes in an effort to increase data availability. Beginning in Version 8, DB2 allows you to insert into, and delete from, tables that have non-unique indexes which are currently in an RBDP state. DB2 also allows you to delete from tables that have unique indexes which are currently in an RBDP state.

In certain situations, when an index is inaccessible, DB2 can bypass the index to allow applications access to the underlying data. In these situations, DB2 offers accessibility at the expense of possibly-degraded performance. In making its determination of the best access path, DB2 can bypass an index under the following circumstances:

- Dynamic PREPAREs

DB2 avoids choosing an index that is in an RBDP state. Bypassing the index typically degrades performance, but provides availability that would not be possible otherwise.

- Cached PREPAREs

  DB2 avoids choosing an index that is both in an RBDP state and within a cached PREPARE statement because the dynamic cache for the table is invalidated whenever an index is put into, or taken out of, an RBDP state. Avoiding indexes that are within cached PREPARE statements ensures that after an index is rebuilt, DB2 uses the index for all future queries.

- PADDED and NOT PADDED indexes

  DB2 avoids choosing a PADDED index or a NOT PADDED index that is currently in an RBDP state for any static or dynamic SQL statements.

In the case of static BINDs, DB2 might choose an index that is currently in an RBDP state as the best access path. It does so by making the optimistic assumption that the index will be available by the time it is actually used. (If the index is not available at that time, an application can receive a resource unavailable message.)

Padding. Whether an index is padded or not padded depends on when the index was created and whether the index contains any varying length columns. In pre-Version 8 releases, an index is padded by default. In new-function mode for Version 8 or later, an index is not padded by default for new installations of DB2 , but will be padded for migrated systems which came from at least Version 7 forward. In Version 8 or later, this default behavior is specified by the value of the PAD INDEX BY DEFAULT parameter on the DSNTIPE installation panel, which can be set to YES or NO.

When an index is not padded, the value of the PADDED column of the SYSINDEXES table is set to N. An index is only considered not padded when it is created with at least one varying length column and either:

- The NOT PADDED keyword is specified.
- The default padding value is NO.

When an index is padded, the value of the PADDED column of the SYSINDEXES table is set to Y. An index is padded if it is created with at least one varying length column and either:

- The PADDED keyword is specified
- The default padding is YES.

In the example of the ACCOUNTS table, the IX2 index retains its padding attribute. The padding attribute of an index is altered only if the value is inconsistent with the current state of the index. The value can be inconsistent, for example, if you change the value of the PADDED column in the SYSINDEXES table after creating the index.

Consider the following information when you migrate indexes from one version of DB2 to the next version, or when you install a new DB2 subsystem and create indexes:

- If the index was migrated from a pre-Version 8 release, the index is padded by default. In this case, the value of the PADDED column of the SYSINDEXES table is blank (*PADDED* = ' '). The PADDED column is also blank when there are no varying length columns.

- If a subsystem has been migrated from Version 7 to Version 8 compatibility mode or new-function mode, the default is to pad all indexes that have a key with at least one varying length column. In this case, the value of the PADDED column of the SYSINDEXES table is YES (*PADDED = 'Y'*).
- If an installed subsystem is new to Version 8 or later, and the install is done directly into new-function mode for Version 8 or later, indexes created with at least one varying length column are not padded by default. In this case, the PADDED column of the SYSINDEXES table is set to NO (*PADDED = 'N'*).

Table space versions

DB2 creates a table space version each time you commit one or more of the following schema changes by using the ALTER TABLE statement:

- Extend the length of a character (char data type) or graphic (graphic data type) column
- Change the type of a column within character data types (char, varchar)
- Change the type of a column within graphic data types (graphic, vargraphic)
- Change the type of a column within numeric data types (small integer, integer, float, real, float8, double, decimal)
- Add a column to a table that has a version number greater than 0
- Extend the length of a varying character (varchar data type) or varying graphic (vargraphic data type) column, if the table is already versioned.

Exceptions: DB2 does not create a table space version under the following circumstances:

- When the table space was created with DEFINE NO
- When you extend the length of a varying character (varchar data type) or varying graphic (vargraphic data type) column, and the table is not already versioned.
- When you specify the same data type and length that a column currently has, such that its definition does not actually change

DB2 creates only one table space version if, in the same unit of work, you make multiple schema changes. If you make these same schema changes in separate units of work, each change results in a new table space version. For example, the first three ALTER TABLE statements in the following example are all associated with the same table space version. The scope of the first COMMIT statement encompasses all three schema changes. The last ALTER TABLE statement is associated with the next table space version. The scope of the second COMMIT statement encompasses a single schema change.

```
ALTER TABLE ACCOUNTS ALTER COLUMN NAME    SET DATA TYPE VARCHAR(40);
ALTER TABLE ACCOUNTS ALTER COLUMN ADDRESS SET DATA TYPE VARCHAR(60);
ALTER TABLE ACCOUNTS ALTER COLUMN BALANCE SET DATA TYPE DECIMAL(15,2);
COMMIT;

ALTER TABLE ACCOUNTS ALTER COLUMN ACCTID  SET DATA TYPE INTEGER;
COMMIT;
```

Reorganizing table spaces: After you commit a schema change, DB2 puts the affected table space into an advisory REORG-pending (AREO*) state. The table space stays in this state until you run the REORG TABLESPACE utility, which reorganizes the table space and applies the schema changes.

DB2 uses table space versions to maximize data availability. Table space versions enable DB2 to keep track of schema changes and simultaneously, provide users

with access to data in altered table spaces. When users retrieve rows from an altered table, the data is displayed in the format that is described by the most recent schema definition, even though the data is not currently stored in this format. The most recent schema definition is associated with the *current* table space version.

Although data availability is maximized by the use of table space versions, performance might suffer because DB2 does not automatically reformat the data in the table space to conform to the most recent schema definition. DB2 defers any reformatting of existing data until you reorganize the table space with the REORG TABLESPACE utility. The more ALTER statements you commit between reorganizations, the more table space versions DB2 must track, and the more performance can suffer.

Recommendation: Run the REORG TABLESPACE utility as soon as possible after a schema change to correct any performance degradation that might occur and to keep performance at its highest level.

Recycling table space version numbers: DB2 can store up to 256 table space versions, numbered sequentially from 0 to 255. (The next consecutive version number after 255 is 1. Version number 0 is never reused; it is reserved for the original version of the table space.) The versions that are associated with schema changes that have not been applied yet are considered to be "in use," and the range of used versions is stored in the catalog. In-use versions can be recovered from image copies of the table space, if necessary. To determine the range of version numbers currently in use for a table space, query the OLDEST_VERSION and CURRENT_VERSION columns of the SYSIBM.SYSTABLESPACE catalog table.

To prevent DB2 from running out of table space version numbers (and to prevent subsequent ALTER statements from failing), you must recycle unused table space version numbers regularly by running the MODIFY RECOVERY utility. Version numbers are considered to be "unused" if the schema changes that are associated with them have been applied and there are no image copies that contain data at those versions. If all reusable version numbers (1 to 255) are currently in use, you must reorganize the table space by running REORG TABLESPACE before you can recycle the version numbers. For more information about managing table space version numbers, see *DB2 Utility Guide and Reference*.

Altering a table for referential integrity

You can alter a table to add, change, or remove referential constraints. If you plan to let DB2 enforce referential integrity in a set of tables, see Part 2 of *DB2 Application Programming and SQL Guide* for a description of the requirements for referential constraints. DB2 does not enforce informational referential constraints.

This section discusses these topics:
- "Adding referential constraints to existing tables"
- "Adding parent keys and foreign keys" on page 78
- "Dropping parent keys and foreign keys" on page 79

Adding referential constraints to existing tables

Assume that the tables in the sample application already exist, have the appropriate column definitions, and are already populated. This section refers to the DB2 sample activity, project, project activity, employee, and department tables, which are described in Appendix A (Volume 2) of *DB2 Administration Guide*.

Suppose that you want to define relationships among the sample tables by adding primary and foreign keys with the ALTER TABLE statement. The rules for these relationships are as follows:

- An existing table must have a unique index on its primary key columns before you can add the primary key. The index becomes the primary index.
- The parent key of the parent table must be added before the corresponding foreign key of the dependent table.

You can build the same referential structure in several different ways. The following example sequence does not have the fewest number of possible operations, but it is perhaps the simplest to understand.

1. Create a unique index on the primary key columns for any table that does not already have one.
2. For each table, issue an ALTER TABLE statement to add its primary key.

   In the next steps, you issue an ALTER TABLE statement to add foreign keys for each table except the activity table. This leaves the table space in CHECK-pending status, which you reset by running the CHECK DATA utility with the DELETE(YES) option.

   CHECK DATA deletes are not bound by delete rules; they cascade to all descendents of a deleted row, which can be disastrous. For example, if you delete the row for department A00 from the department table, the delete might propagate through most of the referential structure. The remaining steps prevent deletion from more than one table at a time.
3. Add the foreign keys for the department table and run CHECK DATA DELETE(YES) on its table space. Correct any rows in the exception table, and use INSERT to replace them in the department table. This table is now consistent with existing data.
4. Drop the foreign key on MGRNO in the department table. This drops the association of the department table with the employee table, without changing the data of either table.
5. Add the foreign key to the employee table, run CHECK DATA again, and correct any errors. If errors are reported, be particularly careful not to make any row inconsistent with the department table when you make corrections.
6. Again, add the foreign key on MGRNO to the department table, which again leaves the table space in CHECK-pending status. Run CHECK DATA. If you have not changed the data since the previous check, you can use DELETE(YES) with no fear of cascading deletions.
7. For each of the following tables, in the order shown, add its foreign keys, run CHECK DATA DELETE(YES), and correct any rows that are in error:
   a. Project table
   b. Project activity table
   c. Employee to project activity table

Adding parent keys and foreign keys

You can add parent keys, both primary and unique, and foreign keys to an existing table.

Adding a primary key: To add a primary key to an existing table, use the PRIMARY KEY clause in an ALTER TABLE statement. For example, if the department table and its index XDEPT1 already exist, create its primary key by issuing the following statement:

```
ALTER TABLE DSN8810.DEPT
   ADD PRIMARY KEY (DEPTNO);
```

Adding a unique key: To add a unique key to an existing table, use the UNIQUE clause of the ALTER TABLE statement. For example, if the department table has a unique index defined on column DEPTNAME, you can add a unique key constraint, KEY_DEPTNAME, consisting of column DEPTNAME by issuing the following statement:

```
ALTER TABLE DSN8810.DEPT
   ADD CONSTRAINT KEY_DEPTNAME UNIQUE (DEPTNAME);
```

Adding a foreign key: To add a foreign key to an existing table, use the FOREIGN KEY clause of the ALTER TABLE statement. The parent key must exist in the parent table before you add the foreign key. For example, if the department table has a primary key defined on the DEPTNO column, you can add a referential constraint, REFKEY_DEPTNO, on the DEPTNO column of the project table by issuing the following statement:

```
ALTER TABLE DSN8810.PROJ
   ADD CONSTRAINT REFKEY_DEPTNO FOREIGN KEY (DEPTNO) REFERENCES DSN8810.DEPT
   ON DELETE RESTRICT;
```

Considerations: Adding a parent key or a foreign key to an existing table has the following restrictions and implications:

- If you add a primary key, the table must already have a unique index on the key columns. If multiple unique indexes include the primary key columns, the index that was most recently created on the key columns becomes the primary index. Because of the unique index, no duplicate values of the key exist in the table; therefore you do not need to check the validity of the data.
- If you add a unique key, the table must already have a unique index with a key that is identical to the unique key. If multiple unique indexes include the primary key columns, DB2 arbitrarily chooses a unique index on the key columns to enforce the unique key. Because of the unique index, no duplicate values of the key exist in the table; therefore you do not need to check the validity of the data.
- You can use only one FOREIGN KEY clause in each ALTER TABLE statement; if you want to add two foreign keys to a table, you must execute two ALTER TABLE statements.
- If you add a foreign key, the parent key and unique index of the parent table must already exist. Adding the foreign key requires the ALTER privilege on the dependent table and either the ALTER or REFERENCES privilege on the parent table.
- Adding a foreign key establishes a referential constraint relationship, with the many implications that are described in Part 2 of *DB2 Application Programming and SQL Guide*. DB2 does not validate the data when you add the foreign key. Instead, if the table is populated (or, in the case of a nonsegmented table space, if the table space has ever been populated), the table space that contains the table is placed in CHECK-pending status, just as if it had been loaded with ENFORCE NO. In this case, you need to execute the CHECK DATA utility to clear the CHECK-pending status.
- You can add a foreign key with the NOT ENFORCED option to create an informational referential constraint. This action does not leave the table space in CHECK-pending status, and you do not need to execute CHECK DATA.

Dropping parent keys and foreign keys

You can drop parent keys, both primary and unique, and foreign keys from an existing table. Before you drop a foreign key or a parent key, consider carefully the effects on your application programs. The primary key of a table serves as a permanent, unique identifier of the occurrences of the entities it describes.

Application programs often depend on that identifier. The foreign key defines a referential relationship and a delete rule. Without the key, your application programs must enforce the constraints.

Dropping a foreign key: When you drop a foreign key using the DROP FOREIGN KEY clause of the ALTER TABLE statement, DB2 drops the corresponding referential relationships. (You must have the ALTER privilege on the dependent table and either the ALTER or REFERENCES privilege on the parent table.) If the referential constraint references a unique key that was created implicitly, and no other relationships are dependent on that unique key, the implicit unique key is also dropped.

Dropping a unique key: When you drop a unique key using the DROP UNIQUE clause of the ALTER TABLE statement, DB2 drops all the referential relationships in which the unique key is a parent key. The dependent tables no longer have foreign keys. (You must have the ALTER privilege on any dependent tables.) The table's unique index that enforced the unique key no longer indicates that it enforces a unique key, although it is still a unique index.

Dropping a primary key: When you drop a primary key using the DROP PRIMARY KEY clause of the ALTER TABLE statement, DB2 drops all the referential relationships in which the primary key is a parent key. The dependent tables no longer have foreign keys. (You must have the ALTER privilege on any dependent tables.) The table's primary index is no longer primary, although it is still a unique index.

Adding or dropping table check constraints

You can define a check constraint on a table by using the ADD CHECK clause of the ALTER TABLE statement. If the table is empty, the check constraint is added to the description of the table.

If the table is not empty, what happens when you define the check constraint depends on the value of the CURRENT RULES special register, which can be either STD or DB2.

- If the value is STD, the check constraint is enforced immediately when it is defined. If a row does not conform, the table check constraint is not added to the table and an error occurs.
- If the value is DB2, the check constraint is added to the table description but its enforcement is deferred. Because some rows in the table might violate the check constraint, the table is placed in check-pending status.

The ALTER TABLE statement that is used to define a check constraint always fails if the table space or partition that contains the table is in a CHECK-pending status, the CURRENT RULES special register value is STD, and the table is not empty.

To remove a check constraint from a table, use the DROP CONSTRAINT or DROP CHECK clauses of the ALTER TABLE statement. You must not use DROP CONSTRAINT on the same ALTER TABLE statement as DROP FOREIGN KEY, DROP CHECK, DROP PRIMARY KEY, or DROP UNIQUE.

Adding a partition

You can use the ALTER TABLE statement to add a partition to an existing partitioned table space and to each partitioned index in the table space.

Restriction: You cannot add a new partition to an existing partitioned table space if the table has LOB columns. Additionally, you cannot add or alter a partition for a materialized query table.

When you add a partition, DB2 uses the next physical partition that is not already in use until you reach the maximum number of partitions for the table space. When DB2 manages your data sets, the next available data set is allocated for the table space and for each partitioned index. When you manage your own data sets, you must first define the data sets for the table space and the partitioned indexes before issuing the ALTER TABLE ADD PARTITION statement.

Example: Assume that a table space that contains a transaction table named TRANS is divided into 10 partitions, and each partition contains one year of data. Partitioning is defined on the transaction date, and the limit key value is the end of the year. Table 18 shows a representation of the table space.

Table 18. Initial table space with 10 partitions

Partition	Limit value	Data set name that backs the partition
P001	12/31/1994	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A001
P002	12/31/1995	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A002
P003	12/31/1996	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A003
P004	12/31/1997	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A004
P005	12/31/1998	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A005
P006	12/31/1999	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A006
P007	12/31/2000	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A007
P008	12/31/2001	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A008
P009	12/31/2002	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A009
P010	12/31/2003	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A010

Assume that you want to add a new partition to handle the transactions for the next year. To add a partition, issue the following statement:

```
ALTER TABLE TRANS ADD PARTITION ENDING AT ('12/31/2004');
```

What happens:

- DB2 adds a new partition to the table space and to each partitioned index on the TRANS table. For the table space, DB2 uses the existing table space PRIQTY and SECQTY attributes of the previous partition for the space attributes of the new partition. For each partitioned index, DB2 uses the existing PRIQTY and SECQTY attributes of the previous index partition.
- When the ALTER completes, you can use the new partition immediately if the table space is a large table space. In this case, the partition is not placed in a REORG-pending (REORP) state because it extends the high-range values that were not previously used. For non-large table spaces, the partition **is** placed in a REORG-pending (REORP) state because the last partition boundary was not previously enforced.

Table 19 on page 82 shows a representative excerpt of the table space after the partition for the year 2004 was added.

Table 19. An excerpt of the table space after adding a new partition (P011)

Partition	Limit value	Data set name that backs the partition
P008	12/31/2001	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A008
P009	12/31/2002	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A009
P010	12/31/2003	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A010
P011	12/31/2004	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A011

Specifying space attributes: If you want to specify the space attributes for a new partition, use the ALTER TABLESPACE and ALTER INDEX statements. For example, suppose the new partition is PARTITION 11 for the table space and the index. Issue the following statements to specify quantities for the PRIQTY, SECQTY, FREEPAGE, and PCTFREE attributes:

```
ALTER TABLESPACE tsname ALTER PARTITION 11
   USING STOGROUP stogroup-name
    PRIQTY 200 SECQTY 200
    FREEPAGE 20 PCTFREE 10;

ALTER INDEX index-name ALTER PARTITION 11
   USING STOGROUP stogroup-name
    PRIQTY 100 SECQTY 100
    FREEPAGE 25 PCTFREE 5;
```

Recommendation: When you create your partitioned table space, you do not need to allocate extra partitions for expected growth. Instead, use ALTER TABLE ADD PARTITION to add partitions as needed.

Altering partitions

You can use the ALTER TABLE statement to change the boundary between partitions, to rotate the first partition to be the last partition, or to extend the boundary of the last partition.

Example: Assume that a table space that contains a transaction table named TRANS is divided into 10 partitions, and each partition contains one year of data. Partitioning is defined on the transaction date, and the limit key value is the end of the year. Table 18 on page 81 shows a representation of the table space.

Changing the boundary between partitions

Assume that the year 2003 resulted in more data than was projected so that the allocation for partition 10 reached its maximum of 4 GB. In addition, the year 2002 resulted in less data than was projected. You want to change the boundary between partition 9 and partition 10 so that some of the data in partition 10 becomes part of the data in partition 9.

To change the boundary, issue the following statement:

```
ALTER TABLE TRANS ALTER PARTITION 9 ENDING AT ('03/31/2003');
```

Now the data in the first quarter of the year 2003 is part of partition 9. The partitions on either side of the new boundary (partitions 9 and 10) are placed in REORG-pending (REORP) status and are not available until the partitions are reorganized.

Alternatively, you can rebalance the data in partitions 9 and 10 by using the REBALANCE option of the REORG utility:

```
REORG TABLESPACE dbname.tsname PART(9:10) REBALANCE
```

This method avoids leaving the partitions in a REORP state. When you use the REBALANCE option on partitions, DB2 automatically changes the limit key values.

Rotating partitions

Assume that the partition structure of the table space is sufficient through the year 2004. Table 19 on page 82 shows a representation of the table space through the year 2004. When another partition is needed for the year 2005, you determine that the data for 1994 is no longer needed. You want to recycle the partition for the year 1994 to hold the transactions for the year 2005.

To rotate the first partition to be the last partition, issue the following statement:

```
ALTER TABLE TRANS ROTATE PARTITION FIRST TO LAST
   ENDING AT ('12/31/2005') RESET;
```

For a table with limit values in ascending order, the data in the ENDING AT clause must be higher than the limit value for previous partitions. DB2 chooses the first partition to be the partition with the lowest limit value.

For a table with limit values in descending order, the data must be lower than the limit value for previous partitions. DB2 chooses the first partition to be the partition with the highest limit value.

The RESET keyword specifies that the existing data in the oldest partition is deleted, and no delete triggers are activated.

What happens:

- Because the oldest (or first) partition is P001, DB2 assigns the new limit value to P001. This partition holds all rows in the range between the new limit value of 12/31/2005 and the previous limit value of 12/31/2004.
- The RESET operation deletes all existing data. You can use the partition immediately after the ALTER completes. The partition is not placed in REORG-pending (REORP) status because it extends the high-range values that were not previously used.

Table 20 shows a representation of the table space after the first partition is rotated to become the last partition.

Table 20. Rotating the low partition to the end

Partition	Limit value	Data set name that backs the partition
P002	12/31/1995	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A002
P003	12/31/1996	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A003
P004	12/31/1997	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A004
P005	12/31/1998	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A005
P006	12/31/1999	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A006
P007	12/31/2000	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A007
P008	12/31/2001	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A008
P009	12/31/2002	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A009
P010	12/31/2003	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A010
P011	12/31/2004	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A011
P001	12/31/2005	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A001

Recommendation: When you create a partitioned table space, you do not need to allocate extra partitions for expected growth. Instead, use either ALTER TABLE ADD PARTITION to add partitions as needed or, if rotating partitions is appropriate for your application, use ALTER TABLE ROTATE PARTITION to avoid adding another partition.

Nullable partitioning columns: DB2 lets you use nullable columns as partitioning columns. But with table-controlled partitioning, DB2 can restrict the insertion of null values into a table with nullable partitioning columns, depending on the order of the partitioning key. After a rotate operation:

- If the partitioning key is ascending, DB2 prevents an INSERT of a row with a null value for the key column.
- If the partitioning key is descending, DB2 allows an INSERT of a row with a null value for the key column. The row is inserted into the first partition.

Extending the boundary of the last partition

Assume that a decision has been made to save the data for the year 2005 and the data for the year 2006 into one partition. You have rotated the first partition to be the last partition. Table 20 on page 83 shows a representation of the table space through the year 2005.

To extend the boundary of the last partition, issue the following statement:

```
ALTER TABLE TRANS ALTER PARTITION 1 ENDING AT ('12/31/2006');
```

You can use the partition immediately after the ALTER completes. The partition is not placed in REORG-pending (REORP) status because it extends the high-range values that were not previously used.

Changing back to the previous boundary: Table 20 on page 83 shows a representation of the table space through the year 2005, where each year of data is saved into separate partitions. Assume that you changed the limit key for P001 to be 12/31/2006 so that the data for the year 2005 and the data for the year 2006 is saved into one partition.

To change the limit key back to 12/31/2005, issue the following statement:

```
ALTER TABLE TRANS ALTER PARTITION 1 ENDING AT ('12/31/2005');
```

This partition is placed in REORG-pending (REORP) status because some of the data could fall outside of the boundary that is defined by the limit key value of 12/31/2005. You can take either of the following corrective actions:

- Run REORG with the DISCARD option to clear the REORG-pending status, set the new partition boundary, and discard the data rows that fall outside of the new boundary.
- Add a new partition for the data rows that fall outside of the current partition boundaries.

Adding a partition when the last partition is in REORP: Assume that you extended the boundary of the last partition and then changed back to the previous boundary for that partition. Table 20 on page 83 shows a representation of the table space through the year 2005. The last partition is in REORP.

You want to add a new partition with a limit key value of 12/31/2006. You can use ALTER TABLE ADD PARTITION because this limit key value is higher than the previous limit key value of 12/31/2005.

Issue the following statement:

```
ALTER TABLE TRANS ADD PARTITION ENDING AT ('12/31/2006');
```

The new partition is placed in REORG-pending (REORP) status because it inherits the REORP state from the previous partition. You can now reorganize the table space or only the last two partitions without discarding any of the data rows.

Registering or changing materialized query tables

You can use the ALTER TABLE statement to register an existing table as a materialized query table, to change a materialized query table to a base table, or to change the attributes of a materialized query table.

Materialized query tables enable DB2 to use automatic query rewrite to optimize queries. Automatic query rewrite is a process that DB2 uses to examine a query and, if appropriate, to rewrite the query so that it executes against a materialized query table that has been derived from the base tables in the submitted query. For additional information about the use of automatic query rewrite in query optimization, see Part 5 (Volume 2) of *DB2 Administration Guide*.

Registering an existing table as a materialized query table

Assume that you have a very large transaction table named TRANS that contains one row for each transaction. The table has many columns, but you are interested in only the following columns:

- ACCTID, which is the customer account ID
- LOCID, which is the customer location ID
- YEAR, which holds the year of the transaction

You created another base table named TRANSCOUNT that consists of these columns and a count of the number of rows in TRANS that are grouped by the account, location, and year of the transaction. Suppose that you repopulate TRANSCOUNT periodically by deleting the rows and then by using the following INSERT statement:

```
INSERT INTO TRANSCOUNT (ACCTID, LOCID, YEAR, CNT)
   SELECT ACCTID, LOCID, YEAR, COUNT(*)
   FROM TRANS
   GROUP BY ACCTID, LOCID, YEAR;
```

You want to take advantage of automatic query rewrite for TRANSCOUNT by registering it as a materialized query table. You can do this by issuing the following ALTER TABLE statement:

```
ALTER TABLE TRANSCOUNT ADD MATERIALIZED QUERY AS (
   SELECT ACCTID, LOCID, YEAR, COUNT(*) AS CNT
   FROM TRANS
   GROUP BY ACCTID, LOCID, YEAR )
   DATA INITIALLY DEFERRED
   REFRESH DEFERRED
   MAINTAINED BY USER;
```

This statement registers TRANSCOUNT with its associated subselect as a materialized query table, and DB2 can now use it in automatic query rewrite. The data in TRANSCOUNT remains the same, as specified by the DATA INITIALLY DEFERRED option.

You can still maintain the data, as specified by the MAINTAINED BY USER option, which means that you can continue to load, insert, update, or delete data. You can also use the REFRESH TABLE statement to populate the table. REFRESH

DEFERRED indicates that the data in the table is the result of your most recent update or, if more recent, the result of a REFRESH TABLE statement.

The REFRESH TABLE statement deletes all the rows in a materialized query table, executes the fullselect in its definition, inserts the result into the table, and updates the catalog with the refresh timestamp and cardinality of the table. For more information about the REFRESH TABLE statement, see Chapter 5 of *DB2 SQL Reference*.

Changing a materialized query table to a base table

You can use the ALTER TABLE statement to change a materialized query table into a base table. For example:

```
ALTER TABLE TRANSCOUNT DROP MATERIALIZED QUERY;
```

After you issue this statement, DB2 can no longer use the table for query optimization, and you cannot populate the table by using the REFRESH TABLE statement.

Changing the attributes of a materialized query table

You can use the ALTER TABLE statement to change the following attributes of an existing materialized query table:

- You can enable or disable automatic query rewrite.

  By default, when you create or register a materialized query table, DB2 enables it for automatic query rewrite. To disable automatic query rewrite, issue the following statement:

  ```
  ALTER TABLE TRANSCOUNT DISABLE QUERY OPTIMIZATION;
  ```

  To change back to automatic query rewrite, use the ENABLE QUERY OPTIMIZATON option.
- You can switch between system-maintained and user-maintained.

  By default, a materialized query table is system-maintained; the only way you can change the data is by using the REFRESH TABLE statement. To change to a user-maintained materialized query table, issue the following statement:

  ```
  ALTER TABLE TRANSCOUNT SET MAINTAINED BY USER;
  ```

  To change back to a system-maintained materialized query table, use the MAINTAINED BY SYSTEM option.

Changing the definition of a materialized query table

After you create a materialized query table, you can change the definition, as specified by the fullselect, in one of the following ways:

- Drop and re-create the materialized query table with a different definition.
- Use ALTER TABLE to change the materialized query table into a base table. Then, change it back to a materialized query table with a different but equivalent definition (that is, with a different but equivalent SELECT for the query).

Altering the assignment of a validation routine

If you have a validation exit routine associated with a table, you can use the ALTER TABLE statement to make the following changes:

- Disassociate the validation routine from the table using the VALIDPROC NULL clause. The routine is no longer given control when DB2 accesses the table. For example:

  ```
  ALTER TABLE DSN8810.EMP
    VALIDPROC NULL;
  ```

- Assign a new validation routine to the table using the VALIDPROC clause. (Only one validation routine can be connected to a table at a time; so if a validation routine already exists, DB2 disconnects the old one and connects the new routine.) Rows that existed before the connection of a new validation routine are not validated. In this example, the previous validation routine is disconnected and a new routine is connected with the program name EMPLNEWE:

  ```
  ALTER TABLE DSN8810.EMP
    VALIDPROC EMPLNEWE;
  ```

To ensure that the rows of a table conform to a new validation routine, you must run the validation routine against the old rows. One way to accomplish this is to use the REORG and LOAD utilities as shown in the following steps:

1. Use REORG to reorganize the table space that contains the table with the new validation routine. Specify UNLOAD ONLY, as in this example:

   ```
   REORG TABLESPACE DSN8D81A.DSN8S81E
     UNLOAD ONLY
   ```

   This step creates a data set that is used as input to the LOAD utility.
2. Run LOAD with the REPLACE option, and specify a discard data set to hold any invalid records. For example:

   ```
   LOAD INTO TABLE DSN8810.EMP
      REPLACE
      FORMAT UNLOAD
      DISCARDDN SYSDISC
   ```

The EMPLNEWE validation routine validates all rows after the LOAD step has completed. DB2 copies any invalid rows into the SYSDISC data set.

Altering a table for capture of changed data

You can use the DATA CAPTURE CHANGES option on the ALTER TABLE statement to have data changes to that table written to the log in an expanded format. You can then retrieve the log by using a program such as the log apply feature of the Remote Recovery Data Facility (RRDF) program offering, or DB2 DataPropagator.

LOB values are not available for DATA CAPTURE CHANGES. To return a table back to normal logging, use DATA CAPTURE NONE.

Changing an edit procedure or a field procedure

You cannot use ALTER TABLE to change the assignment of an edit procedure or a field procedure. However, with the assistance of DB2 utilities, you can change an existing edit procedure or field procedure.

To change an edit procedure or a field procedure for a table space in which the maximum record length is less than 32 KB, use the following procedure:

1. Run the UNLOAD utility or run the REORG TABLESPACE utility with the UNLOAD EXTERNAL option to unload the data and decode it using the existing edit procedure or field procedure.

   These utilities generate a LOAD statement in the data set (specified by the PUNCHDDN option of the REORG TABLESPACE utility) that you can use to reload the data into the original table space.

   If you are using the same edit procedure or field procedure for many tables, unload the data from all the table spaces that have tables that use the procedure.

2. Modify the code of the edit procedure or the field procedure.
3. After the unload operation is completed, stop DB2.
4. Link-edit the modified procedure, using its original name.
5. Start DB2.
6. Use the LOAD utility to reload the data. LOAD then uses the modified procedure or field procedure to encode the data.

To change an edit procedure or a field procedure for a table space in which the maximum record length is greater than 32 KB, use the DSNTIAUL sample program to unload the data.

Altering the subtype of a string column

If you add a column with a string data type, you can specify its subtype in the ALTER TABLE statement. Subtypes are valid for string columns of data types CHAR and VARCHAR. SBCS and MIXED are valid subtypes for CLOB data.

The interpretation of the FOREIGNKEY column depends on whether the MIXED DATA install option is YES or NO.

Entering an M in the column when the MIXED DATA install option is NO specifies SBCS data, not MIXED data.

1. To alter the subtype of an existing string column, enter the following SQL statement:

```
ALTER TABLE table-name ALTER COLUMN column-name
      SET DATA TYPE altered-data-type
```

Altering the attributes of an identity column

At some point, you might need to change the attributes of an identity column. For example, you might want to allow or disallow identity column values to wrap. You can use the ALTER TABLE with the ALTER COLUMN clause to change all of the attributes of an identity column except the data type. However, if the ALTER TABLE statement is rolled back, a gap in the sequence of identity column values can occur because of unassigned cache values.

Changing the data type of an identity column, like changing some other data types, requires that you drop and then re-create the table (see “Changing data types by dropping and re-creating the table”).

See Chapter 5 of *DB2 SQL Reference* for more information about identity column attributes.

Changing data types by dropping and re-creating the table

Some changes to a table cannot be made with the ALTER TABLE statement. For example, you must make the following changes by redefining the column (that is, dropping the table and then re-creating the table with the new definitions):

- An original specification of CHAR (25) to CHAR (20)
- A column defined as INTEGER to SMALLINT
- A column defined as NOT NULL to allow null values
- The data type of an identity column

To make such changes, you need to perform the following steps:

1. Unload the table.
2. Drop the table.

Be very careful about dropping a table; in most cases, recovering a dropped table is nearly impossible. If you decide to drop a table, remember that such changes might invalidate a plan or a package.

You must alter tables that have been created with RESTRICT ON DROP to remove the restriction before you can drop them.

3. Commit the changes.
4. Re-create the table.

   If the table has an identity column:

   - Choose carefully the new value for the START WITH attribute of the identity column in the CREATE TABLE statement if you want the first generated value for the identity column of the new table to resume the sequence after the last generated value for the table that was saved by the unload in step 1.
   - Define the identity column as GENERATED BY DEFAULT so that the previously generated identity values can be reloaded into the new table.
5. Reload the table.

Implications of dropping a table

The DROP TABLE statement deletes a table. For example, to drop the project table, execute the following statement:

```
DROP TABLE DSN8810.PROJ;
```

The statement deletes the row in the SYSIBM.SYSTABLES catalog table that contains information about DSN8810.PROJ. It also drops any other objects that depend on the project table. As a result:

- The column names of the table are dropped from SYSIBM.SYSCOLUMNS.
- If the dropped table has an identity column, the sequence attributes of the identity column are removed from SYSIBM.SYSSEQUENCES.
- If triggers are defined on the table, they are dropped, and the corresponding rows are removed from SYSIBM.SYSTRIGGERS and SYSIBM.SYSPACKAGES.
- Any views based on the table are dropped.
- Application plans or packages that involve the use of the table are invalidated.
- Cached dynamic statements that involve the use of the table are removed from the cache.
- Synonyms for the table are dropped from SYSIBM.SYSSYNONYMS.
- Indexes created on any columns of the table are dropped.
- Referential constraints that involve the table are dropped. In this case, the project table is no longer a dependent of the department and employee tables, nor is it a parent of the project activity table.
- Authorization information that is kept in the DB2 catalog authorization tables is updated to reflect the dropping of the table. Users who were previously authorized to use the table, or views on it, no longer have those privileges, because catalog rows are deleted.
- Access path statistics and space statistics for the table are deleted from the catalog.
- The storage space of the dropped table might be reclaimed.

  If the table space containing the table is:

  – Implicitly created (using CREATE TABLE without the TABLESPACE clause), the table space is also dropped. If the data sets are in a storage group, dropping the table space reclaims the space. For user-managed data sets, you must reclaim the space yourself.

- Partitioned, or contains only the one table, you can drop the table space.
- Segmented, DB2 reclaims the space.
- Simple, and contains other tables, you must run the REORG utility to reclaim the space.

- If the table contains a LOB column, the auxiliary table and the index on the auxiliary table are dropped. The LOB table space is dropped if it was created with SQLRULES(STD). See *DB2 SQL Reference* for details.

If a table has a partitioning index, you must drop the table space or use LOAD REPLACE when loading the redefined table. If the CREATE TABLE that is used to redefine the table creates a table space implicitly, commit the DROP statement before re-creating a table by the same name. You must also commit the DROP statement before you create any new indexes with the same name as the original indexes.

Check objects that depend on the table

Before dropping a table, check to see what other objects are dependent on it. For example, the DB2 catalog tables SYSIBM.SYSVIEWDEP, SYSIBM.SYSPLANDEP, and SYSIBM.SYSPACKDEP indicate what views, application plans, and packages are dependent on different DB2 objects.

Finding dependent views: The following example query lists the views, with their creators, that are affected if you drop the project table:

```
SELECT DNAME, DCREATOR
  FROM SYSIBM.SYSVIEWDEP
  WHERE BNAME = 'PROJ'
  AND BCREATOR = 'DSN8810'
  AND BTYPE = 'T';
```

Finding dependent packages: The next example lists the packages, identified by the package name, collection ID, and consistency token (in hexadecimal representation), that are affected if you drop the project table:

```
SELECT DNAME, DCOLLID, HEX(DCONTOKEN)
  FROM SYSIBM.SYSPACKDEP
  WHERE BNAME = 'PROJ'
  AND BQUALIFIER = 'DSN8810'
  AND BTYPE = 'T';
```

Finding dependent plans: The next example lists the plans, identified by plan name, that are affected if you drop the project table:

```
SELECT DNAME
  FROM SYSIBM.SYSPLANDEP
  WHERE BNAME = 'PROJ'
  AND BCREATOR = 'DSN8810'
  AND BTYPE = 'T';
```

Finding other dependencies: In addition, the SYSIBM.SYSINDEXES table tells you what indexes currently exist on a table. From the SYSIBM.SYSTABAUTH table, you can determine which users are authorized to use the table. For more information about what you can retrieve from the DB2 catalog tables, see Appendix G of *DB2 SQL Reference*.

Re-creating a table

To re-create a DB2 table to decrease the length attribute of a string column or the precision of a numeric column, follow these steps:

1. If you do not have the original CREATE TABLE statement and all authorization statements for the table (call it T1), query the catalog to determine its description, the description of all indexes and views on it, and all users with privileges on it.
2. Create a new table (call it T2) with the desired attributes.
3. Copy the data from the old table T1 into the new table T2 by using one of the following methods:
   - Execute the following INSERT statement:
     ```
     INSERT INTO T2
       SELECT * FROM T1;
     ```
   - Load data from your old table into the new table by using the INCURSOR option of the LOAD utility. This option uses the DB2 UDB family cross-loader function.
4. Execute the statement DROP TABLE T1. If T1 is the only table in an explicitly created table space, and you do not mind losing the compression dictionary, if one exists, drop the table space instead, so that the space is reclaimed.
5. Commit the DROP statement.
6. Use the statement RENAME TABLE to rename table T2 to T1.
7. Run the REORG utility on the table space that contains table T1.
8. Notify users to re-create any synonyms, indexes, views, and authorizations they had on T1.

If you want to change a data type from string to numeric or from numeric to string (for example, INTEGER to CHAR or CHAR to INTEGER), use the CHAR and DECIMAL scalar functions in the SELECT statement to do the conversion. Another alternative is to use the following method:

1. Use UNLOAD or REORG UNLOAD EXTERNAL (if the data to unload in less than 32 KB) to save the data in a sequential file, and then
2. Use the LOAD utility to repopulate the table after re-creating it. When you reload the table, make sure you edit the LOAD statement to match the new column definition.

This method is particularly appealing when you are trying to re-create a large table.

Moving a table to a table space of a different page size

You cannot alter a table to use a different page size. However, you can move a table to a table space of a different page size:

1. Unload the table using UNLOAD FROM TABLE or REORG UNLOAD EXTERNAL FROM TABLE.
2. Use CREATE TABLE LIKE on the table to re-create it in the table space of the new page size.
3. Use DB2 Control Center, DB2 Administration Tool, or catalog queries to determine the dependent objects: views, authorization, plans, packages, synonyms, triggers, referential integrity, and indexes.
4. Drop the original table.
5. Rename the new table to the name of the old table using RENAME TABLE.
6. Re-create all dependent objects.
7. Rebind plans and packages.

8. Reload the table using data from the SYSRECnn data set and the control statements from the SYSPUNCH data set, which was created when the table was unloaded.

Altering indexes

You can add a new column to an index or change most of the index clauses for an existing index with the ALTER INDEX statement, including BUFFERPOOL, CLOSE, COPY, PIECESIZE, PADDED or NOT PADDED, CLUSTER or NOT CLUSTER, and those clauses that are associated with storage space, free space, and group buffer pool caching.

With the ALTER INDEX statement, you can:

- Add a new column to an index; see "Adding a new column to an index."
- Alter the PADDED or NOT PADDED attribute to change how varying-length columns are stored in the index; see "Altering how varying-length index columns are stored" on page 93.
- Alter the CLUSTER or NOT CLUSTER attribute to change how data is stored; see "Altering the clustering index" on page 93.
- Change the limit key for index-controlled partitioning to rebalance data among the partitions in a partitioned table space; see "Rebalancing data in partitioned table spaces" on page 93.

For other changes, you must drop and re-create the index as described in "Dropping and redefining an index" on page 94.

When you add a new column to an index, change how varying-length columns are stored in the index, or change the data type of a column in the index, DB2 creates a new version of the index, as described in "Index versions" on page 94.

The ALTER INDEX statement can be embedded in an application program or issued interactively. For details on the ALTER INDEX statement, see Chapter 5 of *DB2 SQL Reference*.

Adding a new column to an index

When you use ALTER INDEX to add a new column to an existing index, the new column becomes the rightmost column of the index key. If the column that is being added to the index is already part of the table on which the index is defined, the index is left in a REBUILD-pending (RBDP) status. However, if you add a new column to a table and to an existing index on that table within the same unit of work, the index is left in advisory REORG-pending (AREO*) status and can be used immediately for data access.

For example, assume that you have a table that was created with columns that include ACCTID, STATE, and POSTED:

```
CREATE TABLE TRANS
   (ACCTID ...,
    STATE ...,
    POSTED ...,
    ... , ...)
    ...;
```

You have an existing index on the STATE column:

```
CREATE INDEX STATE_IX ON TRANS(STATE);
```

To add a ZIPCODE column to the table and the index, issue the following statements:

```
ALTER TABLE TRANS ADD COLUMN ZIPCODE CHAR(5);
ALTER INDEX STATE_IX ADD COLUMN (ZIPCODE);
COMMIT;
```

Because the ALTER TABLE and ALTER INDEX statements are executed within the same unit of work, DB2 can use the new index with the key STATE, ZIPCODE immediately for data access.

Restriction: You cannot add a column to an index that enforces a primary key, unique key, or referential constraint.

Altering how varying-length index columns are stored

You can use the ALTER INDEX statement to change how varying-length column values are stored in an index:

- Use the NOT PADDED clause if you do not want column values to be padded to their maximum length.

  This clause specifies that VARCHAR and VARGRAPHIC columns of an existing index are stored as varying-length columns. NOT PADDED is the default.
- Use the PADDED clause if you want column values to be padded to the maximum lengths of the columns.

  This clause specifies that VARCHAR and VARGRAPHIC columns of an existing index are stored as fixed-length columns.

The ALTER INDEX statement is successful only if the index has at least one varying-length column.

When you alter the padding attribute of an index, the index is placed into a restricted REBUILD-pending (RBDP) state. When you alter the padding attribute of a nonpartitioned secondary index (NPSI), the index is placed into a page set REBUILD-pending (PSRBD) state. In both cases, the indexes cannot be accessed until they are rebuilt from the data.

Altering the clustering index

You can use the ALTER INDEX statement to change the clustering index for a table:

- CLUSTER indicates that the index is to be used as the clustering index of the table. The change takes effect immediately. Any subsequently inserted rows use the new clustering index. Existing data remains clustered by the previous clustering index until the table space is reorganized.
- NOT CLUSTER indicates that the index is not to be used as the clustering index of the table. However, if the index was previously defined as the clustering index, it continues to be used as the clustering index until you explicitly specify CLUSTER for a different index.

Rebalancing data in partitioned table spaces

When data becomes out of balance in partitioned table spaces, performance can decrease. It is possible to improve performance by rebalancing the partitions to redistribute the data.

If an index established the partitions for the table space, you can use the ALTER INDEX statement for that index and a REORG job to shift data among the affected

partitions. The result is that data is balanced according to your specifications. You can rebalance data by changing the limit key values of all or most of the partitions. The limit key is the highest value of the index key for a partition. You roll the changes through the partitions one or more at a time, making relatively small parts of the data unavailable at any given time. For more information about rebalancing data for index-controlled partitioned table spaces by using the ALTER INDEX statement, see *The Official Introduction to DB2 UDB for z/OS*.

In addition, for index-controlled and table-controlled partitioned table spaces, you can use the REBALANCE option of the REORG TABLESPACE utility to shift data among the partitions. When you use REORG TABLESPACE with REBALANCE, DB2 automatically changes the limit key values for the partitioned table space.

Dropping and redefining an index

Dropping an index does not cause DB2 to drop any other objects. The consequence of dropping indexes is that DB2 invalidates application plans and packages that use the index and automatically rebinds them when they are next used.

If you want to drop a unique index, you must take additional steps before running a DROP INDEX statement. Any primary key, unique key, or referential constraints associated with the unique index must be dropped before you drop the unique index. However, you can drop a unique index for a unique key without dropping the unique constraint if the unique key was created before Version 8.

You must commit the DROP INDEX statement before you create any new table spaces or indexes by the same name. If an index is dropped and then an application program using that index is run (and thereby automatically rebound), that application program does not use the old index. If, at a later time, the index is re-created, and the application program is not rebound, the application program cannot take advantage of the new index.

Index versions

DB2 creates at least one index version each time you commit one of the following schema changes:

- Change the data type of a non-numeric column that is contained in one or more indexes by using the ALTER TABLE statement

  DB2 creates a new index version for each index that is affected by this operation.
- Change the length of a VARCHAR column that is contained in one or more PADDED indexes by using the ALTER TABLE statement

  DB2 creates a new index version for each index that is affected by this operation.
- Add a column to an index by using the ALTER INDEX statement

  DB2 creates one new index version, because only one index is affected by this operation.

Exceptions: DB2 does not create an index version under the following circumstances:

- When the index was created with DEFINE NO
- When you extend the length of a varying character (varchar data type) or varying graphic (vargraphic data type) column that is contained in one or more NOT PADDED indexes
- When you specify the same data type and length that a column—which is contained in one or more indexes—currently has, such that its definition does not actually change

DB2 creates only one index version if, in the same unit of work, you make multiple schema changes to columns contained in the same index. If you make these same schema changes in separate units of work, each change results in a new index version.

Reorganizing indexes: DB2 uses index versions to maximize data availability. Index versions enable DB2 to keep track of schema changes and simultaneously, provide users with access to data in altered columns that are contained in one or more indexes. When users retrieve rows from a table with an altered column, the data is displayed in the format that is described by the most recent schema definition, even though the data is not currently stored in this format. The most recent schema definition is associated with the *current* index version.

Although data availability is maximized by the use of index versions, performance might suffer because DB2 does not automatically reformat the data in the index to conform to the most recent schema definition. DB2 defers any reformatting of existing data until you reorganize the index and apply the schema changes with the REORG INDEX utility. The more ALTER statements (which affect indexes) that you commit between reorganizations, the more index versions DB2 must track, and the more performance can suffer.

Recommendation: Run the REORG INDEX utility as soon as possible after a schema change that affects an index to correct any performance degradation that might occur and to keep performance at its highest level.

Recycling index version numbers: DB2 can store up to 16 index versions, numbered sequentially from 0 to 15. (The next consecutive version number after 15 is 1. Version number 0 is never reused; it is reserved for the original version of the index.) The versions that are associated with schema changes that have not been applied yet are considered to be "in use," and the range of used versions is stored in the catalog. In-use versions can be recovered from image copies of the table space, if necessary. To determine the range of version numbers currently in use for an index, query the OLDEST_VERSION and CURRENT_VERSION columns of the SYSIBM.SYSINDEXES catalog table.

To prevent DB2 from running out of index version numbers (and to prevent subsequent ALTER statements from failing), you must recycle unused index version numbers regularly:

- For indexes defined as COPY YES, run the MODIFY RECOVERY utility.

  If all reusable version numbers (1 to 15) are currently in use, you must reorganize the index by running REORG INDEX before you can recycle the version numbers.
- For indexes defined as COPY NO, run the REORG TABLESPACE, REORG INDEX, LOAD REPLACE, or REBUILD INDEX utility.

  These utilities recycle the version numbers in the process of performing their primary functions.

Version numbers are considered to be unused if the schema changes that are associated with them have been applied and there are no image copies that contain data at those versions. For more information about managing index version numbers, see *DB2 Utility Guide and Reference*.

Altering views

In many cases, changing user requirements can be satisfied by modifying an existing view. But no ALTER VIEW statement exists; the only way to change a view is by dropping the view, committing the drop, and re-creating the view. When you drop a view, DB2 also drops the dependent views.

When you drop a view, DB2 invalidates application plans and packages that are dependent on the view and revokes the privileges of users who are authorized to use it. DB2 attempts to rebind the package or plan the next time it is executed, and you receive an error if you do not re-create the view.

To tell how much rebinding and reauthorizing is needed if you drop a view, check the catalog tables in Table 21.

Table 21. Catalog tables to check after dropping a view

Catalog table	What to check
SYSIBM.SYSPLANDEP	Application plans dependent on the view
SYSIBM.SYSPACKDEP	Packages dependent on the view
SYSIBM.SYSVIEWDEP	Views dependent on the view
SYSIBM.SYSTABAUTH	Users authorized to use the view

For more information about defining and dropping views, see Chapter 5 of *DB2 SQL Reference*.

Altering stored procedures

The ALTER PROCEDURE statement updates the description of a stored procedure. This example changes SYSPROC.MYPROC to run only in the WLM environment PARTSEC:

```
ALTER PROCEDURE SYSPROC.MYPROC
   WLM ENVIRONMENT PARTSEC;
```

If SYSPROC.MYPROC is defined with SECURITY DEFINER, the external security environment for the stored procedure uses the authorization ID of the owner of the stored procedure. This example changes the procedure to use the authorization ID of the person running it:

```
ALTER PROCEDURE SYSPROC.MYPROC
   SECURITY USER;
```

Altering user-defined functions

The ALTER FUNCTION statement updates the description of user-defined functions. Changes take effect immediately.

In this example, two functions named CENTER exist in the SMITH schema. The first function has two input parameters with INTEGER and FLOAT data types, respectively. The specific name for the first function is FOCUS1. The second function has three parameters with CHAR(25), DEC(5,2), and INTEGER data types.

Using the specific name to identify the function, change the WLM environment in which the first function runs from WLMENVNAME1 to WLMENVNAME2:

```
ALTER SPECIFIC FUNCTION SMITH.FOCUS1
   WLM ENVIRONMENT WLMENVNAME2;
```

This example changes the second function when any arguments are null:

```
ALTER FUNCTION SMITH.CENTER (CHAR(25), DEC(5,2), INTEGER)
   RETURNS ON NULL CALL;
```

Moving from index-controlled to table-controlled partitioning

This section describes how you can:

- Change an existing index-controlled partitioned table space to a table-controlled partitioned table space.

  For a description of table-controlled partitioning, see "Implementing table-controlled partitioning" on page 51.
- Implement a partitioned clustering index on a table-controlled partitioned table space so that the index clusters the data within each partition.

  This type of index is called a *data-partitioned secondary index* (DPSI). For more information about DPSIs, see "Data-partitioned secondary index (DPSI)" on page 56.

Example: Assume that you have a very large transaction table named TRANS that contains one row for each transaction. The table includes the following columns:

- ACCTID, which is the customer account ID
- POSTED, which holds the date of the transaction

The table space that contains TRANS is divided into 13 partitions, each of which contains one month of data. Two existing indexes are defined as follows:

- A partitioning index is defined on the transaction date by the following CREATE INDEX statement with a PARTITION ENDING AT clause:

  ```
  CREATE INDEX IX1 ON TRANS(POSTED)
     CLUSTER
     (PARTITION 1  ENDING AT ('01/31/2002'),
      PARTITION 2  ENDING AT ('02/28/2002'),
      ...
      PARTITION 13 ENDING AT ('01/31/2003'));
  ```

  The partitioning index is the clustering index, and the data rows in the table are in order by the transaction date. The partitioning index controls the partitioning of the data in the table space.
- A nonpartitioning index is defined on the customer account ID:

  ```
  CREATE INDEX IX2 ON TRANS(ACCTID);
  ```

DB2 usually accesses the transaction table through the customer account ID by using the nonpartitioning index IX2. The partitioning index IX1 is not used for data access and is wasting space. In addition, you have a critical requirement for availability on the table, and you want to be able to run an online REORG job at the partition level with minimal disruption to data availability.

To save space and to facilitate reorganization of the table space, you can drop the partitioning index IX1, and you can replace the access index IX2 with a partitioned clustering index that matches the 13 data partitions in the table.

Issue the following statements:

```
DROP INDEX IX1;
CREATE INDEX IX3 ON TRANS(ACCTID)
   PARTITIONED CLUSTER;
```

```
COMMIT;

DROP INDEX IX2;
COMMIT;
```

What happens:

- When you drop the partitioning index IX1, DB2 converts the table space from index-controlled partitioning to table-controlled partitioning. DB2 changes the high limit key value that was originally specified to the highest value for the key column.
- When you create the index IX3, DB2 creates a partitioned index with 13 partitions that match the 13 data partitions in the table. Each index partition contains the account numbers for the transactions during that month, and those account numbers are ordered within each partition. For example, partition 11 of the index matches the table partition that contains the transactions for November, 2002, and it contains the ordered account numbers of those transactions.
- You drop the nonpartitioning index IX2 because it has been replaced by IX3.

You can now run an online REORG at the partition level with minimal impact on availability. For example:

```
REORG TABLESPACE dbname.tsname PART 11
   SHRLEVEL CHANGE
```

Running this utility reorganizes the data for partition 11 of *dbname.tsname*. The data rows are ordered within each partition to match the ordering of the clustering index.

Recommendation:

- Drop a partitioning index if it is used only to define partitions. When you drop a partitioning index, DB2 automatically converts the associated index-controlled partitioned table space to a table-controlled partitioned table space.
- You can create a data-partitioned secondary index (DPSI) as the clustering index so that the data rows are ordered within each partition of the table space to match the ordering of the keys of the DPSI.
- Create any new tables in a partitioned table space by using the PARTITION BY clause and the PARTITION ENDING AT clause in the CREATE TABLE statement to specify the partitioning key and the limit key values.

Changing the high-level qualifier for DB2 data sets

The high-level qualifier for DB2 data sets is the catalog name of the integrated catalog facility, which is commonly called the “user catalog”. You cannot change this qualifier for DB2 data sets using the DB2 installation or migration update process. This section describes other ways to change this qualifier for both system data sets and user data sets.

These procedures do not actually move or copy data. For information about moving data, see “Moving DB2 data” on page 105.

Changing the high-level qualifier for DB2 data sets is a complex task; so, you should have experience both with DB2 and with managing user catalogs. The following tasks are described:

- “Defining a new integrated catalog alias” on page 99

- "Changing the qualifier for system data sets," which includes the DB2 catalog, directory, active and archive logs, and the BSDS
- "Changing qualifiers for other databases and user data sets" on page 102, which includes the work file database (DSNDB07), the default database (DSNDB04), and other DB2 databases and user databases

To concentrate on DB2-related issues, this procedure assumes that the catalog alias resides in the *same* user catalog as the one that is currently used. If the new catalog alias resides in a *different* user catalog, see *DFSMS/MVS: Access Method Services for the Integrated Catalog* for information about planning such a move.

If the data sets are managed by the Storage Management Subsystem (SMS), make sure that automatic class selection routines are in place for the new data set name.

Defining a new integrated catalog alias

This step can be done at any time before changing the high-level qualifier for system or user data sets.

Set up the new high-level qualifier as an alias to a current integrated catalog, using the following access method services command:

```
DEFINE ALIAS (NAME (newcat) RELATE (usercat) CATALOG (master-cat))
```

See *DFSMS/MVS: Access Method Services for the Integrated Catalog* for more information.

Changing the qualifier for system data sets

In this task, you stop DB2, change the high-level qualifier in the system parameter load module (possibly DSNZPARM), and establish a new xxxxMSTR cataloged procedure before restarting DB2. These steps must be done in sequence.

Step 1: Change the load module to reflect the new qualifier

To change the system parameter load module to specify the new qualifier for new archive data sets and the DB2 catalog and directory data sets, you must use the installation process.

1. Run the installation CLIST, and specify INSTALL TYPE=INSTALL and DATA SHARING FUNCTION=NONE.
2. Enter new values for the fields shown in Table 22.

Table 22. CLIST panels and fields to change to reflect new qualifier

Panel name	Field name	Comments
DSNTIPA1	INSTALL TYPE	Specify INSTALL. Do *not* specify a new default prefix for the input data sets listed on this panel.
DSNTIPA1	OUTPUT MEMBER NAME	
DSNTIPA2	CATALOG ALIAS	
DSNTIPH	COPY 1 NAME and COPY 2 NAME	These are the bootstrap data set names.
DSNTIPH	COPY 1 PREFIX and COPY 2 PREFIX	These fields appear for both active and archive log prefixes.

Table 22. CLIST panels and fields to change to reflect new qualifier (continued)

Panel name	Field name	Comments
DSNTIPT	SAMPLE LIBRARY	This field allows you to specify a field name for edited output of the installation CLIST. Avoid overlaying existing data sets by changing the middle node, NEW, to something else. The only members you use in this procedure are xxxxMSTR and DSNTIJUZ in the sample library.
DSNTIPO	PARAMETER MODULE	Change this value only if you want to preserve the existing member through the CLIST.

The output from the CLIST is a new set of tailored JCL with new cataloged procedures and a DSNTIJUZ job, which produces a new member.

3. Run the first two job steps of DSNTIJUZ to update the subsystem parameter load module. Unless you have specified a new name for the load module, make sure the output load module does not go to the SDSNEXIT or SDSNLOAD library used by the active DB2 subsystem.

   If you are changing the subsystem ID in addition to the system data set name qualifier, you should also run job steps DSNTIZP and DSNTIZQ to update the DSNHDECP module (zparm parameter SSID). Make sure that the updated DSNHDECP module does not go to the SDSNEXIT or SDSNLOAD library used by the active DB2 subsystem. Use caution when changing the subsystem ID. For more information, see the heading "MVS PARMLIB updates panel: DSNTIPM" for the discussion of panel DSNTIPM for PARMLIB members where the subsystem ID has to be changed.

   Do not run the remaining steps of DSNTIJUZ.

Step 2: Stop DB2 with no outstanding activity

In this step, make sure the subsystem does not have any outstanding activity, such as outstanding units of recovery or pending writes. This ensures that DB2 does not need to access the data sets on restart through the log, which contains the *old* data set qualifiers.

1. Enter the following command:

   ```
   -STOP DB2 MODE(QUIESCE)
   ```

   This command allows DB2 to complete processing currently executing programs.

2. Enter the following command:

   ```
   -START DB2 ACCESS(MAINT)
   ```

3. Use the following commands to make sure the subsystem is in a consistent state. See Part 4, "Operation and recovery," on page 311 and Chapter 2 of *DB2 Command Reference* for more information about these commands.

   -DISPLAY THREAD(*) TYPE(*)
   -DISPLAY UTILITY (*)
   -TERM UTILITY(*)
   -DISPLAY DATABASE(*) RESTRICT
   -DISPLAY DATABASE(*) SPACENAM(*) RESTRICT
   -RECOVER INDOUBT

   Correct any problems before continuing.

4. Stop DB2, using the following command:

   ```
   -STOP DB2 MODE(QUIESCE)
   ```

5. Run the print log map utility (DSNJU004) to identify the current active log data set and the last checkpoint RBA. For information about the print log map utility, see Part 3 of *DB2 Utility Guide and Reference*.
6. Run DSN1LOGP with the SUMMARY (YES) option, using the last checkpoint RBA from the output of the print log map utility you ran in the previous step. For information about DSN1LOGP, see Part 3 of *DB2 Utility Guide and Reference*.

   The report headed DSN1157I RESTART SUMMARY identifies active units of recovery or pending writes. If either situation exists, do not attempt to continue. Start DB2 with ACCESS(MAINT), use the necessary commands to correct the problem, and repeat steps 4 through 6 until all activity is complete.

Step 3: Rename system data sets with the new qualifier

All of the following steps assume that the new qualifier and the old qualifier reside in the same user catalog. Access method services does not allow ALTER where the new name does not match the existing catalog structure for an SMS-managed VSAM data set. If the data set is not managed by SMS, the rename succeeds, but DB2 cannot allocate it as described in *DFSMS/MVS: Access Method Services for the Integrated Catalog*.

DB2 table spaces are defined as linear data sets with DSNDBC as the second node of the name for the cluster and DSNDBD for the data component (as described in "Requirements for your own data sets" on page 38). The examples shown here assume the normal defaults for DB2 and VSAM data set names. Use access method services statements with a generic name (*) to simplify the process. Access method services allows only one generic name per data set name string.

1. Using IDCAMS, change the names of the catalog and directory table spaces. Also, be sure to specify the instance qualifier of your data set, *y*, which can be either I or J:

```
ALTER oldcat.DSNDBC.DSNDB01.*.y0001.A001  -
   NEWNAME (newcat.DSNDBC.DSNDB01.*.y0001.A001)
ALTER oldcat.DSNDBD.DSNDB01.*.y0001.A001  -
   NEWNAME (newcat.DSNDBD.DSNDB01.*.y0001.A001)
ALTER oldcat.DSNDBC.DSNDB06.*.y0001.A001  -
   NEWNAME (newcat.DSNDBC.DSNDB06.*.y0001.A001)
ALTER oldcat.DSNDBD.DSNDB06.*.y0001.A001  -
   NEWNAME (newcat.DSNDBD.DSNDB06.*.y0001.A001)
```

2. Using IDCAMS, change the active log names. Active log data sets are named *oldcat*.LOGCOPY1.COPY01 for the cluster component and *oldcat*.LOGCOPY1.COPY01.DATA for the data component.

```
ALTER oldcat.LOGCOPY1.*  -
   NEWNAME (newcat.LOGCOPY1.*)
ALTER oldcat.LOGCOPY1.*.DATA  -
   NEWNAME (newcat.LOGCOPY1.*.DATA)
ALTER oldcat.LOGCOPY2.*  -
   NEWNAME (newcat.LOGCOPY2.*)
ALTER oldcat.LOGCOPY2.*.DATA  -
   NEWNAME (newcat.LOGCOPY2.*.DATA)
```

3. Using IDCAMS, change the BSDS names.

```
ALTER oldcat.BSDS01  -
   NEWNAME (newcat.BSDS01)
ALTER oldcat.BSDS01.*  -
   NEWNAME (newcat.BSDS01.*)
ALTER oldcat.BSDS02  -
   NEWNAME (newcat.BSDS02)
ALTER oldcat.BSDS02.*  -
   NEWNAME (newcat.BSDS02.*)
```

Step 4: Update the BSDS with the new qualifier

Update the first BSDS with the new alias and correct data set names for the active logs. This procedure does not attempt to change the names of existing archive log data sets. If these catalog entries or data sets will not be available in the future, copy all the table spaces in the DB2 subsystem to establish a new recovery point. You can optionally delete the entries from the BSDS. If you do not delete the entries, they will gradually be replaced by newer entries.

1. Run the change log inventory utility (DSNJU003).

   Use the new qualifier for the BSDS because it has now been renamed. The following example illustrates the control statements required for three logs and dual copy is specified for the logs. This is only an example; the number of logs can vary and dual copy is an option. The starting and ending log RBAs are from the print log map report.

   ```
   NEWCAT VSAMCAT=newcat
   DELETE DSNAME=oldcat.LOGCOPY1.DS01
   DELETE DSNAME=oldcat.LOGCOPY1.DS02
   DELETE DSNAME=oldcat.LOGCOPY1.DS03
   DELETE DSNAME=oldcat.LOGCOPY2.DS01
   DELETE DSNAME=oldcat.LOGCOPY2.DS02
   DELETE DSNAME=oldcat.LOGCOPY2.DS03
   NEWLOG DSNAME=newcat.LOGCOPY1.DS01,COPY1,STARTRBA=strtrba,ENDRBA=endrba
   NEWLOG DSNAME=newcat.LOGCOPY1.DS02,COPY1,STARTRBA=strtrba,ENDRBA=endrba
   NEWLOG DSNAME=newcat.LOGCOPY1.DS03,COPY1,STARTRBA=strtrba,ENDRBA=endrba
   NEWLOG DSNAME=newcat.LOGCOPY2.DS01,COPY2,STARTRBA=strtrba,ENDRBA=endrba
   NEWLOG DSNAME=newcat.LOGCOPY2.DS02,COPY2,STARTRBA=strtrba,ENDRBA=endrba
   NEWLOG DSNAME=newcat.LOGCOPY2.DS03,COPY2,STARTRBA=strtrba,ENDRBA=endrba
   ```

   During startup, DB2 compares the *newcat* value with the value in the system parameter load module, and they must be the same.
2. Using the IDCAMS REPRO command, replace the contents of BSDS2 with the contents of BSDS01.
3. Run the print log map utility (DSNJU004) to verify your changes to the BSDS.
4. At a convenient time, change the DD statements for the BSDS in any of your offline utilities to use the new qualifier.

Step 5: Establish a new xxxxMSTR cataloged procedure

Before you start DB2, follow these steps:

1. Update xxxxMSTR in SYS1.PROCLIB with the new BSDS data set names.
2. Copy the new system parameter load module to the active SDSNEXIT/SDSNLOAD library.

Step 6: Start DB2 with the new xxxxMSTR and load module

Use the START DB2 command with the new load module name as shown here:

```
-START DB2 PARM(new_name)
```

If you stopped DSNDB01 or DSNDB06 in “Step 2: Stop DB2 with no outstanding activity” on page 100, you must explicitly start them in this step.

Changing qualifiers for other databases and user data sets

This step changes qualifiers for DB2 databases other than the catalog and directory. DSNDB07 is a system database but contains no permanent data, and can be deleted and redefined with the new qualifier. If you are changing its qualifier, do that before the rest of the user databases.

Change the databases in the following list that apply to your environment:

- DSNDB07 (work file database)

- DSNDB04 (default database)
- DSNDDF (communications database)
- DSNRLST (resource limit facility database)
- DSNRGFDB (the database for data definition control)
- Any other application databases that use the old high-level qualifier

At this point, the DB2 catalog tables SYSSTOGROUP, SYSTABLEPART, and SYSINDEXPART contain information about the old integrated user catalog alias. To update those tables with the new alias, you must use the following procedures. Until you do so, the underlying resources are not available. The following procedures are described separately:
- "Changing your work database to use the new high-level qualifier"
- "Changing user-managed objects to use the new qualifier" on page 104
- "Changing DB2-managed objects to use the new qualifier" on page 104

Table spaces and indexes that span more than one data set require special procedures. Partitioned table spaces can have different partitions allocated to different DB2 storage groups. Nonpartitioned table spaces or indexes only have the additional data sets to rename (those with the lowest level name of A002, A003, and so on).

Changing your work database to use the new high-level qualifier

You can use one of two methods to change the high-level qualifier for your work database or possibly DSNDB07. Which method you use depends on if you have a new installation or a migrated installation.

New installation:

1. Reallocate the database using the installation job DSNTIJTM from *prefix*.SDSNSAMP
2. Modify your existing job. Change the job to remove the BIND step for DSNTIAD and rename the data set names in the DSNTTMP step to your new names, making sure you include your current allocations.

Migrated installations: Migrated installations do not have a usable DSNTIJTM, because the IDCAMS allocation step is missing. For migrated installations, you must:

1. Stop the database, using the following command (for a database named DSNDB07):
   ```
   -STOP DATABASE (DSNDB07)
   ```
2. Drop the database, using the following SQL statement:
   ```
   DROP DATABASE DSNDB07;
   ```
3. Re-create the database, using the following SQL statement:
   ```
   CREATE DATABASE DSNDB07;
   ```
4. Define the clusters, using the following access method services commands. Also, be sure to specify the instance qualifier of your data set, *y*, which can be either I or J:
   ```
   ALTER oldcat.DSNDBC.DSNDB07.DSN4K01.y0001.A001
      NEWNAME newcat.DSNDBC.DSNDB07.DSN4K01.y0001.A001
   ALTER oldcat.DSNDBC.DSNDB07.DSN32K01.y0001.A001
      NEWNAME newcat.DSNDBC.DSNDB07.DSN32K01.y0001.A001
   ```
   Repeat the preceding statements (with the appropriate table space name) for as many table spaces as you use.
5. Create the table spaces in DSNDB07.

```
CREATE TABLESPACE DSN4K01
  IN DSNDB07
  BUFFERPOOL BP0
  CLOSE NO
  USING VCAT DSNC810;

CREATE TABLESPACE DSN32K01
  IN DSNDB07
  BUFFERPOOL BP32K
  CLOSE NO
  USING VCAT DSNC810;
```

6. Start the database, using the following command:

   ```
   -START DATABASE (DSNDB07)
   ```

Changing user-managed objects to use the new qualifier

1. Stop the table spaces and index spaces, using the following command:

   ```
   -STOP DATABASE(dbname) SPACENAM(*)
   ```

2. Use the following SQL ALTER TABLESPACE and ALTER INDEX statements with the USING clause to specify the new qualifier:

   ```
   ALTER TABLESPACE dbname.tsname
     USING VCAT newcat;

   ALTER INDEX creator.index-name
     USING VCAT newcat;
   ```

   Repeat this step for all the objects in the database.

3. Using IDCAMS, rename the data sets to the new qualifier. Also, be sure to specify the instance qualifier of your data set, *y*, which can be either I or J:

   ```
   ALTER oldcat.DSNDBC.dbname.*.y0001.A001  -
      NEWNAME newcat.DSNDBC.dbname.*.y0001.A001
   ALTER oldcat.DSNDBD.dbname.*.y0001.A001  -
      NEWNAME newcat.DSNDBD.dbname.*.y0001.A001
   ```

4. Start the table spaces and index spaces, using the following command:

   ```
   -START DATABASE(dbname) SPACENAM(*)
   ```

5. Verify the success of the procedure by entering the following command:

   ```
   -DISPLAY DATABASE(dbname)
   ```

6. Using SQL, verify that you can access the data.

Renaming the data sets can be done while DB2 is down. They are included here because the names must be generated for each database, table space, and index space that is to change.

Changing DB2-managed objects to use the new qualifier

Use this procedure when you want to keep the existing DB2 storage group, changing only the high-level qualifier. If you want to move the data to a new DB2 storage group, see “Moving a DB2 data set” on page 108.

1. Remove all table spaces and index spaces from the storage group by converting the data sets temporarily to user-managed data sets.

   a. Stop each database that has data sets you are going to convert, using the following command:

      ```
      -STOP DATABASE(dbname) SPACENAM(*)
      ```

      Note: Some databases (such as DSNRTSDB) will have to be explicitly stopped to allow any alterations. For these, use the following command:

      ```
      -STOP DATABASE(dbname)
      ```

b. Convert to user-managed data sets with the USING VCAT clause of the SQL ALTER TABLESPACE and ALTER INDEX statements, as shown in the following statements. Use the new catalog name for VCAT.

```
ALTER TABLESPACE dbname.tsname
    USING VCAT newcat;
```

```
ALTER INDEX creator.index-name
    USING VCAT newcat;
```

2. Drop the storage group, using the following statement:

```
DROP STOGROUP stogroup-name;
```

The DROP succeeds only if all the objects that referenced this STOGROUP are dropped or converted to user-managed (USING VCAT clause).

3. Re-create the storage group using the correct volumes and the new alias, using the following statement:

```
CREATE STOGROUP stogroup-name
   VOLUMES (VOL1,VOL2)
   VCAT newcat;
```

4. Using IDCAMS, rename the data sets for the index spaces and table spaces to use the new high-level qualifier. Also, be sure to specify the instance qualifier of your data set, *y*, which can be either I or J:

```
ALTER oldcat.DSNDBC.dbname.*.y0001.A001  -
   NEWNAME newcat.DSNDBC.dbname.*.y0001.A001
ALTER oldcat.DSNDBD.dbname.*.y0001.A001  -
   NEWNAME newcat.DSNDBD.dbname.*.y0001.A001
```

If your table space or index space spans more than one data set, be sure to rename those data sets also.

5. Convert the data sets back to DB2-managed data sets by using the new DB2 storage group. Use the following SQL ALTER TABLESPACE and ALTER INDEX statements:

```
ALTER TABLESPACE dbname.tsname
  USING STOGROUP stogroup-name
  PRIQTY priqty
  SECQTY secqty;
```

```
ALTER INDEX creator.index-name
  USING STOGROUP stogroup-name
  PRIQTY priqty
  SECQTY secqty;
```

If you specify USING STOGROUP without specifying the PRIQTY and SECQTY clauses, DB2 uses the default values. For more information about USING STOGROUP, see *DB2 SQL Reference*.

6. Start each database, using the following command:

```
-START DATABASE(dbname) SPACENAM(*)
```

7. Verify the success of the procedure by entering the following command:

```
-DISPLAY DATABASE(dbname)
```

8. Using SQL, verify that you can access the data.

Moving DB2 data

This section discusses the following topics:

- "Tools for moving DB2 data" on page 106 introduces some of the tools available to make moving DB2 data easier.
- "Moving a DB2 data set" on page 108 describes moving a data set from one volume to another.

- "Copying a relational database" on page 109 describes copying a user-managed relational database, with its object definitions and its data, from one DB2 subsystem to another, on the same or different z/OS system.
- "Copying an entire DB2 subsystem" on page 109 describes copying a DB2 subsystem from one z/OS system to another. Copying a subsystem includes these items:
  - All the user data and object definitions
  - The DB2 system data sets:
    - The log
    - The bootstrap data set
    - Image copy data sets
    - The DB2 catalog
    - The integrated catalog that records all the DB2 data sets

Tools for moving DB2 data

Important: Before copying any DB2 data, resolve any data that is in an inconsistent state. Use the DISPLAY DATABASE command to determine whether any inconsistent state exists, and the RECOVER INDOUBT command or the RECOVER utility to resolve the inconsistency. The copying process generally loses all traces of an inconsistency except the problems that result.

Although DB2 data sets are created using VSAM access method services, they are specially formatted for DB2 and cannot be processed by services that use VSAM record processing. They can be processed by VSAM utilities that use control-interval (CI) processing and, if they are linear data sets (LDSs), also by utilities that recognize the LDS type.

Furthermore, copying the data might not be enough. Some operations require copying DB2 object definitions. And when copying from one subsystem to another, you must consider internal values that appear in the DB2 catalog and the log, for example, the DB2 object identifiers (OBIDs) and log relative byte addresses (RBAs).

Fortunately, several tools exist that simplify the operations:

- The REORG and LOAD utilities move data sets from one disk device type to another within the same DB2 subsystem. For instructions on using LOAD and REORG, see Part 2 of *DB2 Utility Guide and Reference*.

  The INCURSOR option of the LOAD utility allows you to specify a cursor to select data from another DB2 table or tables, which can be on a remote DB2 system. Use the EXEC SQL utility control statement to declare the cursor before executing the LOAD utility. This option uses the DB2 UDB family cross-loader function.
- The COPY and RECOVER utilities allow you to recover an image copy of a DB2 table space or index space onto another disk device within the same subsystem. For instructions on using COPY and RECOVER, see Part 2 of *DB2 Utility Guide and Reference*.
- The UNLOAD or REORG UNLOAD EXTERNAL utility unloads a DB2 table into a sequential file and generates statements to allow the LOAD utility to load it elsewhere. For instructions on using UNLOAD or REORG UNLOAD EXTERNAL, see *DB2 Utility Guide and Reference*.
- The DSN1COPY utility copies the data set for a table space or index space to another data set. It can also translate the object identifiers and reset the log RBAs in the target data set. When you use the OBIDXLAT option of DSN1COPY to move objects from one system to another, use REPAIR VERSIONS to update

the version information in the catalog and directory for the target table space or index. For instructions, see Part 3 of *DB2 Utility Guide and Reference*.

You might also want to use the following tools to move DB2 data:

- The DB2 DataPropagator is a licensed program that can extract data from DB2 tables, DL/I databases, VSAM files, and sequential files. For instructions, see "Loading data from DL/I" on page 65.
- DFSMS, which contains the following functional components:
  - Data Set Services (DFSMSdss)

    Use DFSMSdss to copy data between disk devices. For instructions, see *Data Facility Data Set Services: User's Guide and Reference*. You can use online panels to control this, through the Interactive Storage Management Facility (ISMF) that is available with DFSMS; for instructions, refer to *z/OS DFSMSdfp Storage Administration Reference*.
  - Data Facility Product (DFSMSdfp)

    This is a prerequisite for DB2. You can use access method services EXPORT and IMPORT commands with DB2 data sets when control interval processing (CIMODE) is used. For instructions on EXPORT and IMPORT, see *DFSMS/MVS: Access Method Services for the Integrated Catalog*.
  - Hierarchical Storage Manager (DFSMShsm)

    With the MIGRATE, HMIGRATE, or HRECALL commands, which can specify specific data set names, you can move data sets from one disk device type to another within the same DB2 subsystem. Do not migrate the DB2 directory, DB2 catalog, and the work file database (DSNDB07). Do not migrate any data sets that are in use frequently, such as the bootstrap data set and the active log. With the MIGRATE VOLUME command, you can move an entire disk volume from one device type to another. The program can be controlled using online panels, through the Interactive Storage Management Facility (ISMF). For instructions, see *z/OS DFSMShsm Managing Your Own Data*.

Table 23 shows which tools are applicable to which operations.

Table 23. Tools applicable to data-moving operations

Tool	Moving a data set	Copying a data base	Copying an entire subsystem
REORG and LOAD	Yes	Yes	No
COPY and RECOVER	Yes	No	No
DSNTIAUL	Yes	Yes	No
DSN1COPY	Yes	Yes	No
DataRefresher or DXT	Yes	Yes	No
DFSMSdss	Yes	No	Yes
DFSMSdfp	Yes	No	Yes
DFSMShsm	Yes	No	No

Some of the listed tools rebuild the table space and index space data sets, and they therefore generally require longer to execute than the tools that merely copy them. The tools that rebuild are REORG and LOAD, RECOVER and REBUILD, DSNTIAUL, and DataRefresher. The tools that merely copy data sets are DSN1COPY, DFSMSdss, DFSMSdfp EXPORT and IMPORT, and DFSMShsm.

DSN1COPY is fairly efficient in use, but somewhat complex to set up. It requires a separate job step to allocate the target data sets, one job step for each data set to copy the data, and a step to delete or rename the source data sets. DFSMSdss, DFSMSdfp, and DFSMShsm all simplify the job setup significantly.

Although less efficient in execution, RECOVER is easy to set up if image copies and recover jobs already exist. You might only need to redefine the data sets involved and recover the objects as usual.

Moving a DB2 data set

Moving DB2 data is accomplished by RECOVER, REORG, or DSN1COPY, or by the use of non-DB2 facilities, such as DFSMSdss. Both the DB2 utilities and the non-DB2 tools can be used while DB2 is running, but the space to be moved should be stopped to prevent users from accessing it.

The following procedures differ mainly in that the first one assumes you do not want to reorganize or recover the data. Generally, this means that the first procedure is faster. In all cases, make sure that there is enough space on the target volume to accommodate the data set.

If you use storage groups, then you can change the storage group definition to include the new volumes, as described in "Altering DB2 storage groups" on page 67.

Moving data without REORG or RECOVER:

1. Stop the database.
2. Move the data, using DSN1COPY or a non-DB2 facility.
3. Issue the ALTER INDEX or ALTER TABLESPACE statement to use the new integrated catalog facility catalog name or DB2 storage group name.
4. Start the database.

Moving DB2-managed data with REORG, RECOVER, or REBUILD: With this procedure you create a storage group (possibly using a new catalog alias) and move the data to that new storage group.

1. Create a new storage group using the correct volumes and the new alias, as shown in the following statement:

```
CREATE STOGROUP stogroup-name
   VOLUMES (VOL1,VOL2)
   VCAT (newcat);
```

2. Prevent access to the data sets you are going to move, by entering the following command:

```
-STOP DATABASE(dbname) SPACENAM(*)
```

3. Enter the ALTER TABLESPACE and ALTER INDEX SQL statements to use the new storage group name, as shown in the following statements:

```
ALTER TABLESPACE dbname.tsname
  USING STOGROUP stogroup-name;

ALTER INDEX creator.index-name
  USING STOGROUP stogroup-name;
```

4. Using IDCAMS, rename the data sets for the index spaces and table spaces to use the new high-level qualifier. Also, be sure to specify the instance qualifier of your data set, *y*, which can be either I or J. If you have run REORG with SHRLEVEL CHANGE or SHRLEVEL REFERENCE on any table spaces or index spaces, the fifth-level qualifier might be J0001.

```
ALTER oldcat.DSNDBC.dbname.*.y0001.A001  -
   NEWNAME newcat.DSNDBC.dbname.*.y0001.A001
ALTER oldcat.DSNDBD.dbname.*.y0001.A001  -
   NEWNAME newcat.DSNDBD.dbname.*.y0001.A001
```

5. Start the database for utility processing only, using the following command:

   ```
   -START DATABASE(dbname) SPACENAM(*) ACCESS(UT)
   ```

6. Use the REORG or the RECOVER utility on the table space or index space, or use the REBUILD utility on the index space.
7. Start the database, using the following command:

   ```
   -START DATABASE(dbname) SPACENAM(*)
   ```

Copying a relational database

This operation involves not only copying data, but finding or generating, and executing, SQL statements to create storage groups, databases, table spaces, tables, indexes, views, synonyms, and aliases.

As with the other operations, DSN1COPY is likely to execute faster than the other applicable tools. It copies directly from one data set to another, while the other tools extract input for LOAD, which then loads table spaces and builds indexes. But again, DSN1COPY is more difficult to set up. In particular, you must know the internal DB2 object identifiers, which other tools translate automatically.

Copying an entire DB2 subsystem

This operation involves copying an entire DB2 subsystem from one z/OS system to another. (Although you can have two DB2 subsystems on the same z/OS system, one cannot be a copy of the other.)

Only two of the tools listed are applicable: DFSMSdss DUMP and RESTORE, and DFSMSdfp EXPORT and IMPORT. Refer to the documentation on those programs for the most recent information about their use.

Chapter 7. Estimating disk storage for user data

This chapter provides information about how to estimate the amount of disk storage you need for your data, including:
- "Factors that affect storage"
- "Calculating the space required for a table" on page 112
- "Calculating the space required for a dictionary" on page 116
- "Calculating the space required for an index" on page 117

Recommendation: Use DB2 Estimator to calculate space estimates for tables, indexes, and factors discussed in this chapter.

Factors that affect storage

The amount of disk space you need for your data is not just the number of bytes of data; the true number is some multiple of that. That is,

space required = *M* × (number of bytes of data)

The multiplier M depends on your circumstances. It includes factors that are common to all data sets on disk, as well as others that are peculiar to DB2. It can vary significantly, from a low of about 1.25, to 4.0 or more. For a first approximation, set M=2, and skip to "Calculating the space required for a table" on page 112.

For more accuracy, calculate M as the product of the following factors:
- Record overhead
- Free space
- Unusable space
- Data set excess
- Indexes

Record overhead: Allows for eight bytes of record header and control data, plus space wasted for records that do not fit exactly into a DB2 page. For the second consideration, see "Choosing a page size" on page 45. The factor can range from about 1.01 (for a careful space-saving design) to as great as 4.0. A typical value is about 1.10.

Free space: Allows for space intentionally left empty to allow for inserts and updates. You can specify this factor on the CREATE TABLESPACE statement; see "Specifying free space on pages" on page 652 for more information. The factor can range from 1.0 (for no free space) to 200 (99% of each page used left free, and a free page following each used page). With default values, the factor is about 1.05.

Unusable space: Track lengths in excess of the nearest multiple of page lengths. Table 24 shows the track size, number of pages per track, and the value of the unusable-space factor for several different device types.

Table 24. Unusable space factor by device type

Device type	Track size	Pages per track	Factor value
3380	47476	10	1.16
3390	56664	12	1.15

© Copyright IBM Corp. 1982, 2008

Data set excess: Allows for unused space within allocated data sets, occurring as unused tracks or part of a track at the end of any data set. The amount of unused space depends upon the volatility of the data, the amount of space management done, and the size of the data set. Generally, large data sets can be managed more closely, and those that do not change in size are easier to manage. The factor can range without limit above 1.02. A typical value is 1.10.

Indexes: Allows for storage for indexes to data. For data with no indexes, the factor is 1.0. For a single index on a short column, the factor is 1.01. If every column is indexed, the factor can be greater than 2.0. A typical value is 1.20. For further discussion of the factor, see "Calculating the space required for an index" on page 117.

Table 25 shows calculations of the multiplier M for three different database designs:

- The *tight* design is carefully chosen to save space and allows only one index on a single, short field.
- The *loose* design allows a large value for every factor, but still well short of the maximum. Free space adds 30% to the estimate, and indexes add 40%.
- The *medium* design has values between the other two. You might want to use these values in an early stage of database design.

In each design, the device type is assumed to be a 3390. Therefore, the unusable-space factor is 1.15. M is always the product of the five factors.

Table 25. Calculations for different database designs

Factor	Tight design	Medium design	Loose design
Record overhead ×	1.02	1.10	1.30
Free space ×	1.00	1.05	1.30
Unusable space ×	1.15	1.15	1.15
Data set excess ×	1.02	1.10	1.30
Indexes =	1.02	1.20	1.40
Multiplier M	1.22	1.75	3.54

In addition to the space for your data, external storage devices are required for:

- Image copies of data sets, which can be on tape
- System libraries, system databases, and the system log
- Temporary work files for utility and sort jobs

A rough estimate of the additional external storage needed is three times the amount calculated for disk storage.

Calculating the space required for a table

This section provides information about how to calculate the space required for a table, including:

- "Calculating record lengths and pages" on page 113
- "Saving space with data compression" on page 114
- "Estimating storage for LOBs" on page 114
- "Estimating storage when using the LOAD utility" on page 114

Space allocation parameters are specified in kilobytes (KB).

Calculating record lengths and pages

An important consideration in the design of a table is the record size. In DB2, a record is the storage representation of a row. Records are stored within pages that are 4 KB, 8 KB, 16 KB, or 32 KB. Generally, you cannot create a table in which the maximum record size is greater than the page size.

Also consider:
- Normalizing your entities
- Using larger page sizes
- Using LOB data types if a single column in a table is greater than 32 K

In addition to the bytes of actual data in the row (not including LOB data, which is not stored in the base row or included in the total length of the row), each record has:
- A six-byte prefix
- One additional byte for each column that can contain null values
- Two additional bytes for each varying-length column or ROWID column
- Six bytes of descriptive information in the base table for each LOB column

The sum of each column's length is the *record length*, which is the length of data that is physically stored in the table. You can retrieve the value of the AVGROWLEN column in the SYSIBM.SYSTABLES catalog table to determine the average length of rows within a table. The *logical record length* can be longer, for example, if the table contains LOBs.

Every data page has:
- A 22-byte header
- A 2-byte directory entry for each record stored in the page

To simplify the calculation of record and page length, consider the directory entry as part of the record. Then, every record has a fixed overhead of 8 bytes, and the space available to store records in a 4 KB page is 4074 bytes. Achieving that maximum in practice is not always simple. For example, if you are using the default values, the LOAD utility leaves approximately 5 percent of a page as free space when loading more than one record per page. Therefore, if two records are to fit in a page, each record cannot be longer than 1934 bytes (approximately $0.95 \times 4074 \times 0.5$).

Furthermore, the page size of the table space in which the table is defined limits the record length. If the table space is 4 KB, the record length of each record cannot be greater than 4056 bytes. Because of the 8-byte overhead for each record, the sum of column lengths cannot be greater than 4048 bytes (4056 minus the 8-byte overhead for a record).

DB2 provides three larger page sizes to allow for longer records. You can improve performance by using pages for record lengths that best suit your needs. For details on selecting an appropriate page size, see "Choosing a page size" on page 45.

As shown in Table 26 on page 114, the maximum record size for each page size depends on the size of the table space and on whether you specified the EDITPROC clause.

Table 26. Maximum record size (in bytes)

EDITPROC	4-KB page	8-KB page	16-KB page	32-KB page
NO	4056	8138	16330	32714
YES	4046	8128	16320	32704

Creating a table using CREATE TABLE LIKE in a table space of a larger page size changes the specification of LONG VARCHAR to VARCHAR and LONG VARGRAPHIC to VARGRAPHIC. You can also use CREATE TABLE LIKE to create a table with a smaller page size in a table space if the maximum record size is within the allowable record size of the new table space.

Saving space with data compression

You can reduce the space required for a table by using data compression if your system meets the requirements. To find out how much space you can save by compressing your data, run the DSN1COMP utility on your data sets. Message DSN1940I of DSN1COMP reports an estimate of the percentage of kilobytes that would be saved by using data compression. You cannot compress data in a LOB table space. See Part 3 of *DB2 Utility Guide and Reference* for more information about the DSN1COMP utility.

The disk saved by data compression is countered by the disk required for a *dictionary*. Every compressed table space or partition requires a dictionary. See "Calculating the space required for a dictionary" on page 116 to figure the disk requirements and the virtual storage requirements for a dictionary.

Estimating storage for LOBs

Tables with large object data types (LOBs) can store byte strings up to 2 GB. A base table can be defined with one or more LOB columns. The LOB columns are logically part of the base table but are physically stored in an auxiliary table. In place of each LOB column, there is an *indicator column*, which is a column with descriptive information about the LOB. When a base table has LOB columns, then each row of the table has a *row identifier*, which is a varying-length 17-byte field. You must consider the overhead of the indicator column and row identifiers when estimating table size. If the LOB column is NULL or has a value of zero, no space is allocated in the auxiliary table.

When estimating the storage required for LOB table spaces, begin with your estimates from other table spaces, round up to the next page size, and then multiply by 1.1. One page never contains more than one LOB. When a LOB value is deleted, the space occupied by that value remains allocated as long as any application might access that value.

An auxiliary table resides in a LOB table space. There can be only one auxiliary table in a LOB table space. An auxiliary table can store only one LOB column of a base table and there must be one and only one index on this column.

See *DB2 Installation Guide* for information about storage options for LOB values.

Estimating storage when using the LOAD utility

For a table to be loaded by the LOAD utility, the value can be estimated as follows:

- Let FLOOR be the operation of discarding the decimal portion of a real number.

- Let CEILING be the operation of rounding a real number up to the next highest integer.
- Let *number of records* be the total number of records to be loaded.
- Let *average record size* be the sum of the lengths of the fields in each record, using an average value for varying-length fields, and including the following amounts for overhead:
  - 8 bytes for the total record
  - 1 byte for each field that allows nulls
  - 2 bytes for each varying-length field

  See the CREATE TABLE statement in Chapter 5 of *DB2 SQL Reference* for information about how many bytes are required for different column types.
- Let *percsave* be the percentage of kilobytes saved by compression (as reported by the DSN1COMP utility in message DSN1940I)
- Let *compression ratio* be *percsave*/100

Then calculate as follows:

1. *Usable page size* is the page size minus a number of bytes of overhead (that is, 4 KB – 40 for 4 KB pages, 8 KB – 54 for 8 KB pages, 16 KB – 54 for 16 KB pages, or 32 KB – 54 for 32 KB pages) multiplied by (100-*p*) / 100, where *p* is the value of PCTFREE. If your average record size is less than 16, then usable page size is 255 (maximum records per page) multiplied by average record size multiplied by (100-*p*) / 100.
2. *Records per page* is MIN(MAXROWS, FLOOR(*usable page size / average record size*)), but cannot exceed 255 and cannot exceed the value you specify for MAXROWS.
3. *Pages used* is 2+CEILING(*number of records / records per page*).
4. *Total pages* is FLOOR(*pages used*× (1+*fp*) / *fp*), where *fp* is the (nonzero) value of FREEPAGE. If FREEPAGE is 0, then *total pages* is equal to *pages used*. (See "Free space" on page 111 for more information about FREEPAGE.) If you are using data compression, you need additional pages to store the dictionary. See "Calculating the space required for a dictionary" on page 116 to figure how many pages the dictionary requires.
5. Estimated number of kilobytes required for a table:
   - **If you do not use data compression**, the estimated number of kilobytes is *total pages*× page size (4 KB, 8 KB, 16 KB, or 32 KB).
   - **If you use data compression**, the estimated number of kilobytes is *total pages*× page size (4 KB, 8 KB, 16 KB, or 32 KB) × (1 - *compression ratio*).

For example, consider a table space containing a single table with the following characteristics:
- *Number of records* = 100000
- *Average record size* = 80 bytes
- Page size = 4 KB
- PCTFREE = 5 (5% of space is left free on each page)
- FREEPAGE = 20 (one page is left free for each 20 pages used)
- MAXROWS = 255

If the data is not compressed, you get the following results:
- *Usable page size* = 4056 × 0.95 = 3853 bytes
- *Records per page* = MIN(MAXROWS, FLOOR(3853 / 80)) = 48
- *Pages used* = 2 + CEILING(100000 / 48) = 2085
- *Total pages* = FLOOR(2085 × 21 / 20) = 2189
- *Estimated number of kilobytes* = 2189 × 4 = 8756

If the data is compressed, multiply the estimated number of kilobytes for an uncompressed table by (1 - *compression ratio*) for the estimated number of kilobytes required for the compressed table.

Calculating the space required for a dictionary

This section helps you calculate the disk space required by a *dictionary* and the virtual storage required in the xxxxDBM1 address space when a dictionary is read into storage from a buffer pool. A dictionary contains the information used for compressing and decompressing the data in a table space or partition, and it resides in that table space or partition. You can skip this section if you are not going to compress data. Space allocation parameters are specified in pages (either 4 KB, 8 KB, 16 KB, or 32 KB).

The following topics provide additional information:
- "Disk requirements"
- "Virtual storage requirements" on page 117

Disk requirements

This section helps you calculate the disk requirements for a dictionary associated with a compressed nonsegmented table space and for a dictionary associated with a compressed segmented table space.

Nonsegmented table space: The dictionary contains 4096 entries in most cases. This means you need to allocate an additional sixteen 4-KB pages, eight 8-KB pages, four 16-KB pages, or two 32-KB pages. Although it is possible that your dictionary can contain fewer entries, allocate enough space to accommodate a dictionary with 4096 entries. For 32-KB pages, one segment (minimum of four pages) is sufficient to contain the dictionary. Use Table 27 to determine how many 4-KB pages, 8-KB pages, 16-KB pages, or 32-KB pages to allocate for the dictionary of a compressed nonsegmented table space.

Table 27. Pages required for the dictionary of a compressed nonsegmented table space

Table space page size (KB)	Dictionary size (512 entries)	Dictionary size (1024 entries)	Dictionary size (2048 entries)	Dictionary size (4096 entries)	Dictionary size (8192 entries)
4	2	4	8	16	32
8	1	2	4	8	16
16	1	1	2	4	8
32	1	1	1	2	4

Segmented table space: The size of the dictionary depends on the size of your segments. Assuming 4096 entries is recommended. Use Table 28 to determine how many 4-KB pages to allocate for the dictionary of a compressed segmented table space.

Table 28. Pages required for the dictionary of a compressed segmented table space

Segment size (4-KB pages)	Dictionary size (512 entries)	Dictionary size (1024 entries)	Dictionary size (2048 entries)	Dictionary size (4096 entries)	Dictionary size (8192 entries)
4	4	4	8	16	32
8	8	8	8	16	32

Table 28. Pages required for the dictionary of a compressed segmented table space (continued)

Segment size (4-KB pages)	Dictionary size (512 entries)	Dictionary size (1024 entries)	Dictionary size (2048 entries)	Dictionary size (4096 entries)	Dictionary size (8192 entries)
12	12	12	12	24	36
≥16	Segment size	Segment size	Segment size	Segment size	Segment size

Virtual storage requirements

You can calculate how much storage is needed in the xxxxDBM1 address space for each dictionary with this formula:

dictionary size (number of entries) × 16 bytes

When a dictionary is read into storage from a buffer pool, the *whole* dictionary is read, and it remains there as long as the compressed table space is being accessed.

Calculating the space required for an index

This section describes the levels of index pages and helps you calculate the storage required for an index, including:

- "Levels of index pages"
- "Estimating storage from number of index pages" on page 118

Levels of index pages

Indexes can have more than one level of pages. Index pages that point directly to the data in tables are called *leaf pages* and are said to be on *level 0*. In addition to data pointers, leaf pages contain the key and record-ID (RID).

If an index has more than one leaf page, it must have at least one nonleaf page that contains entries that point to the leaf pages. If the index has more than one nonleaf page, then the nonleaf pages whose entries point to leaf pages are said to be on *level 1*. If an index has a second level of nonleaf pages whose entries point to nonleaf pages on level 1, then those nonleaf pages are said to be on *level 2*, and so on. The highest level of an index contains a single page, which DB2 creates when it first builds the index. This page is called the *root page*. The root page is a 4-KB index page. Figure 6 on page 118 shows, in schematic form, a typical index.

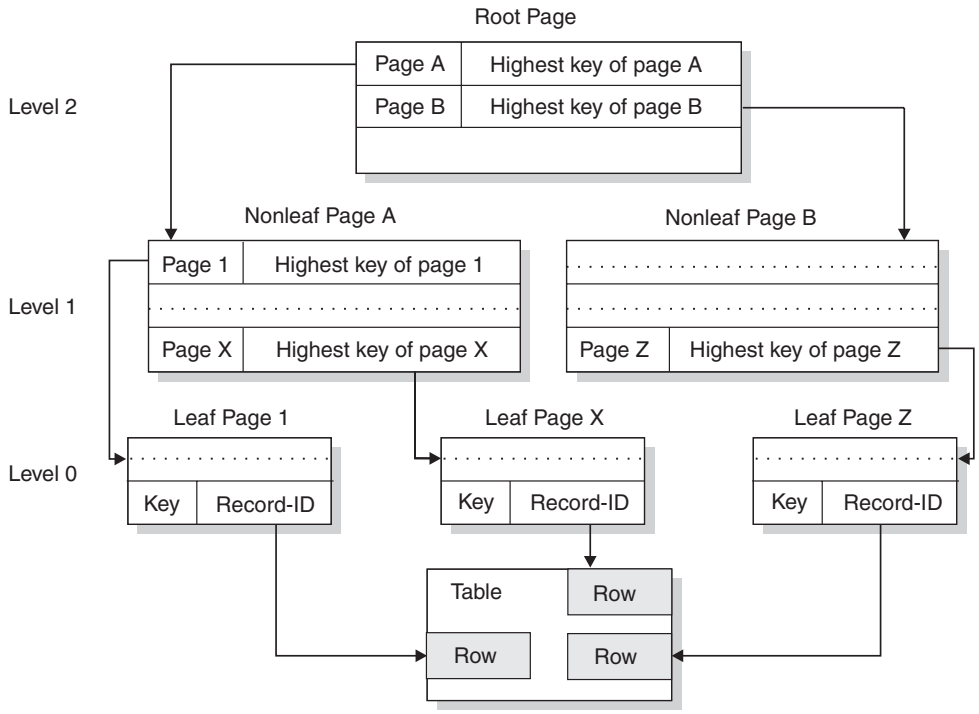


Figure 6. Sample index structure and pointers (three-level index)

If you insert data with a constantly increasing key, DB2 adds the new highest key to the top of a new page. Be aware, however, that DB2 treats nulls as the highest value. When the existing high key contains a null value in the first column that differentiates it from the new key that is inserted, the inserted nonnull index entries cannot take advantage of the *highest-value* split.

Estimating storage from number of index pages

Space allocation parameters are specified in kilobytes. For an index to be loaded by the LOAD utility, you should estimate the future storage requirements of the index.

Estimating the space requirements for DB2 objects is easier if you collect and maintain a statistical history of those objects. The accuracy of your estimates depends on the currentness of the statistical data. To ensure that the statistics history is current, use the MODIFY STATISTICS utility to delete outdated statistical data from the catalog history tables.

The storage required for an index, newly built by the LOAD utility, depends on the number of index pages at all levels. That, in turn, depends on whether the index is unique or not. The numbers of leaf pages (index pages that point directly to the data in your tables) and of nonleaf pages (index pages that contain the page number and the highest key of each page in the next-level index) are calculated separately.

An index key on an auxiliary table used for LOBs is 19 bytes and uses the same formula as other indexes. The RID value stored within the index is 5 bytes, the same as for large table spaces (defined with DSSIZE greater than or equal to 4 GB).

In general, the length of the index key is the sum of the lengths of all the columns of the key, plus the number of columns that allow nulls. The length of a

varying-length column is the maximum length if the index is padded. Otherwise, if an index is not padded, estimate the length of a varying-length column to be the average length of the column data, and add a two-byte length field to the estimate. You can retrieve the value of the AVGKEYLEN column in the SYSIBM.SYSINDEXES catalog table to determine the average length of keys within an index.

The following index calculations are intended only to help you estimate the storage required for an index. Because there is no way to predict the exact number of duplicate keys that can occur in an index, the results of these calculations are not absolute. It is possible, for example, that for a nonunique index, more index entries than the calculations indicate might fit on an index page. The calculations are divided into cases using a unique index and using a nonunique index.

In the following calculations, let:

- *k* = the length of the index key.
- *n* = the average number of data records per distinct key value of a nonunique index. For example:
  - *a* = number of data records per index
  - *b* = number of distinct key values per index
  - *n* = a / b
- *f* = the value of PCTFREE.
- *p* = the value of FREEPAGE.
- *r* = record identifier (RID) length. Let *r* = 4 for indexes on nonlarge table spaces and *r* = 5 for indexes on large spaces (defined with DSSIZE greater than or equal to 4 GB) and on auxiliary tables.
- FLOOR = the operation of discarding the decimal portion of a real number.
- CEILING = the operation of rounding a real number up to the next highest integer.
- MAX = the operation of selecting the highest integer value.

Calculate pages for a unique index: Use the following calculations to estimate the number of leaf and nonleaf pages in a unique index.

Calculate the *total leaf pages*:

1. *Space per key* ≅*k* + *r* + 3
2. *Usable space per page* ≅FLOOR((100 - *f*) × 4038 / 100)
3. *Entries per page* ≅FLOOR(*usable space per page* / *space per key*)
4. ***Total leaf pages*** ≅CEILING(*number of table rows* / *entries per page*)

Calculate the *total nonleaf pages*:

1. *Space per key* ≅*k* + 7
2. *Usable space per page* ≅FLOOR(MAX(90, (100 - *f*)) × 4046/100)
3. *Entries per page* ≅FLOOR(*usable space per page* / *space per key*)
4. *Minimum child pages* ≅MAX(2, (entries per page + 1))
5. *Level 2 pages* ≅CEILING(*total leaf pages* / *minimum child pages*)
6. *Level 3 pages* ≅CEILING(*level 2 pages* / *minimum child pages*)
7. *Level x pages* ≅CEILING(*previous level pages* / *minimum child pages*)
8. ***Total nonleaf pages*** ≅(*level 2 pages* + *level 3 pages* + ...+ *level x pages* until the number of *level x pages* = 1)

Calculate pages for a nonunique index: Use the following calculations to estimate the number of leaf and nonleaf pages for a nonunique index.

Calculate the *total leaf pages*:

1. *Space per key* ≅4 + *k* + (n × *(r+1)*)

2. *Usable space per page* ≅FLOOR((100 - *f*) × 4038 / 100)
3. *Key entries per page* ≅*n* × (*usable space per page* / *space per key*)
4. *Remaining space per page* ≅*usable space per page* - (*key entries per page* / n) ×*space per key*
5. *Data records per partial entry* ≅FLOOR((*remaining space per page* - (*k* + 4)) / 5)
6. *Partial entries per page* ≅(*n* / CEILING(*n* / *data records per partial entry*)) if *data records per partial entry* >= 1, or 0 if *data records per partial entry* < 1
7. *Entries per page* ≅MAX(1, (*key entries per page* + *partial entries per page*))
8. ***Total leaf pages*** ≅CEILING(*number of table rows* / *entries per page*)

Calculate the *total nonleaf pages*:

1. *Space per key* ≅*k* + *r* + 7
2. *Usable space per page* ≅FLOOR (MAX(90, (100- *f*)) × (4046 / 100)
3. *Entries per page* ≅FLOOR((*usable space per page* / *space per key*)
4. *Minimum child pages* ≅MAX(2, (entries per page + 1))
5. *Level 2 pages* ≅CEILING(*total leaf pages* / *minimum child pages*)
6. *Level 3 pages* ≅CEILING(*level 2 pages* / *minimum child pages*)
7. *Level x pages* ≅CEILING(*previous level pages* / *minimum child pages*)
8. ***Total nonleaf pages*** ≅(*level 2 pages* + *level 3 pages* + ...+ *level x pages* until *x* = 1)

Calculate the total space requirement: Finally, calculate the number of kilobytes required for an index built by LOAD.

1. *Free pages* ≅FLOOR(*total leaf pages* / *p*), or 0 if *p* = 0
2. *Space map pages* ≅CEILING((*tree pages* + *free pages*) / 8131)
3. *Tree pages* ≅MAX(2, (*total leaf pages* + *total nonleaf pages*))
4. *Total index pages* ≅MAX(4, (1 + *tree pages* + *free pages* + *space map pages*))
5. *Total space requirement* ≅4 × (*total index pages* + 2)

In the following example of the entire calculation, assume that an index is defined with these characteristics:

- It is unique.
- The table it indexes has 100000 rows.
- The key is a single column defined as CHAR(10) NOT NULL.
- The value of PCTFREE is 5.
- The value of FREEPAGE is 4.

The calculations are shown in Table 29.

Table 29. The total space requirement for an index

Quantity	Calculation	Result
Length of key	*k*	10
Average number of duplicate keys	*n*	1
PCTFREE	*f*	5
FREEPAGE	*p*	4
Calculate total leaf pages		
Space per key	*k* + 7	17
Usable space per page	FLOOR((100 − *f*) × 4038/100)	3844
Entries per page	FLOOR(*usable space per page* / *space per key*)	225
Total leaf pages	CEILING(*number of table rows* / *entries per page*)	445

Table 29. The total space requirement for an index (continued)

Quantity	Calculation	Result
Calculate total nonleaf pages		
Space per key	$k + 7$	17
Usable space per page	FLOOR(MAX(90, (100 − f)) × (4046/100)	3836
Entries per page	FLOOR(*usable space per page / space per key*)	226
Minimum child pages	MAX(2, (*entries per page* + 1))	227
Level 2 pages	CEILING(*total leaf pages / minimum child pages*)	2
Level 3 pages	CEILING(*level 2 pages / minimum child pages*)	1
Total nonleaf pages	(*level 2 pages* + *level 3 pages* +...+ *level x pages* until $x = 1$)	3
Calculate total space required		
Free pages	FLOOR(*total leaf pages / p*), or 0 if $p = 0$	111
Tree pages	MAX(2, (*total leaf pages* + *total nonleaf pages*))	448
Space map pages	CEILING((*tree pages* + *free pages*)/8131)	1
Total index pages	MAX(4, (1 + *tree pages* + *free pages* + *space map pages*))	561
TOTAL SPACE REQUIRED, in KB	4 × (*total index pages* + 2)	2252

Part 3. Security and auditing

Chapter 8. Introduction to security and auditing in DB2 . . . 127
Reading strategies for security . . . 127
If you are new to DB2 . . . 127
| If you have used DB2 before . . . 128
Reading strategies for auditing . . . 128
Controlling data access . . . 128
Access control within DB2 . . . 129
Granting privileges within DB2 . . . 130
Using ownership privileges within DB2 . . . 130
| Using multilevel security . . . 130
Disabling access control within DB2 . . . 131
Using an exit routine to control authorization checking . . . 131
Controlling access to a DB2 subsystem . . . 131
Controlling access at a local DB2 subsystem 131
Controlling access from a remote application 132
Data set protection . . . 132

Chapter 9. Controlling access to DB2 objects 133
Explicit privileges and authorities . . . 133
Authorization IDs . . . 134
Granting explicit privileges . . . 134
Administrative authorities . . . 138
Field-level access control by views . . . 143
Authority over the catalog and directory . . . 144
Implicit privileges of ownership . . . 145
Establishing ownership of objects with unqualified names . . . 145
Establishing ownership of objects with qualified names . . . 146
Privileges by type of object . . . 146
Granting implicit privileges . . . 147
Changing ownership . . . 148
Privileges exercised through a plan or a package 148
Establishing or changing ownership of a plan or a package . . . 148
Qualifying unqualified names . . . 149
Authorization to execute . . . 149
Checking authorization at a second DB2 server . . . 150
Checking authorization to execute an RRSAF application without a plan . . . 151
Caching authorization IDs for best performance . . . 152
Controls in the program . . . 153
Recommendation against use of controls in the program . . . 153
Restricting a plan or a package to particular systems . . . 154
Privileges required for remote packages . . . 154
Access control for user-defined functions and stored procedures . . . 155
Additional authorization for stored procedures 156
Controlling access to catalog tables for stored procedures . . . 156
Example of roles and authorizations for a routine . . . 157
Implementing the user-defined function . . . 157
Defining the user-defined function . . . 159
Using the user-defined function . . . 160
How DB2 determines authorization IDs . . . 161
Which IDs can exercise which privileges . . . 161
Authorization for dynamic SQL statements . . . 164
Run behavior . . . 165
Bind behavior . . . 165
Define behavior . . . 165
Invoke behavior . . . 166
Common attribute values for bind, define, and invoke behavior . . . 166
Example of determining authorization IDs for dynamic SQL statements in routines . . . 168
Simplifying authorization . . . 171
Composite privileges . . . 171
Multiple actions in one statement . . . 171
Matching job titles with privileges . . . 171
Examples of granting and revoking privileges . . . 173
Examples using the GRANT statement . . . 174
System administrator's privileges . . . 175
Package administrator's privileges . . . 176
Database administrator's privileges . . . 176
Database controller's privileges . . . 176
Examples with secondary IDs . . . 176
Application programmers' privileges . . . 177
Privileges for binding the plan . . . 178
Moving PROGRAM1 into production . . . 178
Spiffy Computer Company's approach to distributed data . . . 179
Examples using the REVOKE statement . . . 180
Privileges granted from two or more IDs . . . 181
Revoking privileges granted by other IDs 181
Restricting revocation of privileges . . . 182
Other implications of the REVOKE statement 185
Finding catalog information about privileges . . . 187
Retrieving information in the catalog . . . 187
Retrieving all DB2 authorization IDs with granted privileges . . . 188
Retrieving multiple grants of the same authorization . . . 188
Retrieving all IDs with DBADM authority 189
Retrieving IDs authorized to access a table 189
Retrieving IDs authorized to access a routine 190
Retrieving the tables an ID is authorized to access . . . 190
Retrieving the plans and packages that access a table . . . 190
Creating views of the DB2 catalog tables . . . 191
| Multilevel security . . . 192
| Introduction to multilevel security . . . 192
| Users and objects in multilevel security . . . 192
| Security labels . . . 193
| Mandatory access checking . . . 193

© Copyright IBM Corp. 1982, 2008

Dominant, reverse dominant, equivalent, and disjoint security labels . . . 194
Write-down control . . . 195
Implementing multilevel security with DB2 . . . 195
Implementing multilevel security at the object level . . . 196
Implementing multilevel security with row-level granularity . . . 197
Working with data in a multilevel-secure environment. . . . 199
Using the SELECT statement with multilevel security . . . 199
Using the INSERT statement with multilevel security . . . 200
Using the UPDATE statement with multilevel security . . . 201
Using the DELETE statement with multilevel security . . . 203
Using utilities with multilevel security . . . 204
Using views to restrict access . . . 205
Global temporary tables with multilevel security . . . 205
Materialized query tables with multilevel security . . . 205
Constraints and multilevel security . . . 206
Field procedures, edit procedures, validation procedures, and multilevel security . . . 206
Triggers and multilevel security . . . 206
Implementing multilevel security in a distributed environment . . . 206
TCP/IP support for multilevel security . . . 206
SNA support for multilevel security . . . 207
Data encryption through built-in functions . . . 207
Defining columns for encrypted data . . . 208
Defining encryption at the column level . . . 208
Using column-level encryption with views 209
Using password hints with column-level encryption . . . 210
Defining encryption at the value level . . . 210
Using password hints with value-level encryption . . . 211
Ensuring accurate predicate evaluation for encrypted data . . . 211
Encrypting non-character values . . . 211
Performance recommendations for data encryption . . . 212

Chapter 10. Controlling access through a closed application . . . 215
Registration tables . . . 216
Columns of the ART . . . 216
Columns of the ORT . . . 217
Controlling data definition . . . 218
Installing data definition control support . . . 219
Controlling data definition by application name 220
Controlling data definition by application name with exceptions. . . . 221
Controlling data definition by object name . . . 222
Controlling data definition by object name with exceptions . . . 224
Registering sets of objects . . . 225
Managing the registration tables and their indexes 226
Creating the tables and indexes . . . 226
Naming registration tables and indexes. . . . 227
Dropping registration tables and indexes . . . 227
Creating a table space for the ART and ORT 228
Adding columns . . . 228
Updating the tables . . . 228
Stopping data definition control . . . 228

Chapter 11. Controlling access to a DB2 subsystem . . . 231
Controlling local requests . . . 232
Processing connections . . . 232
Steps in processing connections . . . 233
Supplying secondary IDs for connection requests . . . 234
Required CICS specifications . . . 235
Processing sign-ons . . . 236
Steps in processing sign-ons . . . 236
Supplying secondary IDs for sign-on requests 237
Controlling requests from remote applications . . . 238
Overview of security mechanisms for DRDA and SNA . . . 238
Mechanisms used by DB2 UDB for z/OS as a requester . . . 238
Mechanisms accepted by DB2 UDB for z/OS as a server . . . 239
The communications database for the server . . . 240
Columns used in SYSIBM.LUNAMES . . . 240
Columns used in SYSIBM.USERNAMES . . . 241
Controlling inbound connections that use SNA protocols . . . 242
Controlling what LUs can attach to the network . . . 242
Verifying a partner LU . . . 242
Accepting a remote attachment request . . . 242
Do you permit access? . . . 243
Do you manage inbound IDs through DB2 or RACF? . . . 243
Do you trust the partner LU? . . . 244
If you use passwords, are they encrypted? 244
If you use Kerberos, are users authenticated? 244
Do you translate inbound IDs? . . . 247
How do you associate inbound IDs with secondary IDs? . . . 249
Controlling inbound connections that use TCP/IP protocols . . . 249
Steps, tools, and decisions . . . 250
Planning to send remote requests. . . . 252
The communications database for the requester 253
Columns used in SYSIBM.LUNAMES . . . 253
Columns used in SYSIBM.IPNAMES . . . 254
Columns used in SYSIBM.USERNAMES . . . 255
Columns used in SYSIBM.LOCATIONS . . . 256
What IDs you send . . . 257
Translating outbound IDs . . . 260
Sending passwords . . . 262
Sending RACF encrypted passwords . . . 263
Sending RACF PassTickets . . . 263
Sending encrypted passwords from a workstation . . . 264

Establishing RACF protection for DB2 264
Defining DB2 resources to RACF 265
Define the names of protected access profiles 266
Enable RACF checking for the DSNR and SERVER classes. 266
Enable partner-LU verification. 267
Permitting RACF access 267
Define RACF user IDs for DB2 started tasks 267
Add RACF groups 270
Permit access for users and groups 271
Issuing DB2 commands 274
Establishing RACF protection for stored procedures 274
Step 1: Control access by using the attachment facilities (required). 274
Step 2: Control access to WLM (optional) . . 275
Step 3: Control access to non-DB2 resources (optional). 277
Establishing RACF protection for TCP/IP . . . 278
Establishing Kerberos authentication through RACF 278
Other methods of controlling access 279

Chapter 12. Protecting data sets through RACF 281
Adding groups to control DB2 data sets 281
Creating generic profiles for data sets 281
Permitting DB2 authorization IDs to use the profiles 283
Allowing DB2 authorization IDs to create data sets 283

Chapter 13. Auditing 285
Using the audit trace 285
Starting the audit trace 286
Stopping the audit trace 286
Audit class descriptions 287
Limitations of the audit trace 287
Auditing in a distributed data environment . . 288
Auditing a specific table. 288
The role of authorization IDs in auditing 289
Auditing specific IDs 290
Determining which IDs hold privileges and authorities 290
Using audit records 290
The contents of audit records 291
Formatting audit records 291
Suggested audit trace reports 291
Using other sources of audit information 292
Determining which security measures are enabled 292
Ensuring data accuracy and consistency 293
Ensuring that the required data is present . . . 293
Ensuring that data is unique 293
Ensuring that data fits a pattern or value range 293
Ensuring that data is consistent 294
Ensuring that changes are tracked 295
Ensuring that concurrent users access consistent data 295
Checking for lost and incomplete transactions 296
Determining whether data is consistent. 296
Automatically checking the consistency of data 297
Submitting SQL queries to check data consistency 297
Checking data consistency with the CHECK utility 297
Checking data consistency with the DISPLAY DATABASE command 298
Checking data consistency with the REPORT utility 298
Checking data consistency with the operation log 298
Using internal integrity reports to check data consistency 298

Chapter 14. A sample security plan for employee data 301
Securing manager access 301
Granting the SELECT privilege to managers . . 302
Securing distributed access 303
Actions at the central server location . . . 303
Actions at remote locations 304
Auditing manager use 304
Securing payroll operations access 305
Securing compensation updates 305
Additional controls for compensation updates 306
Granting privileges to payroll operations and payroll management 307
Auditing payroll operations and payroll management 307
Securing administrator, owner, and other access 308
Securing access by IDs with database administrator authority 308
Securing access by IDs with system administrator authority 308
Securing access by owners with implicit privileges on objects 309
Securing access by other users. 310

Chapter 8. Introduction to security and auditing in DB2

The two topics of security and auditing overlap a great deal, but not completely.

Security covers control of access, whether to the DB2 subsystem, its data, or its resources. A security plan sets objectives for a security system, determining who has access to what, and under which circumstances. The security plan also describes how to meet the objectives by using functions of DB2, functions of other programs, and administrative procedures.

Auditing is how you determine whether the security plan is working and who has accessed data. Auditing includes questions, such as:
- Have attempts been made to gain unauthorized access?
- Is the data in the subsystem accurate and consistent?
- Are system resources used efficiently?

Because the two topics are not the same, this chapter suggests different ways to approach the information about security and auditing.

The following topics provide additional information:
- "Reading strategies for security"
- "Reading strategies for auditing" on page 128
- "Controlling data access" on page 128

Reading strategies for security

If you have any sensitive data in your DB2 subsystem, you must plan carefully to protect your data. The security plan sets objectives for allowing and preventing access, and it describes the means of achieving these objectives. Because the nature of the security plan depends entirely on the data that you are protecting, there is no single way to approach the task. Consider the suggestions in the following sections:
- "If you are new to DB2"
- "If you have used DB2 before" on page 128

If you are new to DB2

Use the following guidelines to learn about security:
1. Read "Controlling data access" on page 128 carefully.
2. Skim Chapter 9, "Controlling access to DB2 objects," on page 133 through Chapter 13, "Auditing," on page 285. These chapters describe the tools that you use to implement your plan, but they probably contain more detail than you want during your first reading.
3. Read the case study in Chapter 14, "A sample security plan for employee data," on page 301. The sample plan describes the type of data access decisions that you must make for your own data.
4. Skim Table 37 on page 137 through Table 34 on page 136 for a brief overview of the range of objects that DB2 protects.
5. List your security objectives and the means that you will use to achieve them.

© Copyright IBM Corp. 1982, 2008

6. Reread the sections that describe the functions that you expect to use. Ensure that you can achieve the objectives that you have set, or adjust your plan accordingly.

If you have used DB2 before

This section contains a summary of the changes in Version 8 for security and auditing.

Multilevel security: You can use multilevel security for object-level access control, and you can use multilevel security with row-level granularity. For information about multilevel security, see "Multilevel security" on page 192.

Encryption: You can use new built-in functions for data encryption and decryption that let you protect sensitive data as it is stored in or retrieved from a DB2 subsystem. For more information about encryption, see "Data encryption through built-in functions" on page 207.

Session variables and special registers: You can use new special registers and session variables to facilitate information sharing between applications. You can also use these new values to help enforce a security policy. For more information about new session variables and special registers, see Chapter 9, "Controlling access to DB2 objects," on page 133.

Authorization: You can now return secondary authorization IDs to applications and control authorization on materialized query tables and sequences. For information about these changes to authorization, see Chapter 9, "Controlling access to DB2 objects," on page 133.

Reading strategies for auditing

For specific information about auditing, read Chapter 13, "Auditing," on page 285.

If you are also interested in controlling data access, first read "Controlling data access." Then read Chapter 9, "Controlling access to DB2 objects," on page 133. Finally, read Chapter 13, "Auditing," on page 285.

Controlling data access

Access to data includes, but is not limited to, access by a person that is engaged in an interactive terminal session. For example, access can be from a program that is running in batch mode, or from an IMS or CICS transaction. Hence, so as not to focus your attention too narrowly, this chapter uses the term *process* to represent all access to data.

Several routes exist from a process to DB2 data. DB2 controls each route except the data set protection route, as shown in Figure 7 on page 129.

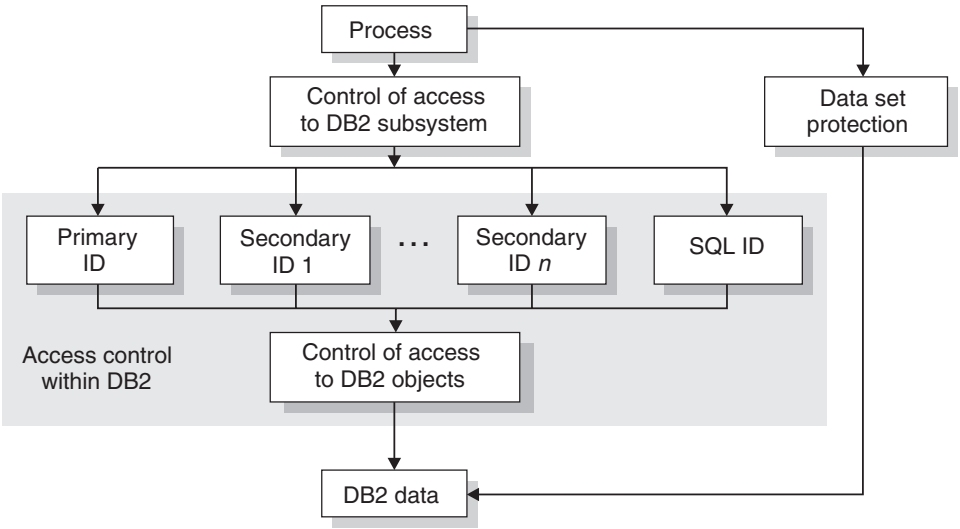


Figure 7. DB2 data access control

One of the ways that DB2 controls access to data is through the use of identifiers (IDs). Four types of IDs exist:

Primary authorization ID
Generally, the primary authorization ID identifies a process. For example, statistics and performance trace records use a primary authorization ID to identify a process.

Secondary authorization ID
A secondary authorization ID, which is optional, can hold additional privileges that are available to the process. For example, a secondary authorization ID can be a Resource Access Control Facility (RACF) group ID.

SQL ID
An SQL ID holds the privileges that are exercised when certain dynamic SQL statements are issued. The SQL ID can be set equal to the primary ID or any of the secondary IDs. If an authorization ID of a process has SYSADM authority, the process can set its SQL ID to any authorization ID.

RACF ID
The RACF ID is generally the source of the primary and secondary authorization IDs (RACF groups). When you use the RACF Access Control Module or multilevel security, the RACF ID is used directly.

Access control within DB2

Within the DB2 subsystem, a process can be represented by a primary authorization ID, one or more secondary IDs, or an SQL ID. Your security and network systems and the choices that you make for DB2 connections affect the use of these IDs.

DB2 relies only on IDs to determine whether to allow or prohibit certain processes. An ID can hold privileges that allow the ID to take certain actions and that prohibit the ID from taking other actions. DB2 does not determine access control based on the process or the person accessing data. Therefore, if the same set of IDs are associated with two different accesses to DB2, DB2 cannot determine whether the IDs involve the same process.

Example: You might know that someone else is using your ID. However, DB2 recognizes only the ID; DB2 does not recognize that someone else is using your ID.

Therefore, this book uses phrases like "an ID owns an object" instead of "a person owns an object" to discuss access control within DB2.

Granting privileges within DB2

DB2 allows you a wide range of granularity when you grant privileges to an ID within DB2. You can grant all of the privileges over a table to an ID, or you can grant an ID individual privileges. For example, you could, separately and specifically, grant an ID the privilege to retrieve data from the table, to insert rows, to delete rows, or to update specific columns. By granting or not granting privileges over views of the table, you can determine exactly what an ID can do to the table, down to the granularity of specific fields. You can also grant an ID specific privileges over databases, plans, packages, and the entire DB2 subsystem.

DB2 also defines sets of related privileges, called *administrative authorities.* When you grant one of the administrative authorities to a person's ID, that person has all of the privileges that are associated with that administrative authority. You can efficiently grant many privileges by granting one administrative authority.

You can also efficiently grant multiple privileges by granting the privilege to execute an application plan or a package. When an ID executes a plan or package, the ID implicitly uses all of the privileges that the owner needed when binding the plan or package. Therefore, granting to an ID the privilege to execute a plan or package can provide a finely detailed set of privileges and can eliminate the need to grant other privileges separately.

Example of granting the privilege to execute: Assume that an application plan issues the INSERT and SELECT statements on several tables. You need to grant INSERT and SELECT privileges only to the plan owner. However, any authorization ID that is later granted the EXECUTE privilege on the plan can perform those same INSERT and SELECT statements by executing the plan. You do not need to explicitly grant the INSERT and SELECT privileges to the ID.

For more information about granting privileges to execute a plan or package, see "Privileges exercised through a plan or a package" on page 148.

Recommendation: Instead of granting privileges to many primary authorization IDs, consider associating each of those primary IDs with the same secondary ID. Then grant the privileges to the secondary ID. You can associate a primary ID with one or more secondary IDs when the primary ID gains access to the DB2 subsystem. DB2 makes the association within an exit routine. The assignment of privileges to the secondary ID is controlled entirely within DB2.

Using ownership privileges within DB2

Ownership of an object carries with it a set of related privileges over the object. For a summary of implicit ownership privileges by type of object, see Table 43 on page 147.

DB2 provides separate controls for creation and ownership of objects. When you create an object, your ID can own the object, or another ID can own the object.

Using multilevel security

Multilevel security allows you to classify objects and users with security labels that are based on hierarchical security levels and non-hierarchical security categories.

Multilevel security prevents unauthorized users from accessing information at a higher classification than their authorization, and prevents users from declassifying information. Using multilevel security with row-level granularity, you can define security for DB2 objects and perform security checks, including row-level security checks. Row-level security checks allow you to control which users have authorization to view, modify, or perform other actions on specific rows of data.

For more information about multilevel security, see "Multilevel security" on page 192

Disabling access control within DB2

If security is not important for your data, you can disable access control within DB2 by setting USE PROTECTION to NO when you install DB2 or update installation settings. If protection in DB2 is disabled, any user that gains access can do anything. However, no user can issue GRANT statements or REVOKE statements when access control within DB2 is disabled.

Using an exit routine to control authorization checking

You can control authorization checking by using a DB2-supplied exit routine or an exit routine that you write. If your installation uses one of these access control authorization exit routines, the exit routine might control authorization checking instead of the authorization checks that are described in this section. For more information about access control authorization exit routines, see Appendix B, "Writing exit routines," on page 1051.

Controlling access to a DB2 subsystem

You can control whether a process can gain access to a specific DB2 subsystem from outside of DB2. A common approach is to grant access only through RACF or some similar security system. Profiles for access to DB2 from various environments and DB2 address spaces are defined as resources to RACF. Each request to access DB2 is associated with an ID. RACF determines whether the ID is authorized for DB2 resources. If the ID is authorized, RACF permits access to DB2.

The RACF system provides several advantages of its own. For example, RACF can:
- Identify and verify the identifier that is associated with a process
- Connect those identifiers to RACF group names
- Log and report unauthorized attempts to access protected resources

Controlling access at a local DB2 subsystem

Local DB2 users are often subject to several checks before they reach DB2. For example, if you run DB2 under TSO and use the TSO logon ID as the DB2 primary authorization ID, TSO verifies your ID when you log on.

When a user gains access to DB2, a user-written or IBM-supplied DSN3@ATH exit routine that is connected to DB2 can perform the following actions:
- Check the authorization ID again
- Change the authorization ID
- Associate the authorization ID with secondary IDs

After these actions are performed, the authorization ID can use the services of an external security system again. For more detailed instructions, see Chapter 11, "Controlling access to a DB2 subsystem," on page 231.

Controlling access from a remote application

You can require remote users to pass several access checks before they reach DB2. For example, you can use RACF or a similar security subsystem to control access from a remote location.

While controlling access from a remote locations, RACF can do the following:

- Verify an ID that is associated with a remote attachment request and check the ID with a password.
- Generate *PassTickets* on the sending side. PassTickets can be used instead of passwords. A PassTicket lets a user gain access to a host system without sending the RACF password across the network. For more information about RACF PassTickets, see"Sending RACF PassTickets" on page 263.

The communications database: You can also control access authentication by using the DB2 communications database (CDB). The CDB is a set of tables in the DB2 catalog that are used to establish conversations with remote database management systems. The CDB can translate IDs before it sends the IDs to the remote system. For more information about CDB requests, see "The communications database for the requester" on page 253. For more information about CDB controls on the server, see "The communications database for the server" on page 240.

Data set protection

The data in a DB2 subsystem is contained in data sets. As Figure 7 on page 129 suggests, those data sets can be accessed without going through DB2. If your data is sensitive, you want to control any route to DB2 data that DB2 does not control.

If you use RACF or a similar security system to control access to DB2, the simplest way to controlling data set access outside of DB2 is to use RACF. If you want to use RACF for data set protection outside of DB2, define RACF profiles for data sets, and permit access to the data sets for certain DB2 IDs.

If your data is very sensitive, consider encrypting the data. Encryption protects against unauthorized access to data sets and to backup copies outside of DB2. You have the following encryption options for protecting sensitive data:

- Built-in data encryption functions, which are described in "Data encryption through built-in functions" on page 207.
- DB2 edit procedures or field procedures, which can use the Integrated Cryptographic Service Facility (ICSF). For information about the ICSF, see *z/OS ICSF Overview*.
- The IBM Data Encryption for IMS and DB2 Databases tool, which is described in *IBM Data Encryption for IMS and DB2 Databases User's Guide*.

Data compression is not a substitute for encryption. In some cases, the compression method does not actually shorten the data. In those cases, the data is left uncompressed and readable. If you encrypt and compress your data, compress it first. After you obtain the maximum compression, encrypt the result. When you retrieve your data, first decrypt the data. After the data is decrypted, decompress the result.

Chapter 9. Controlling access to DB2 objects

The information in this chapter is General-use Programming Interface and Associated Guidance Information, as defined in "Notices" on page 1301.

DB2 controls access to its objects by a set of *privileges*. Each privilege allows a specific action to be taken on some object. Figure 8 shows the four primary ways within DB2 to give an ID access to data.[1]

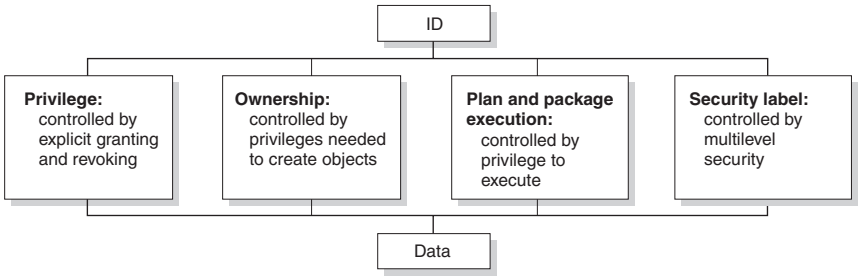


Figure 8. Access to data within DB2

As a security planner, you must be aware of every way to allow access to data. Before you write a security plan, see the following sections:

- "Explicit privileges and authorities"
- "Implicit privileges of ownership" on page 145
- "Privileges exercised through a plan or a package" on page 148
- "Access control for user-defined functions and stored procedures" on page 155
- "Multilevel security" on page 192
- "Data encryption through built-in functions" on page 207

DB2 has primary authorization IDs, secondary authorization IDs, and SQL IDs. Some privileges can be exercised only by one type of ID; other privileges can be exercised by more than one. To decide which IDs should hold specific privileges, see "Which IDs can exercise which privileges" on page 161.

After you decide which IDs should hold specific privileges, you can implement a security plan. Before you begin your plan, you can see what others have done in "Matching job titles with privileges" on page 171 and "Examples of granting and revoking privileges" on page 173.

The DB2 catalog records the privileges that IDs are granted and the objects that IDs own. To check the implementation of your security plan, see "Finding catalog information about privileges" on page 187.

Explicit privileges and authorities

You can control access within DB2 by granting, not granting, or revoking explicit privileges and authorities. Privileges, explicit privileges, and administrative authorities are defined as follows:

1. Certain authorities are assigned when DB2 is installed, and can be reassigned by changing the subsystem parameter (DSNZPARM). You can consider changing the DSNZPARM value to be a fifth way of granting data access in DB2.

© Copyright IBM Corp. 1982, 2008

Privilege
A privilege allows the capability to perform a specific operation, sometimes on a specific object.

Explicit privilege
An explicit privilege is a specific type of privilege. Each explicit privilege has a name and is the result of a GRANT statement or a REVOKE statement. For example, the SELECT privilege.

Administrative authority
An administrative authority is a set of privileges, often covering a related set of objects. Authorities often include privileges that are not explicit, have no name, and cannot be specifically granted. For example, when an ID is granted the SYSOPR administrative authority, the ID is implicitly granted the ability to terminate any utility job.

Privileges and authorities are held by authorization IDs.

Authorization IDs

Every process that connects to or signs on to DB2 is represented by a set of one or more DB2 short identifiers that are called *authorization IDs*. Authorization IDs can be assigned to a process by default procedures or by user-written exit routines. Methods of assigning those IDs are described in detail in Chapter 11, "Controlling access to a DB2 subsystem," on page 231; see especially Table 70 on page 233 and Table 71 on page 234.

When authorization IDs are assigned, every process receives exactly one ID that is called the *primary authorization ID*. All other IDs are *secondary authorization IDs*.

Furthermore, one ID (either primary or secondary) is designated as the *current SQL ID*. You can change the value of the SQL ID during your session.

Example of changing the SQL ID: Suppose that ALPHA is your primary authorization ID or one of your secondary authorization IDs. You can make it your current SQL ID by issuing the following SQL statement:

```
SET CURRENT SQLID = 'ALPHA';
```

If you issue the statement through the distributed data facility (DDF), ALPHA must be one of the IDs that are associated with your process at the location where the statement runs. Your primary ID can be translated before it is sent to a remote location. Secondary IDs are associated with your process at the remote location. The current SQL ID, however, is not translated. For more information about authorization IDs and remote locations, see "Controlling requests from remote applications" on page 238.

An ID with SYSADM authority can set the current SQL ID to any string whose length is less than or equal to 8 bytes.

Authorization IDs that are sent to a connecting server must conform to the security management product guidelines of the server. See the documentation of the security product for the connecting server.

Granting explicit privileges

You can use explicit privileges to grant privileges over objects to IDs. When you grant explicit privileges, you have fine control over which privileges you grant.

This section describes the explicit privileges that you can grant on objects. The descriptions of the explicit privileges are grouped by object and usage:

- Collections in Table 30 on page 135
- Databases in Table 31 on page 135
- Packages in Table 32 on page 136
- Plans in Table 33 on page 136
- Routines in Table 34 on page 136
- Schemas in Table 35 on page 136
- Systems in Table 36 on page 136
- Tables and views in Table 37 on page 137
- Usage in Table 38 on page 138
- Use in Table 39 on page 138

Table 30 shows the collection privileges that DB2 allows.

Table 30. Explicit collection privileges

Collection privilege	Operations allowed for a named package collection
CREATE IN	The BIND PACKAGE subcommand, to name the collection

Table 31 shows the database privileges that DB2 allows.

Table 31. Explicit database privileges

Database privilege	Operations allowed on a named database
CREATETAB	The CREATE TABLE statement, to create tables in the database
CREATETS	The CREATE TABLESPACE statement, to create table spaces in the database
DISPLAYDB	The DISPLAY DATABASE command, to display the database status
DROP	The DROP and ALTER DATABASE statements, to drop or alter the database
IMAGCOPY	The QUIESCE, COPY, and MERGECOPY utilities, to prepare for, make, and merge copies of table spaces in the database; the MODIFY RECOVERY utility, to remove records of copies
LOAD	The LOAD utility, to load tables in the database
RECOVERDB	The RECOVER, REBUILD INDEX, and REPORT utilities, to recover objects in the database and report their recovery status
REORG	The REORG utility, to reorganize objects in the database
REPAIR	The REPAIR and DIAGNOSE utilities (except REPAIR DBD and DIAGNOSE WAIT) to generate diagnostic information about, and repair data in, objects in the database
STARTDB	The START DATABASE command, to start the database
STATS	The RUNSTATS, CHECK, LOAD, REBUILD INDEX, REORG INDEX, REORG TABLESPACE and MODIFY STATISTICS utilities, to gather statistics, check indexes and referential constraints for objects in the database, and delete unwanted statistics history records from the corresponding catalog tables
STOPDB	The STOP DATABASE command, to stop the database

Table 32 shows the package privileges that DB2 allows.

Table 32. Explicit package privileges

Package privilege	Operations allowed for a named package
BIND	The BIND, REBIND, and FREE PACKAGE subcommands, and the DROP PACKAGE statement, to bind or free the package, and, depending on the installation option BIND NEW PACKAGE, to bind a new version of a package
COPY	The COPY option of BIND PACKAGE, to copy a package
EXECUTE	Inclusion of the package in the PKLIST option of BIND PLAN
GRANT ALL	All package privileges

Table 33 shows the plan privileges that DB2 allows.

Table 33. Explicit plan privileges

Plan privilege	Subcommands allowed for a named application plan
BIND	BIND, REBIND, and FREE PLAN, to bind or free the plan
EXECUTE	RUN, to use the plan when running the application

Table 34 shows the routine privileges that DB2 allows.

Table 34. Explicit routine privileges

Routine privileges	Objects available for usage
EXECUTE ON FUNCTION	A user-defined function
EXECUTE ON PROCEDURE	A stored procedure

Table 35 shows the schema privileges that DB2 allows.

Table 35. Explicit schema privileges

Schema privileges	Operations available for usage
CREATEIN	Create distinct types, user-defined functions, triggers, and stored procedures in the designated schemas
ALTERIN	Alter user-defined functions or stored procedures, or specify a comment for distinct types, user-defined functions, triggers, and stored procedures in the designated schemas
DROPIN	Drop distinct types, user-defined functions, triggers, and stored procedures in the designated schemas

Table 36 shows the system privileges that DB2 allows.

Table 36. Explicit system privileges

System privilege	Operations allowed on the system
ARCHIVE	The ARCHIVE LOG command, to archive the current active log, the DISPLAY ARCHIVE command, to give information about input archive logs, the SET LOG command, to modify the checkpoint frequency specified during installation, and the SET ARCHIVE command, to control allocation and deallocation of tape units for archive processing.

Table 36. Explicit system privileges (continued)

System privilege	Operations allowed on the system
BINDADD	The BIND subcommand with the ADD option, to create new plans and packages
BINDAGENT	The BIND, REBIND, and FREE subcommands, and the DROP PACKAGE statement, to bind, rebind, or free a plan or package, or copy a package, on behalf of the grantor. The BINDAGENT privilege is intended for separation of function, not for added security. A bind agent with the EXECUTE privilege might be able to gain all the authority of the grantor of BINDAGENT.
BSDS	The RECOVER BSDS command, to recover the bootstrap data set
CREATEALIAS	The CREATE ALIAS statement, to create an alias for a table or view name
CREATEDBA	The CREATE DATABASE statement, to create a database and have DBADM authority over it
CREATEDBC	The CREATE DATABASE statement, to create a database and have DBCTRL authority over it
CREATESG	The CREATE STOGROUP statement, to create a storage group
CREATETMTAB	The CREATE GLOBAL TEMPORARY TABLE statement, to define a created temporary table
DISPLAY	The DISPLAY ARCHIVE, DISPLAY BUFFERPOOL, DISPLAY DATABASE, DISPLAY LOCATION, DISPLAY LOG, DISPLAY THREAD, and DISPLAY TRACE commands, to display system information
MONITOR1	Receive trace data that is not potentially sensitive
MONITOR2	Receive all trace data
RECOVER	The RECOVER INDOUBT command, to recover threads
STOPALL	The STOP DB2 command, to stop DB2
STOSPACE	The STOSPACE utility, to obtain data about space usage
TRACE	The START TRACE, STOP TRACE, and MODIFY TRACE commands, to control tracing

Table 37 shows the table and view privileges that DB2 allows.

Table 37. Explicit table and view privileges

Table or view privilege	SQL statements allowed for a named table or view
ALTER	ALTER TABLE, to change the table definition
DELETE	DELETE, to delete rows
INDEX	CREATE INDEX, to create an index on the table
INSERT	INSERT, to insert rows
REFERENCES	ALTER or CREATE TABLE, to add or remove a referential constraint referring to the named table or to a list of columns in the table
SELECT	SELECT, to retrieve data from the table
TRIGGER	CREATE TRIGGER, to define a trigger on a table

Table 37. Explicit table and view privileges (continued)

Table or view privilege	SQL statements allowed for a named table or view
UPDATE	UPDATE, to update all columns or a specific list of columns
GRANT ALL	SQL statements of all table privileges

Table 38 shows the usage privileges that DB2 allows.

Table 38. Explicit usage privileges

Usage privileges	Objects available for usage
USAGE ON DISTINCT TYPE	A distinct type
USAGE ON JAR (Java class for a routine)	A Java class
USAGE ON SEQUENCE	A sequence

Table 39 shows the use privileges that DB2 allows.

Table 39. Explicit use privileges

Use privileges	Objects available for use
USE OF BUFFERPOOL	A buffer pool
USE OF STOGROUP	A storage group
USE OF TABLESPACE	A table space

Privileges needed for statements, commands, and utility jobs: For lists of all privileges and authorities that let you perform the following actions, consult the appropriate resource:

- To execute a particular SQL statement, see the description of the statement in Chapter 5 of *DB2 SQL Reference*.
- To issue a particular DB2 command, see the description of the command in Chapter 2 of *DB2 Command Reference*.
- To run a particular type of utility job, see the description of the utility in *DB2 Utility Guide and Reference*.

Administrative authorities

Figure 9 on page 139 shows how privileges are grouped into authorities and how the authorities form a branched hierarchy. Table 41 on page 140 supplements Figure 9 on page 139 and includes the capabilities of each authority.

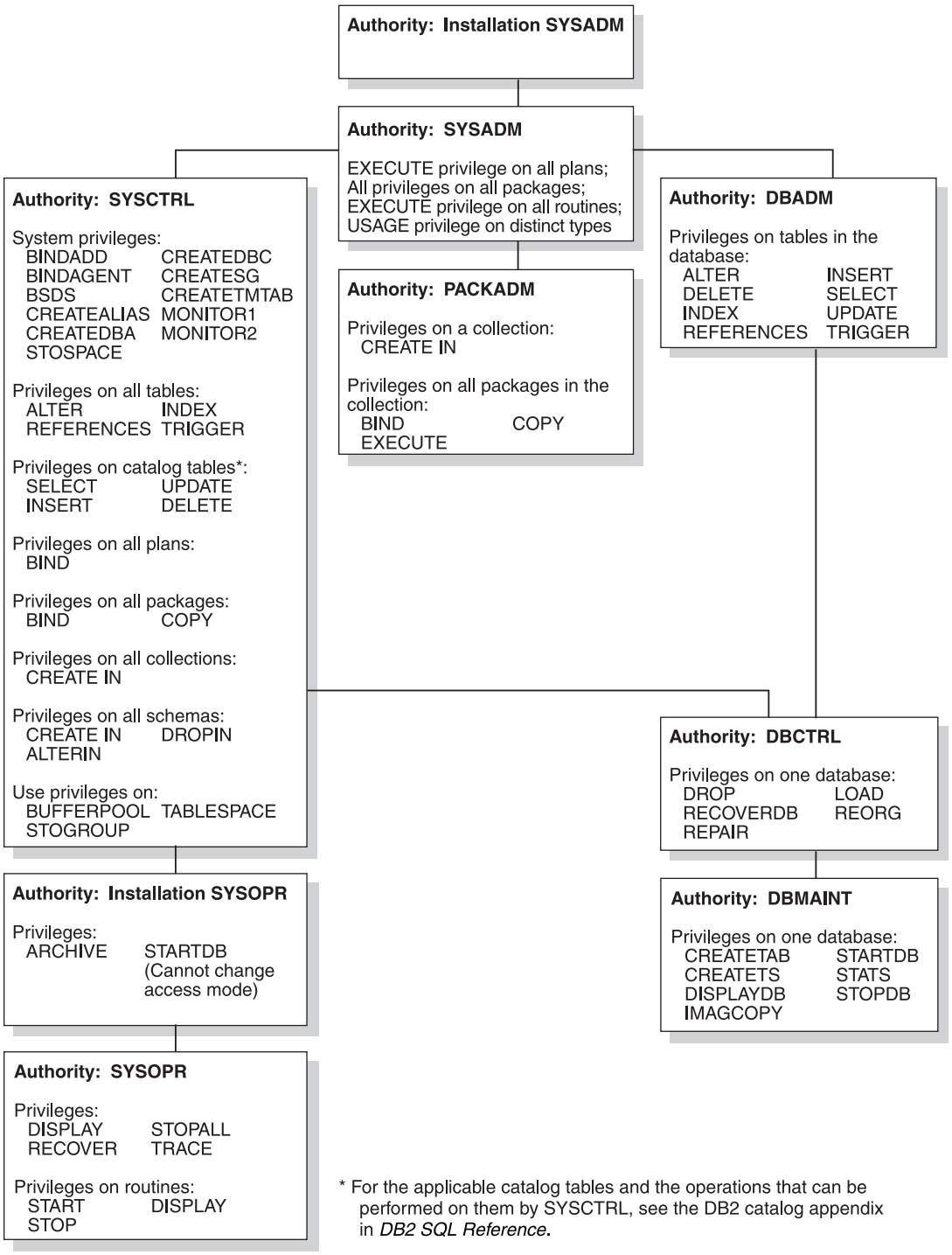


Figure 9. Individual privileges of administrative authorities. Each authority includes the privileges in its box and all of the privileges of each authority in the boxes that are beneath it. Installation SYSOPR authority is an exception; it can do some things that SYSADM and SYSCTRL cannot do.

Table 41 on page 140 shows DB2 authorities and the actions that they are allowed to perform.

Table 41. DB2 authorities

Authority	Description
SYSOPR	System operator: • Can issue most DB2 commands • **Cannot** issue ARCHIVE LOG, START DATABASE, STOP DATABASE, and RECOVER BSDS • Can terminate any utility job • Can run the DSN1SDMP utility If held with the GRANT option, SYSOPR can grant these privileges to others.
Installation SYSOPR	One or two IDs are assigned this authority when DB2 is installed. They have all the privileges of SYSOPR, plus: • The authority is **not** recorded in the DB2 catalog. Therefore, the catalog does not need to be available to check the Installation SYSOPR authority. • No ID can revoke the authority; it can be removed only by changing the module that contains the subsystem initialization parameters (typically DSNZPARM). IDs with Installation SYSOPR authority can also: • Access DB2 when the subsystem is started with ACCESS(MAINT). • Run all allowable utilities on the directory and catalog databases (DSNDB01 and DSNDB06). • Run the REPAIR utility with the DBD statement. • Start and stop the database that contains the application registration table (ART) and the object registration table (ORT). For more information on these tables, see Chapter 10, "Controlling access through a closed application," on page 215. • Issue dynamic SQL statements that are not controlled by the DB2 governor. • Issue a START DATABASE command to recover objects that have LPL entries or group buffer pool recovery-pending status. These IDs cannot change the access mode.
PACKADM	Package administrator: • Has all package privileges on all packages in specific collections • Has the CREATE IN privilege on those specific collections If PACKADM authority is held with the GRANT option, PACKADM can grant those privileges to others. If the installation option BIND NEW PACKAGE is BIND, PACKADM also has the privilege to add new packages or new versions of existing packages.
DBMAINT	Database maintenance authority allows privileges over a specific database to an ID. DBMAINT can perform the following actions within a specific database: • Create objects • Run utilities that don't change data • Issue commands • Terminate all utilities on the database except DIAGNOSE, REPORT, and STOSPACE If held with the GRANT option, DBMAINT can grant those privileges to others.

Table 41. DB2 authorities (continued)

Authority	Description
DBCTRL	Database controller authority includes DBMAINT privileges over a specific database. Additionally, DBCTRL has database privileges to run utilities that can change the data. If the value of field DBADM CREATE AUTH on installation panel DSNTIPP was set to YES during DB2 installation, the ID with DBCTRL authority can create an alias for another user ID on any table in the database. If held with the GRANT option, DBCTRL can grant those privileges to others.
DBADM	Database administrator authority includes DBCTRL privileges over a specific database. Additionally, DBADM has privileges to access any tables in a specific database by using SQL statements. DBADM can also perform the following actions: • Drop or alter any table space, table, or index in the database • Issue a COMMENT, LABEL, or LOCK TABLE statement for any table in the database • Issue a COMMENT statement for any index in the database If the value of field DBADM CREATE AUTH on installation panel DSNTIPP was set to YES during DB2 installation, an ID with DBADM authority can create the following objects: • A view for another ID. The view must be based on at least one table, and that table must be in the database under DBADM authority. For more information about creating views, see the description of the CREATE VIEW statement in *DB2 SQL Reference*. • An alias for another ID on any table in the database. An ID with DBADM authority on one database can create a view on tables and views in that database and other databases only if the ID has all the privileges that are required to create the view. For example, an ID with DBADM authority cannot create a view on a view that is owned by another ID. If held with the GRANT option, DBADM can grant these privileges to others.

Table 41. DB2 authorities (continued)

Authority	Description
SYSCTRL	The system controller authority is designed for administering a system that contains sensitive data. The system controller has nearly complete control of the DB2 subsystem. However, the system controller **cannot** access user data directly unless the privilege to do so is explicitly granted. SYSCTRL can: • Act as installation SYSOPR (when the catalog is available) or DBCTRL over any database • Run any allowable utility on any database • Issue a COMMENT ON, LABEL ON, or LOCK TABLE statement for any table • Create a view on any catalog table for itself or for other IDs • Create tables and aliases for itself or for others IDs • Bind a new plan or package and name any ID as the owner of the plan or package Without additional privileges, SYSCTRL **cannot**: • Execute DML statements on user tables or views • Run plans or packages • Set the current SQL ID to a value that is not one of its primary or secondary IDs • Start or stop the database that contains the ART and the ORT • Act fully as SYSADM or as DBADM over any database • Access DB2 when the subsystem is started with ACCESS(MAINT) SYSCTRL authority is intended to separate system control functions from administrative functions. However, SYSCTRL is not a complete solution for a high-security system. If any plans have their EXECUTE privilege granted to PUBLIC, an ID with SYSCTRL authority can grant itself SYSADM authority. The only control over such actions is to audit the activity of IDs with high levels of authority.
SYSADM	System administrator authority includes all SYSCTRL, PACKADM, and DBADM privileges, including access to all data. SYSADM can perform the following actions and grant other IDs the privilege to perform these actions: • Use all the privileges of DBADM over any database • Use EXECUTE and BIND on any plan or package and COPY on any package • Use privileges over views that are owned by others • Set the current SQL ID to any valid value • Create and drop synonyms and views for other IDs on any table • Use any valid value for OWNER in BIND or REBIND • Drop database DSNDB07 SYSADM can perform the following actions, but **cannot** grant other IDs the privilege to perform the following actions: • Drop or alter any DB2 object, except system databases • Issue a COMMENT ON or LABEL ON statement for any table or view • Terminate any utility job Although SYSADM cannot grant the preceding privileges explicitly, SYSADM can grant these privileges to other IDs by granting them SYSADM authority.

Table 41. DB2 authorities (continued)

Authority	Description
Installation SYSADM	One or two IDs are assigned installation SYSADM authority when DB2 is installed. They have all the privileges of SYSADM, plus: • DB2 does **not** record the authority in the catalog. Therefore, the catalog does not need to be available to check installation SYSADM authority. (The authority outside of the catalog is crucial. For example, if the directory table space DBD01 or the catalog table space SYSDBAUT is stopped, DB2 might not be able to check the authority to start it again. Only an installation SYSADM can start it.) • No ID can revoke installation SYSADM authority. You can remove the authority only by changing the module that contains the subsystem initialization parameters (typically DSNZPARM). IDs with installation SYSADM authority can also perform the following actions: • Run the CATMAINT utility • Access DB2 when the subsystem is started with ACCESS(MAINT) • Start databases DSNDB01 and DSNDB06 when they are stopped or in restricted status • Run the DIAGNOSE utility with the WAIT statement • Start and stop the database that contains the ART and the ORT

Field-level access control by views

Any of the table privileges except ALTER, REFERENCES, TRIGGER, and INDEX can be granted on a view. By creating a view and granting privileges on it, you can give an ID access to only a specific combination of data. The capability is sometimes called “field-level access control” or “field-level sensitivity.”

Example: Suppose that you want the ID MATH110 to be able to extract the following column data from the sample employee table for statistical investigation: HIREDATE, JOB, EDLEVEL, SEX, SALARY, BONUS, and COMM for DSN8810.EMP. However, you want to impose the following restrictions:
- No access to employee names or identification numbers
- No access to data for employees hired before 1996
- No access to data for employees with an education level less than 13
- No access to data for employees whose job is MANAGER or PRES

To do that, create and name a view that shows exactly that combination of data. You can create the view with the following CREATE statement:

```
CREATE VIEW SALARIES AS
   SELECT HIREDATE, JOB, EDLEVEL, SEX, SALARY, BONUS, COMM
      FROM DSN8810.EMP
      WHERE HIREDATE > '1995-12-31' AND
            EDLEVEL >= 13 AND
            JOB <> 'MANAGER' AND
            JOB <> 'PRES';
```

Then grant the SELECT privilege on the view SALARIES to MATH110 with the following statement:

```
GRANT SELECT ON SALARIES TO MATH110;
```

After you grant the privilege, MATH110 can execute SELECT statements on the restricted set of data only.

Alternatively, you can give an ID access to only a specific combination of data by using multilevel security with row-level granularity. For information about multilevel security with row-level granularity, see “Multilevel security” on page 192.

Authority over the catalog and directory

The DB2 catalog is in the DSNDB06 database. An ID with SYSCTRL or SYSADM authority can control access to the catalog in the following ways:

- Grant privileges or authorities on that database or on its tables or views
- Bind plans or packages that access the catalog

However, unlike SYSADM, SYSCTRL cannot act as DBADM over database DSNDB06.

Authorities that are granted on DSNDB06 also cover database DSNDB01, which contains the DB2 directory. An ID with SYSADM authority can control access to the directory by granting privileges to run utilities (that are listed in Table 42) on DSNDB06, but cannot grant privileges on DSNDB01 directly.

Table 42 shows which utilities IDs with different authorities can run on the DSNDB01 and DSNDB06 databases.

Table 42. Utility privileges on the DB2 catalog and directory

	Authorities		
Utilities	**Installation SYSOPR, SYSCTRL, SYSADM, Installation SYSADM**	**DBCTRL, DBADM on DSNDB06**	**DBMAINT on DSNDB06**
LOAD [1]	No	No	No
REPAIR DBD	No	No	No
CHECK DATA	Yes	No	No
CHECK LOB	Yes	No	No
REORG TABLESPACE	Yes	No	No
STOSPACE	Yes	No	No
REBUILD INDEX	Yes	Yes	No
RECOVER	Yes	Yes	No
REORG INDEX	Yes	Yes	No
REPAIR	Yes	Yes	No
REPORT	Yes	Yes	No
CHECK INDEX	Yes	Yes	Yes
COPY	Yes	Yes	Yes
MERGECOPY	Yes	Yes	Yes
MODIFY	Yes	Yes	Yes
QUIESCE	Yes	Yes	Yes
RUNSTATS	Yes	Yes	Yes

Note: LOAD can be used to add lines to SYSIBM.SYSSTRINGS. LOAD cannot be run on other DSNDB01 or DSNDB06 tables.

Implicit privileges of ownership

When you create a DB2 object by issuing an SQL statement, you establish its name and its ownership. The owner of an object implicitly holds certain privileges over that object.

Exception: Plans and packages are not created with SQL CREATE statements, and they have unique features of their own. For information about these features, see "Privileges exercised through a plan or a package" on page 148 and "Access control for user-defined functions and stored procedures" on page 155.

Establishing ownership of objects with unqualified names

If an object name is unqualified, the type of object determines how ownership of the object is established.

If the name of a table, view, index, alias, or synonym is unqualified, you establish the object's ownership in the following ways:

- If you issue the CREATE statement dynamically, perhaps using SPUFI, QMF, or some similar program, the owner of the created object is your current SQL ID. That ID must have the privileges that are needed to create the object.
- If you issue the CREATE statement statically, by running a plan or package that contains it, the ownership of the created object depends on the option used for the bind operation. You can bind the plan or package with either the QUALIFIER option, the OWNER option, or both.
  - If the plan or package is bound with the QUALIFIER option only, the QUALIFIER is the owner of the object. The QUALIFIER option allows the binder to name a qualifier to use for all unqualified names of tables, views, indexes, aliases, or synonyms that appear in the plan or package.
  - If the plan or package is bound with the OWNER option only, the OWNER is the owner of the object.
  - If the plan or package is bound with both the QUALIFIER option and the OWNER option, the QUALIFIER is the owner of the object.
  - If neither option is specified, the binder of the plan or package is implicitly the object owner.

In addition, the plan or package owner must have all required privileges on the objects designated by the qualified names.

If the name of a user-defined function, stored procedure, distinct type, or trigger is unqualified, you establish the ownership of one of these objects in these ways:

- If you issue the CREATE statement dynamically, the owner of the created object is your current SQL ID. That ID must have the privileges that are needed to create the object.
- If you issue the CREATE statement statically, by running a plan or package that contains it, the owner of the object is the plan or package owner. You can use the OWNER bind option to explicitly name the object owner. If you do not use the OWNER bind option, the binder of the package or plan is implicitly the object owner.

If the name of a user-defined function, stored procedure, distinct type, sequence,or trigger is unqualified, the implicit qualifier is determined as described in "Qualifying unqualified names" on page 149.

The owner of a JAR (Java class for a routine) that is used by a stored procedure or a user-defined function is the current SQL ID of the process that performs the INSTALL_JAR function. For information about installing a JAR, see *DB2 Application Programming Guide and Reference for Java*.

Establishing ownership of objects with qualified names

If an object name is qualified, the type of object determines how ownership of the object is established.

If you create a table, view, index, or alias with a qualified name, the qualifier becomes the owner of the object, subject to these restrictions for specifying the qualifier:

- If you issue the CREATE statement dynamically, and have no administrative authority, the qualifier must be your primary ID or one of your secondary IDs. However, if your current SQL ID has at least DBCTRL authority, you can use any qualifier for a table or an index. If your current SQL ID has at least DBADM authority and the value of field DBADM CREATE AUTH on installation panel DSNTIPP was set to YES during DB2 installation, you can also use any qualifier for a view.
- If you issue the CREATE statement statically, and the owner of the plan or package that contains the statement has no administrative authority, the qualifier can only be the owner. However, if the owner has at least DBCTRL authority, the plan or package can use any qualifier for a table or for an index. If the owner of the plan or package has at least DBADM authority and the value of field DBADM CREATE AUTH on installation panel DSNTIPP was set to YES during DB2 installation, the owner can also use any qualifier for a view.

If you create a distinct type, user-defined function, stored procedure, sequence, or trigger with a qualified name, the qualifier is the *schema name*. The schema name identifies the schema to which the object belongs. You can consider all of objects that are qualified by the same schema name as a group of related objects. Unlike other objects, however, the qualifier for these objects does not identify the owner of the object. You establish ownership of a distinct type, user-defined function, stored procedure, or trigger in these ways:

- If you issue the CREATE statement dynamically, the owner of the created object is your current SQL ID. That ID must have the privileges that are needed to create the object.
- If you issue the CREATE statement statically by running a plan or package that contains it, the owner of the plan or package implicitly becomes the owner of the object. If you want to explicitly name the object owner, you can use the OWNER bind option when you run the plan or package.

The owner of a JAR (Java class for a routine) that is used by a stored procedure or a user-defined function is the current SQL ID of the process that performs the INSTALL_JAR function. For information about installing a JAR, see *DB2 Application Programming Guide and Reference for Java*.

Privileges by type of object

When you own a DB2 object, you have implicit privileges over that object. Table 43 on page 147 lists the implicit privileges of ownership for each type of object.

Table 43. Implicit privileges of ownership by object type

Object type	Implicit privileges of ownership
Alias	To drop the alias
Database	DBCTRL or DBADM authority over the database, depending on the privilege (CREATEDBC or CREATEDBA) that is used to create it. DBCTRL authority does **not** include the privilege to access data in tables in the database.
Distinct type	To use or drop a distinct type
Index	To alter, comment on, or drop the index
JAR (Java class for a routine)	To replace, use, or drop the JAR
Package	To bind, rebind, free, copy, execute, or drop the package
Plan	To bind, rebind, free, or execute the plan
Sequence	To alter, comment on, use, or drop the sequence
Storage group	To alter or drop the group and to name it in the USING clause of a CREATE INDEX or CREATE TABLESPACE statement
Stored procedure	To execute, alter, drop, start, stop, or display a stored procedure
Synonym	To use or drop the synonym
Table	• To alter or drop the table or any indexes on it • To lock the table, comment on it, or label it • To create an index or view for the table • To select or update any row or column • To insert or delete any row • To use the LOAD utility for the table • To define referential constraints on any table or set of columns • To create a trigger on the table
Table space	To alter or drop the table space and to name it in the IN clause of a CREATE TABLE statement
User-defined functions	To execute, alter, drop, start, stop, or display a user-defined function
View	• To drop, comment on, or label the view, or to select any row or column • To update any row or column, insert or delete any row (if the view is not read-only)

Implicit privileges do not apply to multilevel security. For information about multilevel security, see "Multilevel security" on page 192.

Granting implicit privileges

Some implicit privileges of ownership correspond to privileges that can be granted by a GRANT statement. For those that do correspond, the owner of the object can grant the privilege to another user.

Example: The owner of a table can grant the SELECT privilege on the table to any other user. To grant the SELECT privilege on TABLE3 to USER4, the owner of the table can issue the following statement:

```
GRANT SELECT ON TABLE3 TO USER4
```

Changing ownership

The privileges that are implicit in ownership cannot be revoked. Therefore, you cannot replace an object's owner while the object exists.

If you want to change ownership of an object, follow these steps:
1. Drop the object, which usually deletes all privileges on it[2].
2. Re-create the object with a new owner.

Exception: You can change package or plan ownership while a package or plan exists. For more information about changing package or plan ownership, see "Establishing or changing ownership of a plan or a package."

You might want to share ownership privileges on an object instead of replacing an old owner with a new owner. If so, make the owning ID a secondary ID to which several primary IDs are connected. You can change the list of primary IDs connected to the secondary ID without dropping and re-creating the object.

Privileges exercised through a plan or a package

This section describes the privileges that are required for executing plans and packages. User-defined function and stored procedure packages, also known as routine packages, have unique requirements that are described in "Access control for user-defined functions and stored procedures" on page 155.

The owner of the plan or package must hold privileges for every action an application plan or package performs. However, the owner of a plan or package can grant the privilege to execute a plan or package to any ID. When the EXECUTE privilege on a plan or package is granted to an ID, that ID can execute a plan or package without holding the privileges for every action that the plan or package performs. However, the ID is restricted by the SQL statements in the original program.

Example: The program might contain the following statement:

```
EXEC SQL
  SELECT * INTO :EMPREC FROM DSN8810.EMP
    WHERE EMPNO='000010';
```

The statement puts the data for employee number 000010 into the host structure EMPREC. The data comes from table DSN8810.EMP, but the ID does not have unlimited access to DSN8810.EMP. Instead, the ID that has EXECUTE privilege for this plan can access rows in the DSN8810.EMP table only when EMPNO = '000010'.

If any of the privileges that are required by the plan or package are revoked from the owner, the plan or the package is invalidated. The plan or package must be rebound, and the new owner must have the required privileges.

Establishing or changing ownership of a plan or a package

The BIND and REBIND subcommands create or change an application plan or a package. On either subcommand, you can use the OWNER option to name the owner of the resulting plan or package. Consider these points when naming an owner:

- Any user can name the primary ID or any secondary ID.

2. Dropping a package does not delete all privileges on it if another version of the package still remains in the catalog.

- An ID with the BINDAGENT privilege can name the grantor of that privilege.
- An ID with SYSCTRL or SYSADM authority can name any authorization ID on a BIND command, but not on a REBIND command.

If you omit the OWNER option:
- On BIND, your primary ID becomes the owner.
- On REBIND, the previous owner retains ownership.

Some systems that can bind a package at a DB2 system do not support the OWNER option. When the OWNER option is not supported, the primary authorization ID is always the owner of the package because a secondary ID cannot be named as the owner.

Qualifying unqualified names

A plan or package can contain SQL statements that use unqualified table and view names. For static SQL, the default qualifier for those names is the owner of the plan or package. However, you can use the QUALIFIER option of the BIND command to specify a different qualifier.

When you perform bind operations on plans or packages that contain static SQL, the BINDAGENT privilege and the OWNER and QUALIFIER options give you considerable flexibility.

Example: Suppose that ALPHA has the BINDAGENT privilege from BETA, and BETA has privileges on tables that are owned by GAMMA. ALPHA can bind a plan using OWNER (BETA) and QUALIFIER (GAMMA). ALPHA does not need to have privileges on the tables to bind the plan. However, ALPHA does not have the privilege to execute the plan.

For plans or packages that contain dynamic SQL, DYNAMICRULES behavior determines how DB2 qualifies unqualified object names. See “Authorization for dynamic SQL statements” on page 164 for more information.

For unqualified distinct types, user-defined functions, stored procedures, sequences, and trigger names in dynamic SQL statements, DB2 uses the schema name as the qualifier. DB2 finds the schema name in the CURRENT PATH special register. For static statements, the PATH bind option determines the path that DB2 searches to resolve unqualified distinct types, user-defined functions, stored procedures, sequences, and trigger names.

Exception: ALTER, CREATE, DROP, COMMENT ON, GRANT, and REVOKE statements follow different conventions for assigning qualifiers. For static SQL, you must specify the qualifier for these statements in the QUALIFIER bind option. For dynamic SQL, the qualifier for these statements is the authorization ID of the CURRENT SQLID special register. See Chapter 2 of *DB2 SQL Reference* for more information about unqualified names.

Authorization to execute

The plan or package owner must have authorization to execute all static SQL statements that are embedded in the plan or package. However, you do not need to have the authorization when the plan or package is bound. The objects to which the plan or package refers do not even need to exist at bind time.

A bind operation always checks whether a local object exists and whether the owner has the required privileges on it. Any failure results in a message. However,

you can choose whether the failure prevents the bind operation from completing by using the VALIDATE option on the BIND PLAN and BIND PACKAGE commands. The following values for the VALIDATE option determine how DB2 is to handle existence and authorization errors:

RUN If you choose RUN for the VALIDATE option, the bind succeeds even when existence or authorization errors exist. DB2 checks existence and authorization at run time.

BIND If you choose BIND for the VALIDATE option, the bind fails when existence or authorization errors exist.
Exception: If you use the SQLERROR(CONTINUE) option on the BIND PACKAGE command, the bind succeeds, but the package's SQL statements that have errors cannot execute.

The corresponding existence and authorization checks for remote objects are always made at run time.

Authorization to execute dynamic SQL statements is also checked at run time. Table 46 on page 161 shows which IDs can supply the authorizations that are required for different types of statements.

Applications that use the Resource Recovery Services attachment facility (RRSAF) to connect to DB2 do not require a plan. If the requesting application is an RRSAF application, DB2 follows the rules described in "Checking authorization to execute an RRSAF application without a plan" on page 151 to check authorizations.

Checking authorization at a second DB2 server

Authorization for execution at a second DB2 server (also known as a "double-hop" situation) is a special case of system-directed access when bind option DBPROTOCOL (PRIVATE) is in effect. Figure 10 shows the process for execution at a second DB2 server.

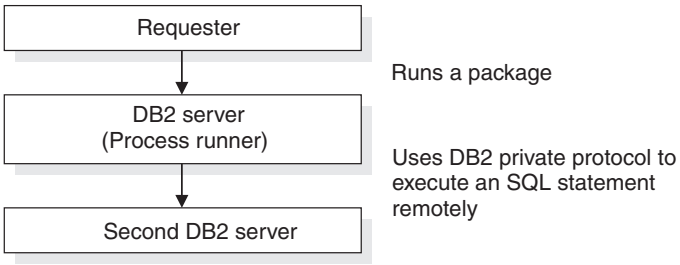


Figure 10. Execution at a second DB2 server

In the figure, a remote requester, either a DB2 UDB for z/OS or some other requesting system, runs a package at the DB2 server. A statement in the package uses an alias or a three-part name to request services from a second DB2 UDB for z/OS server. The ID that is checked for the privileges that are needed to run at the second server can be:

- The owner of the plan that is running at the requester (if the requester is DB2 UDB for z/OS or OS/390)
- The owner of the package that is running at the DB2 server
- The authorization ID of the process that runs the package at the first DB2 server (the "process runner")

In addition, if a remote alias is used in the SQL, the alias must be defined at the requester site. The ID that is used depends on these four factors:

- Whether the requester is DB2 UDB for z/OS or OS/390, or a different system.
- The value of the bind option DYNAMICRULES. See "Authorization for dynamic SQL statements" on page 164 for detailed information about the DYNAMICRULES options.
- Whether the parameter HOPAUTH at the DB2 server site was set to BOTH or RUNNER when the installation job DSNTIJUZ was run. The default value is BOTH.
- Whether the statement that is executed at the second server is static or dynamic SQL.

Hop situation with non-DB2 UDB for z/OS or OS/390 server: Using DBPROTOCOL(DRDA), a three-part name statement can hop to a server other than DB2 UDB for z/OS or OS/390. In this hop situation, only package authorization information is passed to the second server.

A hop is not allowed on a connection that matches the LUWID of another existing DRDA thread. For example, in a hop situation from site A to site B to site C to site A, a hop is not allowed to site A again.

Table 44 shows how these factors determine the ID that must hold the required privileges when bind option DBPROTOCOL (PRIVATE) is in effect.

Table 44. The authorization ID that must hold required privileges for the double-hop situation

Requester	DYNAMICRULES	HOPAUTH	Statement	Authorization ID
DB2 UDB for z/OS	Run behavior (default)[1]	n/a	Static	Plan owner
			Dynamic	Process runner
	Bind behavior[1]	n/a	Either	Plan owner
Different system or RRSAF application without a plan	Run behavior (default)[1]	YES (default)	Static	Package owner
			Dynamic	Process runner
		NO	Either	Process runner
	Bind behavior[1]	n/a	Either	Package owner

Note: [1] If DYNAMICRULES define behavior is in effect, DB2 converts to DYNAMICRULES bind behavior. If DYNAMICRULES invoke behavior is in effect, DB2 converts to DYNAMICRULES run behavior.

Checking authorization to execute an RRSAF application without a plan

RRSAF provides the capability for an application to connect to DB2 and run without a DB2 plan. If an RRSAF application does not have a plan, the following authorization rules are true:

- For the following types of packages, the primary or secondary authorization ID of the process is used for checking authorization to execute the package:
  - A local package
  - A remote package that is on a DB2 UDB for z/OS or OS/390 system and is accessed using DRDA
- At a DB2 UDB for z/OS or OS/390 system, the authorization to execute the DESCRIBE TABLE statement includes checking the primary and secondary authorization IDs.
- For a double hop situation, the authorization ID that must hold the required privileges to execute SQL statements at the second server is determined as if the requester is not a DB2 UDB for z/OS or OS/390 system. Table 44 lists the specific privileges.

Caching authorization IDs for best performance

You can specify that DB2 cache authorization IDs for plans, packages, or routines (user-defined functions and stored procedures). Caching IDs can greatly improve performance, especially when IDs are reused frequently. One cache exists for each plan, one global cache exists for packages, and a global cache exists for routines. The global cache for packages and routines are allocated at DB2 startup. For a data sharing group, each member does its own authorization caching.

Caching IDs for plans: Authorization checking is fastest when the EXECUTE privilege is granted to PUBLIC and, after that, when the plan is reused by an ID that already appears in the cache.

You can set the size of the plan authorization cache by using the BIND PLAN subcommand. For suggestions on setting this cache size, see Part 5 of *DB2 Application Programming and SQL Guide*. The default cache size is specified by an installation option, with an initial default setting of 3072 bytes.

Caching IDs for packages: This performance enhancement provides a run-time benefit for:

- Stored procedures
- Remotely bound packages
- Local packages in a package list in which the plan owner does not have execute authority on the package at bind time, but does at run time
- Local packages that are not explicitly listed in a package list, but are implicitly listed by *collection-id.**, *.*, or **.package-id*

You can set the size of the package authorization cache using the PACKAGE AUTH CACHE field on installation panel DSNTIPP. The default value, 100 KB, is enough storage to support about 690 *collection-id.package-id* entries or *collection-id.** entries.

You can cache more package authorization information by using any of the following strategies:

- Granting package execute authority to *collection.**
- Granting package execute authority to PUBLIC for some packages or collections
- Increasing the size of the cache

The QTPACAUT field in the package accounting trace indicates how often DB2 succeeds at reading package authorization information from the cache.

Caching IDs for routines: The routine authorization cache stores authorization IDs with the EXECUTE privilege on a specific routine. A routine is identified as *schema.routine-name.type*, where the routine name is one of the following names:

- The specific function name for user-defined functions
- The procedure name for stored procedures
- '*' for all routines in the schema

You can set the size of the routine authorization cache by using the ROUTINE AUTH CACHE field on installation panel DSNTIPP. The initial default setting of 100 KB is enough storage to support about 690*schema.routine.type* or *schema.*.type* entries.

You can cache more authorization information about routines by using the following strategies:

- Granting EXECUTE on *schema.**

- Granting routine execute authority to PUBLIC for some or all routines in the schema
- Increasing the size of the cache

Caching and multilevel security: Caching is used with multilevel security with row-level granularity to improve performance. DB2 caches all security labels that are checked (successfully and unsuccessfully) during processing. At commit or rollback, the security labels are removed from the cache. If a security policy that employs multilevel security with row-level granularity requires an immediate change and long-running applications have not committed or rolled back, you might need to cancel the application. For more information about multilevel security with row-level granularity, see "Working with data in a multilevel-secure environment" on page 199.

Controls in the program

Because an ID executes a package or an application plan by running a program, implementing control measures in the program can be useful.

Example: Consider the SQL statement in "Privileges exercised through a plan or a package" on page 148, which permits access to the row of the employee table WHERE EMPNO='000010'. If you replace the value 000010 with a host variable, the program could supply the value of the variable and permit access to various employee numbers. Routines in the program could limit that access to certain IDs, certain times of the day, certain days of the week, or other special circumstances.

Stored procedures provide an alternative to controls in the program. By encapsulating several SQL statements into a single message to the DB2 server, a stored procedure can protect sensitive portions of the application program. Also, stored procedures can include access to non-DB2 resources, as well as DB2.

Recommendation against use of controls in the program

Do not use programs to extend security. Whenever possible, use other techniques, such as stored procedures or views, as a security mechanism. Using programs to extend security has the following drawbacks:

- Program controls are separate from other access controls, can be difficult to implement properly, are difficult to audit, and are relatively simple to bypass.
- Almost any debugging facility can be used to bypass security checks in a program.
- Other programs might use the plan without doing the needed checking.
- Errors in the program checks might allow unauthorized access.
- Because the routines that check security might be quite separate from the SQL statement, the security check could be entirely disabled without requiring a bind operation for a new plan.
- A BIND REPLACE operation for an existing plan does not necessarily revoke the existing EXECUTE privileges on the plan. (Revoking those privileges is the default, but the plan owner has the option to retain them. For packages, the EXECUTE privileges are always retained.)

For those reasons, if the program accesses any sensitive data, the EXECUTE privileges on the plan and on packages are also sensitive. They should be granted only to a carefully planned list of IDs.

Restricting a plan or a package to particular systems

If you use controls in the program, limit the use of a plan or package to the particular systems for which it was designed. DB2 does not ensure that only specific programs are used with a plan, but program-to-plan control can be enforced in IMS and CICS. DB2 provides a consistency check to avoid accidental mismatches between program and plan, but the consistency check is not a security check.

The ENABLE and DISABLE options: You can limit the use of plans and packages by using the ENABLE and DISABLE options on the BIND and REBIND subcommands.

Example: The ENABLE IMS option allows the plan or package to run from any IMS connection. Unless other systems are also named, ENABLE IMS does not allow the plan or package to run from any other type of connection.

Example: DISABLE BATCH prevents a plan or package from running through a batch job, but it allows the plan or package to run from all other types of connection.

You can exercise even finer control with the ENABLE and DISABLE options. You can enable or disable particular IMS connection names, CICS application IDs, requesting locations, and so forth. For details, see the syntax of the BIND and REBIND subcommands in *DB2 Command Reference*.

Privileges required for remote packages

Generally, the privileges that are required for a remote bind (BIND PACKAGE *location.collection*) must be granted at the server location. That is, the ID that owns the package must have all of the privileges that are required to run the package at the server, and BINDADD[3] and CREATE IN privileges at the server.

Exceptions:

- For a BIND COPY operation, the owner must have the COPY privilege at the local DB2 site or subsystem, where the package that is being copied resides.
- If the creator of the package is not the owner, the creator must have SYSCTRL authority or higher, or must have been granted the BINDAGENT privilege by the owner. That authority or privilege is granted at the local DB2.

Binding a plan with a package list (BIND PLAN PKLIST) is done at the local DB2, and bind privileges must be held there. Authorization to execute a package at a remote location is checked at execution time, as follows:

- For DB2 private protocol, the owner of the plan at the requesting DB2 must have EXECUTE privilege for the package at the DB2 server.
- For DRDA, if the server is a DB2 UDB for z/OS subsystem, the authorization ID of the process (primary ID or any secondary ID) must have EXECUTE privilege for the package at the DB2 server.
- If the server is not DB2 UDB for z/OS, the primary authorization ID must have whatever privileges are needed. Check that product's documentation.

3. Or BIND, depending on the installation option BIND NEW PACKAGE.

Access control for user-defined functions and stored procedures

A number of steps are involved in implementing, defining, and invoking user-defined functions and stored procedures, which are also called *routines*. This section explains those steps and the authorizations that they require. The common tasks and required privileges for routines are summarized in Table 45.

Table 45. Common tasks and required privileges for routines

Role	Tasks	Required privileges
Implementer	If SQL is in the routine: codes, precompiles, compiles, and link-edits the program to use as the routine. Binds the program as the routine package. If no SQL is in the routine: codes, compiles, and link-edits the program.	If binding a package, BINDADD system privilege and CREATE IN on the collection.
Definer	Issues a CREATE FUNCTION statement to define a user-defined function or CREATE PROCEDURE statement to define a stored procedure.	CREATEIN privilege on the schema. EXECUTE authority on the routine package when invoked.
Invoker	Invokes a routine from an SQL application.	EXECUTE authority on the routine.

The routine *implementer* typically codes the routine in a program and precompiles the program. If the program contains SQL statements, the implementer binds the DBRM. In general, the authorization ID that binds the DBRM into a package is the package owner. The implementer is the routine package owner. As package owner, the implementer implicitly has EXECUTE authority on the package and has the authority to grant EXECUTE privileges to other users to execute the code within the package.

The implementer grants EXECUTE authority on the routine package to the definer. EXECUTE authority is necessary only if the package contains SQL. For user-defined functions, the definer requires EXECUTE authority on the package. For stored procedures, the EXECUTE privilege on the package is checked for the definer and other IDs. For information about these additional IDs, see the CALL statement in *DB2 SQL Reference*.

The *definer* is the routine owner. The definer issues a CREATE FUNCTION statement to define a user-defined function or a CREATE PROCEDURE statement to define a stored procedure. The definer of a routine is determined as follows:

- If the SQL statement is embedded in an application program, the definer is the authorization ID of the owner of the plan or package.
- If the SQL statement is dynamically prepared, the definer is the SQL authorization ID that is contained in the CURRENT SQLID special register.

The definer grants EXECUTE authority on the routine to the invoker, that is, any user ID that needs to invoke the routine.

The *invoker* invokes the routine from an SQL statement in the invoking plan or package. The invoker for a routine is determined as follows:

- For a static statement, the invoker is the authorization ID of the plan or package owner.

- For a dynamic statement, the invoker depends on DYNAMICRULES behavior. See "Authorization for dynamic SQL statements" on page 164 for a description of the options.

See *DB2 SQL Reference* for more information about the CREATE FUNCTION and CREATE PROCEDURE statements.

Additional authorization for stored procedures

Prior to Version 8, stored procedures were defined to DB2 by inserting rows into catalog table SYSIBM.SYSPROCEDURES. Starting in Version 8, a stored procedure is defined with the CREATE PROCEDURE statement.

The CALL statement invokes a stored procedure. The privileges that are required to execute a stored procedure that is invoked by the CALL statement are described in Chapter 5 of *DB2 SQL Reference*.

Chapter 5 of *DB2 SQL Reference* also describes additional privileges that are required on each package that the stored procedure uses during its execution. The database server determines the privileges that are required and the authorization ID that must have the privileges.

Controlling access to catalog tables for stored procedures

The SYSROUTINES_SRC and SYSROUTINES_OPTS catalog tables contain source code and build options for generated routines that are created by coding tools like the DB2 Stored Procedure Builder. Because a variety of users can use these coding tools, you need to control access to these catalog tables by performing the following steps:

- Determine criteria for limiting each application programmer's access to a subset of the SYSROUTINES_SRC and SYSROUTINES_OPTS rows.
- Create a view for each programmer by using these criteria.
- Grant the SELECT, INSERT, UPDATE, and DELETE privileges on each view to the appropriate programmer.

Example: Suppose that Alan is a programmer and that his ID is A1. Alan is working on a set of stored procedures for project B1. You want to create views that limit Alan's access to specific rows in SYSROUTINES_SRC and SYSROUTINES_OPTS. You can require that Alan use schema names that begin with the characters A1B1. Then you can create views that limit Alan's access to rows where the SCHEMA value begins with A1B1. The following CREATE statement creates a view on SYSROUTINES_SRC:

```
CREATE VIEW A1.B1GRSRC AS
  SELECT SCHEMA, ROUTINENAME, VERSION,
    SEQNO, IBMREQD, CREATESTMT
    FROM SYSIBM.SYSROUTINE_SRC
    WHERE SCHEMA LIKE 'A1B1%'
WITH CHECK OPTION;
```

The following CREATE statement creates a view on SYSROUTINES_OPTS:

```
CREATE VIEW A1.B1GROPTS AS
  SELECT SCHEMA, ROUTINENAME, VERSION,
    BUILDSCHEMA, BUILDNAME, BUILDOWNER, IBMREQD,
    PRECOMPILE_OPTS, COMPILE_OPTS, PRELINK_OPTS,
    LINK_OPTS, BIND_OPTS, SOURCEDSN
    FROM SYSIBM.SYSROUTINE_OPTS
    WHERE SCHEMA LIKE 'A1B1%'
WITH CHECK OPTION;
```

Finally, you can use the following statement to let Alan view or update the appropriate SYSROUTINE_SRC and SYSROUTINE_OPTS rows:

```
GRANT SELECT, INSERT, DELETE, UPDATE
  ON (A1.B1GRSRC,A1.B1GROPTS)
  TO A1;
```

After a set of generated routines goes into production, you can decide to regain control over the routine definitions in SYSROUTINES_SRC and SYSROUTINES_OPTS by revoking the INSERT, DELETE, and UPDATE privileges on the appropriate views. You can allow programmers to keep the SELECT privilege on their views, so that they can use the old rows for reference when they define new generated routines.

Example of roles and authorizations for a routine

This example describes how to get a routine up and running, and how to use and assign the required privileges and authorizations. The routine in the example is an external user-defined function. The steps in the example are divided into the following roles:

- The implementer role is described in "Implementing the user-defined function."
- The definer role is described in"Defining the user-defined function" on page 159.
- The invoker role is described in"Using the user-defined function" on page 160.

Finally, DB2's role in defining authorization IDs is described in "How DB2 determines authorization IDs" on page 161.

Implementing the user-defined function

To implement a user-defined function, the implementer performs the following steps:

1. The implementer codes a program that implements the user-defined function. Assume that the implementer codes the following external user-defined function in C and names the function C_SALARY:

```
/*********************************************************************
* This routine accepts an employee serial number and a percent raise. *
* If the employee is a manager, the raise is not applied.  Otherwise, *
* the new salary is computed, truncated if it exceeds the employee's  *
* manager's salary, and then applied to the database.                 *
*********************************************************************/
 void C_SALARY                        /* main routine               */
 ( char        *employeeSerial        /* in: employee serial no.    */
   decimal     *percentRaise          /* in: percentage raise       */
   decimal     *newSalary,            /* out: employee's new salary */
   short int   *niEmployeeSerial      /* in: indic var, empl ser    */
   short int   *niPercentRaise        /* in: indic var, % raise     */
   short int   *niNewSalary,          /* out: indic var, new salary */
   char        *sqlstate,             /* out: SQLSTATE              */
   char        *fnName,               /* in: family name of function*/
   char        *specificName,         /* in: specific name of func  */
   char        *message               /* out: diagnostic message    */
 )
 {
   EXEC SQL BEGIN DECLARE SECTION;
   char        hvEMPNO-7-;            /* host var for empl serial   */
   decimal     hvSALARY;              /* host var for empl salary   */
   char        hvWORKDEPT-3-;         /* host var for empl dept no. */
   decimal     hvManagerSalary;       /* host var, emp's mgr's salry*/
   EXEC SQL END DECLARE SECTION;

   sqlstate = 0;
   memset( message,0,70 );
   /*******************************************************************
   * Copy the employee's serial into a host variable                  *
   *******************************************************************/
   strcpy( hvEMPNO,employeeSerial );
   /*******************************************************************
   * Get the employee's work department and current salary            *
   *******************************************************************/
   EXEC SQL SELECT  WORKDEPT, SALARY
            INTO :hvWORKDEPT, :hvSALARY
            FROM  EMP
           WHERE  EMPNO = :hvEMPNO;
   /*******************************************************************
   * See if the employee is a manager                                 *
   *******************************************************************/
   EXEC SQL SELECT  DEPTNO
            INTO :hvWORKDEPT
            FROM  DEPT
           WHERE  MGRNO = :hvEMPNO;
   /*******************************************************************
   * If the employee is a manager, do not apply the raise             *
   *******************************************************************/
   if( SQLCODE == 0 )
     {
       newSalary = hvSALARY;
     }
```

Figure 11. Example of a user-defined function (Part 1 of 2)

```
  /*****************************************************************
  * Otherwise, compute and apply the raise such that it does not     *
  * exceed the employee's manager's salary                           *
  *****************************************************************/
  else
    {
      /*************************************************************
      * Get the employee's manager's salary                          *
      *************************************************************/
      EXEC SQL SELECT  SALARY
                 INTO :hvManagerSalary
                 FROM  EMP
                WHERE  EMPNO = (SELECT MGRNO
                                  FROM DSN8610.DEPT
                                 WHERE DEPTNO = :hvWORKDEPT);
      /*************************************************************
      * Compute proposed raise for the employee                      *
      *************************************************************/
      newSalary = hvSALARY * (1 + percentRaise/100);
      /*************************************************************
      * Don't let the proposed raise exceed the manager's salary     *
      *************************************************************/
      if( newSalary > hvManagerSalary
        newSalary = hvManagerSalary;
      /*************************************************************
      * Apply the raise                                              *
      *************************************************************/
      hvSALARY = newSalary;
      EXEC SQL UPDATE  EMP
                 SET  SALARY = :hvSALARY
                WHERE  EMPNO  = :hvEMPNO;
    }

  return;
} /* end C_SALARY */
```

Figure 11. Example of a user-defined function (Part 2 of 2)

The implementer requires the UPDATE privilege on table EMP. Users with the EXECUTE privilege on function C_SALARY do not need the UPDATE privilege on the table.

2. Because this program contains SQL, the implementer performs the following steps:
   a. Precompile the program that implements the user-defined function.
   b. Link-edit the user-defined function with DSNRLI (RRS attachment facility), and name the program's load module C_SALARY.
   c. Bind the DBRM into package MYCOLLID.C_SALARY.

   After performing these steps, the implementer is the *function package owner.*
3. The implementer then grants EXECUTE privilege on the user-defined function package to the definer.

   ```
   GRANT EXECUTE ON PACKAGE MYCOLLID.C_SALARY
    TO definer
   ```

   As package owner, the implementer can grant execute privileges to other users, which allows those users to execute code within the package. For example:

   ```
   GRANT EXECUTE ON PACKAGE MYCOLID.C_SALARY
         TO other_user
   ```

Defining the user-defined function

To define a user-defined function, the definer performs the following steps:

1. The definer executes the following CREATE FUNCTION statement to define the user-defined function salary_change to DB2:

```
CREATE FUNCTION
  SALARY_CHANGE(
    VARCHAR( 6 )
    DECIMAL( 5,2 ) )
  RETURNS
    DECIMAL( 9,2 )
  SPECIFIC schema.SALCHANGE
  LANGUAGE C
  DETERMINISTIC
  MODIFIES SQL DATA
  EXTERNAL NAME C_SALARY
  PARAMETER STYLE DB2SQL
  RETURNS NULL ON NULL CALL
  NO EXTERNAL ACTION
  NO SCRATCHPAD
  NO FINAL CALL
  ALLOW PARALLEL
  NO COLLID
  ASUTIME LIMIT 1
  STAY RESIDENT NO
  PROGRAM TYPE SUB
  WLM ENVIRONMENT WLMENV
  SECURITY DB2
  NO DBINFO;
```

   After executing the CREATE FUNCTION statement, the definer owns the user-defined function. The definer can execute the user-defined function package because the user-defined function package owner, in this case the implementer, granted to the definer the EXECUTE privilege on the package that contains the user-defined function.

2. The definer then grants the EXECUTE privilege on SALARY_CHANGE to all function invokers.

```
GRANT EXECUTE ON FUNCTION SALARY_CHANGE
   TO invoker1, invoker2, invoker3, invoker4
```

Using the user-defined function

To use a user-defined function, the invoker performs the following steps:

1. The invoker codes an application program, named SALARY_ADJ. The application program contains a static SQL statement that invokes the user-defined function SALARY_CHANGE. SALARY_CHANGE gives an employee a 10% raise if the employee is not a manager. The static SQL statement follows:

```
EXEC SQL SELECT  FIRSTNME,
                 LASTNAME
                 SALARY_CHANGE( :hvEMPNO, 10.0 )
           INTO :hvFIRSTNME,
                :hvLASTNAME,
                :hvSALARY
           FROM  EMP
          WHERE  EMPNO = :hvEMPNO;
```

2. The invoker then precompiles, compiles, link-edits, and binds the invoking application's DBRM into the *invoking package* or *invoking plan*. An invoking package or invoking plan is the package or plan that contains the SQL that invokes the user-defined function. After performing these steps, the invoker is the owner of the invoking plan or package.
   Restriction: The invoker must hold the SELECT privilege on the table EMP and the EXECUTE privilege on the function SALARY_CHANGE.

How DB2 determines authorization IDs

In this example, the invoking package SALARY_ADJ contains a static SQL SELECT statement that invokes the user-defined function SALARY_CHANGE. Therefore, DB2 uses the rules for static SQL to determine the authorization ID (invoker) that executes the user-defined function package C_SALARY. For a static statement, the invoker is the authorization ID of the plan or package owner.

- While execution occurs in invoking package SALARY_ADJ, DB2 uses the authorization ID of the invoker (the package owner).

  The invoker requires the EXECUTE privilege on the user-defined function SALARY_CHANGE, which the package SALARY_ADJ invokes. Because the user-defined function definer has the EXECUTE privilege on the user-defined function package C_SALARY, the invoker does not require the explicit EXECUTE privilege.
- When execution changes to the user-defined function package C_SALARY, DB2 uses the authorization ID of the implementer (the package owner). The package owner is the authorization ID with authority to execute all static SQL in the user-defined function package C_SALARY.

For an example of determining authorization IDs for dynamic SQL, see "Example of determining authorization IDs for dynamic SQL statements in routines" on page 168.

Which IDs can exercise which privileges

When a process gains access to DB2, it has a primary authorization ID, an SQL ID, and perhaps one or more secondary authorization IDs. A plan or package also has an owner ID. To perform some actions, one specific ID must hold the required privileges. To perform other actions, any one or several of the IDs must hold the required privileges.

Performance hints: Consider the following performance hints when granting privileges to IDs:

- Limit the number of secondary IDs in your catalog table. For some actions, DB2 searches a catalog table for each ID until it finds a required privilege. A process can have up to 1012 secondary IDs. The more secondary IDs that must be checked, the longer the check takes.
- Ensure that the current SQL ID has the necessary privileges for dynamic SQL. Because the current SQL ID is checked first, the operation is fastest if that ID has all the necessary privileges.

Table 46 and Table 47 on page 162 summarize, for different actions, which IDs can provide the necessary privileges. For more specific details about any statement or command, see *DB2 SQL Reference* or *DB2 Command Reference*.

Table 46. Required privileges for basic operations on dynamic SQL statements

Operation	ID	Required privileges
GRANT	Current SQL ID	Any of the following privileges: • The applicable privilege with the grant option • An authority that includes the privilege, with the grant option (not needed for SYSADM or SYSCTRL) • Ownership that implicitly includes the privilege

Table 46. Required privileges for basic operations on dynamic SQL statements (continued)

Operation	ID	Required privileges
REVOKE	Current SQL ID	Must either have granted the privilege that is being revoked, or hold SYSCTRL or SYSADM authority.
CREATE, for unqualified object name	Current SQL ID	Applicable table, database, or schema privilege
Qualify name of object created	ID named as owner	Applicable table or database privilege. If the current SQL ID has SYSADM authority, the qualifier can be any ID at all, and need not have any privilege.
Other dynamic SQL if DYNAMICRULES uses run behavior	All primary IDs and secondary IDs and the current SQL ID together	As required by the statement; see "Composite privileges" on page 171. Unqualified object names are qualified by the value of the special register CURRENT SQLID. See "Authorization for dynamic SQL statements" on page 164.
Other dynamic SQL if DYNAMICRULES uses bind behavior	Plan or package owner	As required by the statement; see "Composite privileges" on page 171. DYNAMICRULES behavior determines how unqualified object names are qualified; see "Authorization for dynamic SQL statements" on page 164.
Other dynamic SQL if DYNAMICRULES uses define behavior	Function or procedure owner	As required by the statement; see "Composite privileges" on page 171. DYNAMICRULES behavior determines how unqualified object names are qualified; see "Authorization for dynamic SQL statements" on page 164.
Other dynamic SQL if DYNAMICRULES uses invoke behavior	ID of the SQL statement that invoked the function or procedure	As required by the statement; see "Composite privileges" on page 171. DYNAMICRULES behavior determines how unqualified object names are qualified; see "Authorization for dynamic SQL statements" on page 164.

Table 47. Required privileges for basic operations on plans and packages

Operation	ID	Required privileges
Execute a plan	Primary ID or any secondary ID	Any of the following privileges: • Ownership of the plan • EXECUTE privilege for the plan • SYSADM authority
Bind embedded SQL statements, for any bind operation	Plan or package owner	Any of the following privileges: • Applicable privileges required by the statements • Authorities that include the privileges • Ownership that implicitly includes the privileges Object names include the value of QUALIFIER, where it applies.

Table 47. Required privileges for basic operations on plans and packages (continued)

Operation	ID	Required privileges
Include package in PKLIST[1]	Plan owner	Any of the following privileges: • Ownership of the package • EXECUTE privilege for the package • PACKADM authority over the package collection • SYSADM authority
BIND a new plan using the default owner or primary authorization ID	Primary ID	BINDADD privilege, or SYSCTRL or SYSADM authority
BIND a new package using the default owner or primary authorization ID	Primary ID	If the value of the field BIND NEW PACKAGE on installation panel DSNTIPP is BIND, any of the following privileges: • BINDADD privilege and CREATE IN privilege for the collection • PACKADM authority for the collection • SYSADM or SYSCTRL authority If BIND NEW PACKAGE is BINDADD, any of the following privileges: • BINDADD privilege and either the CREATE IN or PACKADM privilege for the collection • SYSADM or SYSCTRL authority
BIND REPLACE or REBIND for a plan or package using the default owner or primary authorization ID	Primary ID or any secondary ID	Any of the following privileges: • Ownership of the plan or package • BIND privilege for the plan or package • BINDAGENT from the plan or package owner • PACKADM authority for the collection (for a package only) • SYSADM or SYSCTRL authority See also "Multiple actions in one statement" on page 171.
BIND a new version of a package, with default owner	Primary ID	If BIND NEW PACKAGE is BIND, any of the following privileges: • BIND privilege on the package or collection • BINDADD privilege and CREATE IN privilege for the collection • PACKADM authority for the collection • SYSADM or SYSCTRL authority If BIND NEW PACKAGE is BINDADD, any of the following: • BINDADD privilege and either the CREATE IN or PACKADM privilege for the collection • SYSADM or SYSCTRL authority
FREE or DROP a package[2]	Primary ID or any secondary ID	Any of the following privileges: • Ownership of the package • BINDAGENT from the package owner • PACKADM authority for the collection • SYSADM or SYSCTRL authority

Table 47. Required privileges for basic operations on plans and packages (continued)

Operation	ID	Required privileges
COPY a package	Primary ID or any secondary ID	Any of the following: • Ownership of the package • COPY privilege for the package • BINDAGENT from the package owner • PACKADM authority for the collection • SYSADM or SYSCTRL authority
FREE a plan	Primary ID or any secondary ID	Any of the following privileges: • Ownership of the plan • BIND privilege for the plan • BINDAGENT from the plan owner • SYSADM or SYSCTRL authority
Name a new OWNER other than the primary authorization ID for any bind operation	Primary ID or any secondary ID	Any of the following privileges: • New owner is the primary or any secondary ID • BINDAGENT from the new owner • SYSADM or SYSCTRL authority

Notes:

1. A user-defined function, stored procedure, or trigger package does not need to be included in a package list.
2. A trigger package cannot be deleted by FREE PACKAGE or DROP PACKAGE. The DROP TRIGGER statement must be used to delete the trigger package.

Authorization for dynamic SQL statements

This section explains authorization behavior for dynamic SQL statements. The two key factors that influence authorization behavior are the DYNAMICRULES value and the run time environment of a package.

The DYNAMICRULES option on the BIND or REBIND command determines what values apply at run time for the following dynamic SQL attributes:

- The authorization ID that is used to check authorization
- The qualifier that is used for unqualified objects
- The source for application programming options that DB2 uses to parse and semantically verify dynamic SQL statements

The DYNAMICRULES option also determines whether dynamic SQL statements can include GRANT, REVOKE, ALTER, CREATE, DROP, and RENAME statements.

In addition to the DYNAMICRULES value, the run-time environment of a package controls how dynamic SQL statements behave at run time. The two possible run-time environments are:

- The package runs as part of a stand-alone program.
- The package runs as a stored procedure or user-defined function package, or runs under a stored procedure or user-defined function.

  A package that runs under a stored procedure or user-defined function is a package whose associated program meets one of the following conditions:
  - The program is called by a stored procedure or user-defined function.
  - The program is in a series of nested calls that start with a stored procedure or user-defined function.

The combination of the DYNAMICRULES value and the run-time environment determine the values for the dynamic SQL attributes. Those attribute values are called the dynamic SQL statement **behaviors**. The four behaviors are:

- "Run behavior"
- "Bind behavior"
- "Define behavior"
- "Invoke behavior" on page 166

The behaviors are summarized in "Common attribute values for bind, define, and invoke behavior" on page 166.

Run behavior

DB2 processes dynamic SQL statements using the standard attribute values for dynamic SQL statements. These attributes are collectively called run behavior and consist of the following attributes:

- DB2 uses the authorization ID of the application process and the current SQL ID to:
  - Check for authorization of dynamic SQL statements
  - Serve as the implicit qualifier of table, view, index, and alias names
- Dynamic SQL statements use the values of application programming options that were specified during installation. The installation option USE FOR DYNAMICRULES has no effect.
- GRANT, REVOKE, CREATE, ALTER, DROP, and RENAME statements can be executed dynamically.

Bind behavior

DB2 processes dynamic SQL statements using bind behavior. Bind behavior consists of the following attributes:

- DB2 uses the authorization ID of the plan or package for authorization checking of dynamic SQL statements.
- Unqualified table, view, index, and alias names in dynamic SQL statements are implicitly qualified with value of the bind option QUALIFIER; if you do not specify QUALIFIER, DB2 uses the authorization ID of the plan or package owner as the implicit qualifier.
- Bind behavior consists of the attribute values that are described in "Common attribute values for bind, define, and invoke behavior" on page 166.

The values of the authorization ID and the qualifier for unqualified objects are the same as those that are used for embedded or static SQL statements.

Define behavior

When the package is run as or under a stored procedure or user-defined function package, DB2 processes dynamic SQL statements using define behavior. Define behavior consists of the following attribute values:

- DB2 uses the authorization ID of the user-defined function or the stored procedure owner for authorization checking of dynamic SQL statements in the application package.
- The default qualifier for unqualified objects is the user-defined function or the stored procedure owner.
- Define behavior consists of the attribute values that are described in "Common attribute values for bind, define, and invoke behavior" on page 166.

When the package is run as a stand-alone program, DB2 processes dynamic SQL statements using bind behavior or run behavior, depending on the DYNAMICRULES value specified.

Invoke behavior

When the package is run as or under a stored procedure or user-defined function package, DB2 processes dynamic SQL statements using invoke behavior. Invoke behavior consists of the following attribute values:

- DB2 uses the authorization ID of the user-defined function or the stored procedure invoker for authorization checking of dynamic SQL statements in the application package.

  If the invoker is the primary authorization ID of the process or the current SQL ID, the following rules apply:

  – The ID of the invoker is checked for the required authorization.

  – Secondary authorization IDs are also checked if they are needed for the required authorization.
- The default qualifier for unqualified objects is the user-defined function or the stored procedure invoker.
- Invoke behavior consists of the attribute values that are described in "Common attribute values for bind, define, and invoke behavior."

When the package is run as a stand-alone program, DB2 processes dynamic SQL statements using bind behavior or run behavior, depending on the DYNAMICRULES specified value.

Common attribute values for bind, define, and invoke behavior

The following attribute values apply to dynamic SQL statements in plans or packages that have bind, define, or invoke behavior:

- You can execute the statement SET CURRENT SQLID in a package or plan that is bound with any DYNAMICRULES value. However, DB2 does not use the current SQL ID as the authorization ID for dynamic SQL statements.

  DB2 always uses the current SQL ID as the qualifier for the EXPLAIN output PLAN_TABLE.
- If the value of installation option USE FOR DYNAMICRULES is YES, DB2 uses the application programming default values that were specified during installation to parse and semantically verify dynamic SQL statements. If the value of USE for DYNAMICRULES is NO, DB2 uses the precompiler options to parse and semantically verify dynamic SQL statements. For a list of the application programming defaults that USE FOR DYNAMICRULES affects, see Part 5 of *DB2 Application Programming and SQL Guide*.
- GRANT, REVOKE, CREATE, ALTER, DROP, and RENAME statements cannot be executed dynamically.

Table 48 shows DYNAMICRULES values, run-time environments, and the dynamic SQL behaviors that they yield.

Table 48. How DYNAMICRULES and the run-time environment determine dynamic SQL statement behavior

	Behavior of dynamic SQL statements	
DYNAMICRULES value	**Stand-alone program environment**	**User-defined function or stored procedure environment**
BIND	Bind behavior	Bind behavior
RUN	Run behavior	Run behavior

Table 48. How DYNAMICRULES and the run-time environment determine dynamic SQL statement behavior (continued)

	Behavior of dynamic SQL statements	
DYNAMICRULES value	**Stand-alone program environment**	**User-defined function or stored procedure environment**
DEFINEBIND	Bind behavior	Define behavior
DEFINERUN	Run behavior	Define behavior
INVOKEBIND	Bind behavior	Invoke behavior
INVOKERUN	Run behavior	Invoke behavior

Note: The BIND and RUN values can be specified for packages and plans. The other values can be specified only for packages.

Table 49 shows the dynamic SQL attribute values for each type of dynamic SQL behavior.

Table 49. Definitions of dynamic SQL statement behaviors

	Setting for dynamic SQL attributes			
Dynamic SQL attribute	**Bind behavior**	**Run behavior**	**Define behavior**	**Invoke behavior**
Authorization ID	Plan or package owner	Current SQLID	User-defined function or stored procedure owner	Authorization ID of invoker [1]
Default qualifier for unqualified objects	Bind OWNER or QUALIFIER value	Current SQLID	User-defined function or stored procedure owner	Authorization ID of invoker
CURRENT SQLID [2]	Not applicable	Applies	Not applicable	Not applicable
Source for application programming options	Determined by DSNHDECP parameter DYNRULS [3]	Install panel DSNTIPF	Determined by DSNHDECP parameter DYNRULS [3]	Determined by DSNHDECP parameter DYNRULS [3]
Can execute GRANT, REVOKE, CREATE, ALTER, DROP, RENAME?	No	Yes	No	No

Notes:

1. If the invoker is the primary authorization ID of the process or the current SQL ID, the following rules apply:
   - The ID of the invoker is checked for the required authorization.
   - Secondary authorization IDs are also checked if they are needed for the required authorization.
2. DB2 uses the current SQL ID as the authorization ID for dynamic SQL statements only for plans and packages that have DYNAMICRULES run behavior. For the other dynamic SQL behaviors, DB2 uses the authorization ID that is associated with each dynamic SQL behavior, as shown in this table.

   The initial current SQL ID is independent of the dynamic SQL behavior. For stand-alone programs, the current SQL ID is initialized to the primary authorization ID. See *DB2 Application Programming and SQL Guide* for information about initialization of current SQL ID for user-defined functions and stored procedures.

   You can execute the SET CURRENT SQLID statement to change the current SQL ID for packages with any dynamic SQL behavior, but DB2 uses the current SQL ID only for plans and packages with run behavior.
3. The value of DSNHDECP parameter DYNRULS, which you specify in field USE FOR DYNAMICRULES in installation panel DSNTIPF, determines whether DB2 uses the precompiler options or the application programming defaults for dynamic SQL statements. See Part 5 of *DB2 Application Programming and SQL Guide* for more information.

Example of determining authorization IDs for dynamic SQL statements in routines

Suppose that A is a stored procedure and C is a program that is neither a user-defined function nor a stored procedure. Also suppose that subroutine B is called by both stored procedure A and program C. Subroutine B, which is invoked by a language call, is neither a user-defined function nor a stored procedure. AP is the package that is associated with stored procedure A, and BP is the package that is associated with subroutine B. A, B, and C execute as shown in Figure 12.

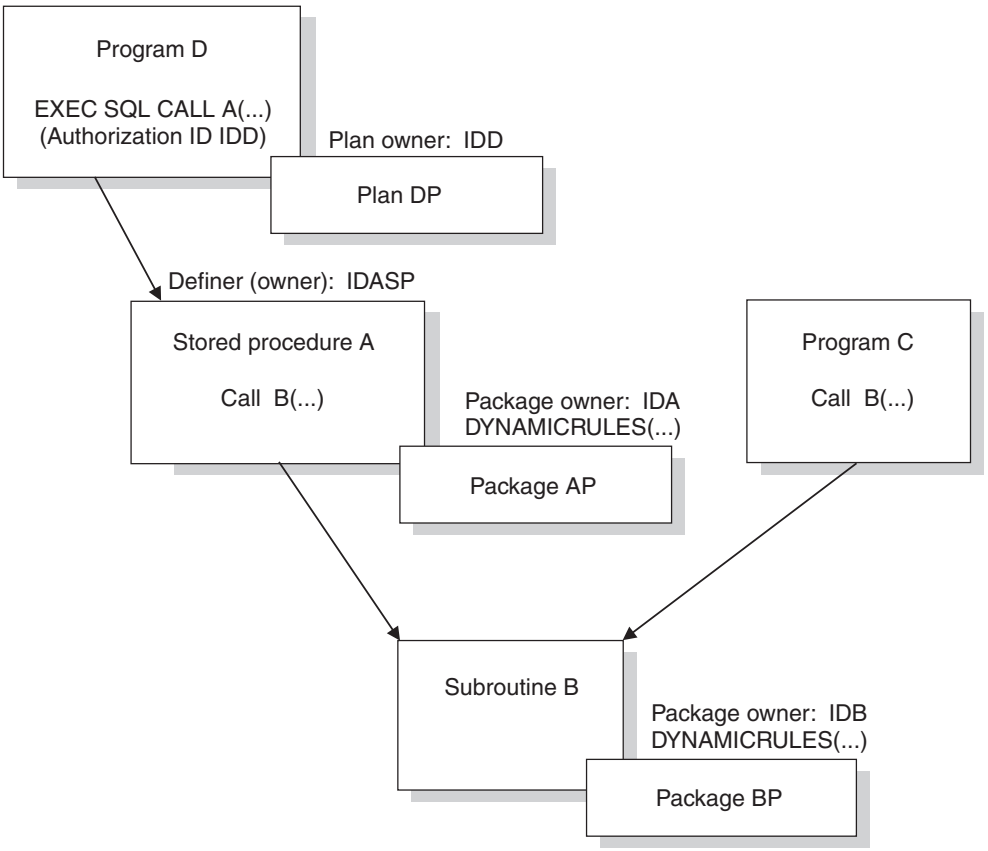


Figure 12. Authorization for dynamic SQL statements in programs and routines

Stored procedure A was defined by IDASP and is therefore owned by IDASP. The stored procedure package AP was bound by IDA and is therefore owned by IDA. Package BP was bound by IDB and is therefore owned by IDB. The authorization ID under which EXEC SQL CALL A runs is IDD, the owner of plan DP.

The authorization ID under which dynamic SQL statements in package AP run is determined in the following way:

- If package AP uses DYNAMICRULES bind behavior, the authorization ID for dynamic SQL statements in package AP is IDA, the owner of package AP.
- If package AP uses DYNAMICRULES run behavior, the authorization ID for dynamic SQL statements in package AP is the value of CURRENT SQLID when the statements execute.
- If package AP uses DYNAMICRULES define behavior, the authorization ID for dynamic SQL statements in package AP is IDASP, the definer (owner) of stored procedure A.

- If package AP uses DYNAMICRULES invoke behavior, the authorization ID for dynamic SQL statements in package AP is IDD, the invoker of stored procedure A.

The authorization ID under which dynamic SQL statements in package BP run is determined in the following way:

- If package BP uses DYNAMICRULES bind behavior, the authorization ID for dynamic SQL statements in package BP is IDB, the owner of package BP.
- If package BP uses DYNAMICRULES run behavior, the authorization ID for dynamic SQL statements in package BP is the value of CURRENT SQLID when the statements execute.
- If package BP uses DYNAMICRULES define behavior:
  - When subroutine B is called by stored procedure A, the authorization ID for dynamic SQL statements in package BP is IDASP, the definer of stored procedure A.
  - When subroutine B is called by program C:
    - If package BP uses the DYNAMICRULES option DEFINERUN, DB2 executes package BP using DYNAMICRULES run behavior, which means that the authorization ID for dynamic SQL statements in package BP is the value of CURRENT SQLID when the statements execute.
    - If package BP uses the DYNAMICRULES option DEFINEBIND, DB2 executes package BP using DYNAMICRULES bind behavior, which means that the authorization ID for dynamic SQL statements in package BP is IDB, the owner of package BP.
- If package BP uses DYNAMICRULES invoke behavior:
  - When subroutine B is called by stored procedure A, the authorization ID for dynamic SQL statements in package BP is IDD, the authorization ID under which EXEC SQL CALL A executed.
  - When subroutine B is called by program C:
    - If package BP uses the DYNAMICRULES option INVOKERUN, DB2 executes package BP using DYNAMICRULES run behavior, which means that the authorization ID for dynamic SQL statements in package BP is the value of CURRENT SQLID when the statements execute.
    - If package BP uses the DYNAMICRULES option INVOKEBIND, DB2 executes package BP using DYNAMICRULES bind behavior, which means that the authorization ID for dynamic SQL statements in package BP is IDB, the owner of package BP.

Now suppose that B is a user-defined function, as shown in Figure 13 on page 170.

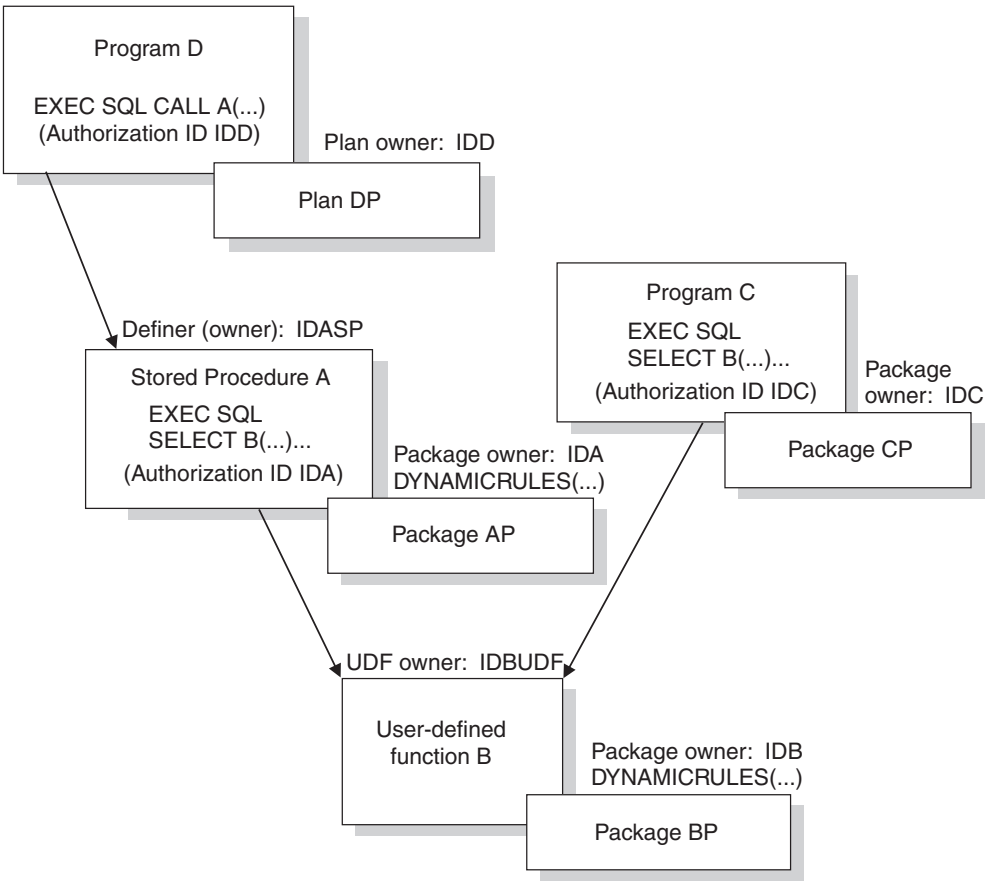


Figure 13. Authorization for dynamic SQL statements in programs and nested routines

User-defined function B was defined by IDBUDF and is therefore owned by ID IDBUDF. Stored procedure A invokes user-defined function B under authorization ID IDA. Program C invokes user-defined function B under authorization ID IDC. In both cases, the invoking SQL statement (EXEC SQL SELECT B) is static.

The authorization ID under which dynamic SQL statements in package BP run is determined in the following way:

- If package BP uses DYNAMICRULES bind behavior, the authorization ID for dynamic SQL statements in package BP is IDB, the owner of package BP.
- If package BP uses DYNAMICRULES run behavior, the authorization ID for dynamic SQL statements in package BP is the value of CURRENT SQLID when the statements execute.
- If package BP uses DYNAMICRULES define behavior, the authorization ID for dynamic SQL statements in package BP is IDBUDF, the definer of user-defined function B.
- If package BP uses DYNAMICRULES invoke behavior:
  - When user-defined function B is invoked by stored procedure A, the authorization ID for dynamic SQL statements in package BP is IDA, the authorization ID under which B is invoked in stored procedure A.
  - When user-defined function B is invoked by program C, the authorization ID for dynamic SQL statements in package BP is IDC, the owner of package CP, and is the authorization ID under which B is invoked in program C.

Simplifying authorization

You can simplify authorization in several ways. However, you should ensure that you do not violate any of the authorization standards at your installation. Consider the following strategies to simplify authorization:

- Have the implementer bind the user-defined function package using DYNAMICRULES define behavior. With this behavior in effect, DB2 only needs to check the definer's ID to execute dynamic SQL statements in the routine. Otherwise, DB2 needs to check the many different IDs that invoke the user-defined function.
- If you have many different routines, group those routines into schemas. Then grant EXECUTE on the routines in the schema to the appropriate users. Users have execute authority on any functions that you add to that schema.

  Example: To grant the EXECUTE privilege on a schema to PUBLIC, issue the following statement:

  ```
  GRANT EXECUTE ON FUNCTION schemaname.* TO PUBLIC;
  ```

Composite privileges

An SQL statement can name more than one object. For example, a SELECT operation can join two or more tables, or an INSERT can use a subquery. Those operations require privileges on all of the tables that are involved in the statement. However, you might be able to issue such a statement dynamically even though one of your IDs alone does not have all the required privileges.

If DYNAMICRULES run behavior is in effect when the dynamic statement is prepared and your primary ID or any of your secondary IDs have all the needed privileges, the statement is validated. However, if you embed the same statement in a host program and try to bind it into a plan or package, the validation fails. The validation also fails for the dynamic statement if DYNAMICRULES bind, define, or invoke behavior is in effect when you issue the dynamic statement. In each case, all the required privileges must be held by the single authorization ID, determined by DYNAMICRULES behavior.

Multiple actions in one statement

A REBIND or FREE subcommand can name more than one plan or package. If no owner is named, the set of privileges associated with the primary ID and the secondary IDs must include the BIND privilege for each object.

Example: Suppose that a user with an ID of FREDDY has the BIND privilege on plan P1 and that a user with an ID of REUBEN has the BIND privilege on plan P2. Assume that someone with FREDDY and REUBEN as secondary authorization IDs issues the following command:

```
REBIND PLAN(P1,P2)
```

P1 and P2 are successfully rebound, even though neither the FREDDY ID nor the REUBEN ID has the BIND privilege for both plans.

Matching job titles with privileges

The names of some authorities suggest job titles. For example, you might expect a system administrator to have SYSADM authority. But not all organizations divide job responsibilities in the same way. Table 50 on page 172 lists some other common job titles, the tasks that usually go with them, and the DB2 authorities or privileges that are needed to carry out those tasks.

Table 50. Some common jobs, tasks, and required privileges

Job title	Tasks	Required privileges
System Operator	Issues commands to: • Start and stop DB2 • Control traces • Display databases and threads • Recover indoubt threads • Start, stop, and display routines	SYSOPR authority
System Administrator	Performs emergency backup, with access to all data.	SYSADM authority
Security Administrator	Authorizes other users, for some or all levels below.	SYSCTRL authority
Database Administrator	Designs, creates, loads, reorganizes, and monitors databases, tables, and other objects.	• DBADM authority over a database • Use of storage groups and buffer pools
System Programmer	• Installs a DB2 subsystem. • Recovers the DB2 catalog. • Repairs data.	Installation SYSADM, which is assigned when DB2 is installed. (Consider securing the password for an ID with this authority so that the authority is available only when needed.)
Application Programmer	• Develops and tests DB2 application programs. • Creates tables of test data.	• BIND on existing plans or packages, or BINDADD • CREATE IN on some collections • Privileges on some objects • CREATETAB on some database, with a default table space provided
Production Binder	Binds, rebinds, and frees application packages and plans	Secondary ID or RACF group with BINDADD, CREATE IN privileges required by application packages and plans
Package Administrator	Manages collections and the packages in them, and delegates the responsibilities.	PACKADM authority
User Analyst	Defines the data requirements for an application program, by examining the DB2 catalog.	• SELECT on the SYSTABLES, SYSCOLUMNS, and SYSVIEWS catalog tables • CREATETMTAB system privilege to create temporary tables
Program End User	Executes an application program.	EXECUTE for the application plan
Information Center Consultant	• Defines the data requirements for a query user. • Provides the data by creating tables and views, loading tables, and granting access.	• DBADM authority over some databases • SELECT on the SYSTABLES, SYSCOLUMNS, and SYSVIEWS catalog tables

Table 50. Some common jobs, tasks, and required privileges (continued)

Job title	Tasks	Required privileges
Query User	• Issues SQL statements to retrieve, add, or change data. • Saves results as tables or in global temporary tables.	• SELECT, INSERT, UPDATE, DELETE on some tables and views • CREATETAB, to create tables in other than the default database • CREATETMTAB system privilege to create temporary tables • SELECT on SYSTABLES, SYSCOLUMNS, or views thereof. QMF provides the views.

Examples of granting and revoking privileges

The SQL GRANT statement lets you grant privileges explicitly. The REVOKE statement lets you take them away. You can revoke only privileges that have been specifically granted. You cannot use either statement unless authorization checking was enabled when DB2 was installed.

You can grant and revoke privileges to and from a single ID, or you can name several IDs on one GRANT or REVOKE statement. Additionally, you can grant privileges to the ID PUBLIC. When you grant privileges to PUBLIC, the privileges become available to all IDs at the local DB2, including the owner IDs of packages that are bound from a remote location.

When you grant any privilege to PUBLIC, DB2 catalog tables record the grantee of the privilege as PUBLIC. Implicit table privileges are also granted to PUBLIC for declared temporary tables. Because PUBLIC is a special identifier that is used by DB2 internally, you should not use PUBLIC as a primary ID or secondary ID. When a privilege is revoked from PUBLIC, authorization IDs to which the privilege was specifically granted retain the privilege.

However, when an ID uses PUBLIC privileges to perform actions, those actions and their resulting objects depend on PUBLIC not losing its privileges. If PUBLIC loses a privilege, objects that are created with that privilege might be dropped or invalidated. The following examples demonstrate how certain objects depend on PUBLIC not losing its privileges.

Example: Suppose that Juan has the ID USER1 and that Meg has the ID USER2. Juan creates a table TAB1 and grants ALL PRIVILEGES on it to PUBLIC. Juan does not explicitly grant any privileges on the table to Meg's ID, USER2. Using the PUBLIC privileges, Meg creates a view on TAB1. Because the ID USER2 requires the SELECT privilege on TAB1 to create the view, the view is dropped if PUBLIC loses the privilege.

Example: Suppose that Kim has the ID USER3. Kim binds a plan and names it PLAN1. PLAN1 contains a program that refers to TAB1 from the previous example. PLAN1 is not valid unless all of the proper privileges are held on objects to which the plan refers, including TAB1. Although Kim does not have any privileges on TAB1, Kim can bind the plan by using the PUBLIC privileges on TAB1. However, if PUBLIC loses its privilege, the plan is invalidated.

You can grant privileges in the following ways:

- Grant a specific privilege on one object in a single statement

- Grant a list of privileges
- Grant privileges over a list of objects
- Grant ALL, for all the privileges of accessing a single table, or for all privileges that are associated with a specific package

DB2 ignores duplicate grants and keeps only one record of a grant in the catalog.

Example: Suppose that Susan grants the SELECT privilege on the EMP table to Ray. Then suppose that Susan grants the same privilege to Ray again, without revoking the first grant. When Susan issues the second grant, DB2 ignores it and maintains the record of the first grant in the catalog.

The suppression of duplicate records applies not only to explicit grants, but also to the implicit grants of privileges that are made when a package is created.

Granting privileges to remote users: A query that arrives at your local DB2 through the distributed data facility (DDF) is accompanied by an authorization ID. That ID can go through connection or sign-on processing when it arrives, it can be translated to another value, and it can be associated with secondary authorization IDs. (For the details of all these processes, see "Controlling requests from remote applications" on page 238.)

As the end result of these processes, the remote query is associated with a set of IDs that is known to your local DB2 subsystem. You assign privileges to these IDs in the same way that you assign privileges to IDs that are associated with local queries.

You can grant a table privilege to any ID anywhere that uses DB2 private protocol access to your data, by issuing the following command:

```
GRANT privilege TO PUBLIC AT ALL LOCATIONS;
```

You can grant SELECT, INSERT, UPDATE, and DELETE table privileges.

If you grant a privilege to PUBLIC AT ALL LOCATIONS, the grantee is PUBLIC*. Because PUBLIC* is a special identifier that is used by DB2 internally, you should not use PUBLIC* as a primary or secondary authorization ID. When a privilege is revoked from PUBLIC AT ALL LOCATIONS, authorization IDs to which the privilege was specifically granted still retain the privilege.

Some differences exist in the privileges for a query that uses system-directed access:

- Although the query can use privileges granted TO PUBLIC AT ALL LOCATIONS, it cannot use privileges granted TO PUBLIC.
- The query can exercise only the SELECT, INSERT, UPDATE, and DELETE privileges at the remote location.

These restrictions do not apply to queries that are run by a package that is bound at your local DB2 subsystem. Those queries can use any privilege that is granted to their associated IDs or any privilege that is granted to PUBLIC.

Examples using the GRANT statement

The following examples show how to use different types of GRANT statements. This example does not demonstrate how to design a security plan for critical data. Instead, the example focuses on demonstrating a variety of situations in which you might use the GRANT statement.

Suppose that the Spiffy Computer Company wants to create a database to hold information that is usually posted on hallway bulletin boards. For example, the database might hold notices of upcoming holidays and bowling scores. Because the president of the Spiffy Computer Company is an excellent bowler, she wants everyone in the company to have access to her scores.

To create and maintain the tables and programs that are needed for this application, Spiffy Computer Company develops the security plan shown in Figure 14.

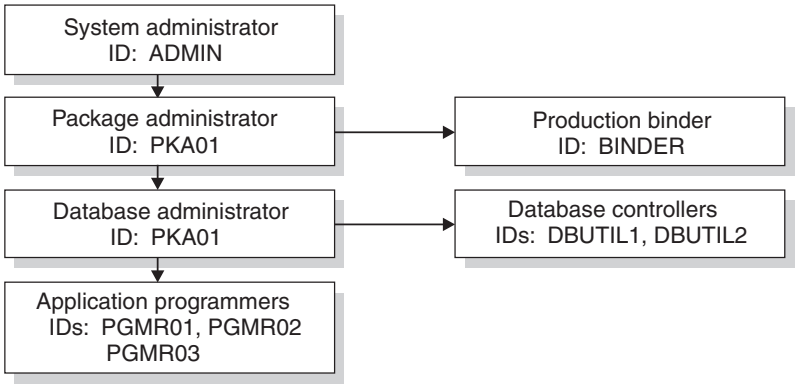
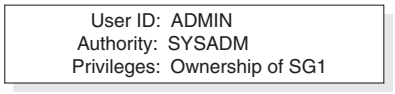


Figure 14. Security plan for the Spiffy Computer Company. Lines connect the grantor of a privilege or authority to the grantee.

Spiffy Computer Company's system of privileges and authorities associates each role with an authorization ID. For example, the System Administrator role has the ADMIN authorization ID.

System administrator's privileges



The system administrator uses the ADMIN authorization ID, which has SYSADM authority, to create a storage group (SG1) and to issue the following statements:

1. `GRANT PACKADM ON COLLECTION BOWLS TO PKA01 WITH GRANT OPTION;`

   This statement grants to PKA01 the CREATE IN privilege on the collection BOWLS and BIND, EXECUTE, and COPY privileges on all packages in the collection. Because ADMIN used the WITH GRANT OPTION clause, PKA01 can grant those privileges to others.
2. `GRANT CREATEDBA TO DBA01;`

   This statement grants to DBA01 the privilege to create a database and to have DBADM authority over that database.
3. `GRANT USE OF STOGROUP SG1 TO DBA01 WITH GRANT OPTION;`

   This statement allows DBA01 to use storage group SG1 and to grant that privilege to others.
4. `GRANT USE OF BUFFERPOOL BP0, BP1 TO DBA01 WITH GRANT OPTION;`

   This statement allows DBA01 to use buffer pools BP0 and BP1 and to grant that privilege to others.

Package administrator's privileges

```
User ID:  PKA01
Authority:  PACKADM over the collection BOWLS
```

The package administrator, PKA01, controls the binding of packages into collections. PKA01 can use the CREATE IN privilege on the collection BOWLS and the BIND, EXECUTE, and COPY privileges on all packages in the collection. PKA01 also has the authority to grant these privileges to others.

Database administrator's privileges

```
User ID:  DBA01
Authority:  DBADM over DB1
Privileges:  CREATEDBA
             Use of SG1 with GRANT
             Use of BP0 and BP1 with GRANT
             Ownership of DB1
```

The database administrator, DBA01, using the CREATEDBA privilege, creates the database DB1. When DBA01 creates DB1, DBA01 automatically has DBADM authority over the database.

Database controller's privileges

```
User ID:  DBUTIL1, DBUTIL2
Authority:  DBCTRL over DB1
```

The database administrator at Spiffy Computer Company wants help with running the COPY and RECOVER utilities. Therefore DBA01 grants DBCTRL authority over database DB1 to DBUTIL1 and DBUTIL2.

To grant DBCTRL authority, the database administrator issues the following statement:

```
GRANT DBCTRL ON DATABASE DB1 TO DBUTIL1, DBUTIL2;
```

Examples with secondary IDs

The Spiffy Computer Company uses RACF (or a similar external security system). Therefore, Spiffy can use secondary authorization IDs to define user groups and associate primary authorization IDs with those user groups. The primary authorization IDs are the RACF user IDs. The secondary authorization IDs are the names of the groups with which the primary IDs are associated.

Spiffy can grant DB2 privileges to primary IDs indirectly, by granting privileges to secondary IDs that are associated with the primary IDs. This approach associates privileges with a *functional ID* rather than an *individual ID*. Functional IDs, also called group IDs, are granted privileges based on the function that certain job roles serve in the system. Multiple primary IDs can be associated with a functional ID and receive the privileges that are granted to that functional ID. In contrast, individual IDs are connected to specific people. Their privileges need to be updated as people join the company, leave the company, or serve different roles within the company. Functional IDs have the following advantages:

- Functional IDs reduce system maintenance because they are more permanent than individual IDs. Individual IDs require frequent updates, but each functional ID can remain in place until Spiffy redesigns its procedures.

  Example: Suppose that Joe retires from the Spiffy Computer Company. Joe is replaced by Mary. If Joe's privileges are associated with functional ID DEPT4,

those privileges are maintained in the system even after Joe's individual ID is removed from the system. When Mary enters the system, she will have all of Joe's privileges after her ID is associated with the functional ID DEPT4.

- Functional IDs reduce the number of grants that are needed because functional IDs often represent groups of individuals.
- Functional IDs reduce the need to revoke privileges and re-create objects when they change ownership.

  Example: Suppose that Bob changes jobs within the Spiffy Computer Company. Bob's individual ID has privileges on many objects in the system and owns three databases. When Bob's job changes, he no longer needs privileges over these objects or ownership of these databases. Because Bob's privileges are associated with his individual ID, a system administrator needs to revoke all of Bob's privileges on objects and drop and re-create Bob's databases with a new owner. If Bob received privileges by association with a functional ID, the system administrator would only need to remove Bob's association with the functional ID.

Application programmers' privileges

The database administrator at Spiffy wants several employees in the Software Support department to create tables in the DB1 database. The database administrator creates DEVGROUP as a RACF group ID for this purpose. To simplify the process, the database administrator decides that each CREATE TABLE statement should implicitly create a unique table space for the table. Hence, DEVGROUP needs the CREATETAB privilege, the CREATETS privilege, the privilege to use the SG1 storage group and, the privilege to use one of the buffer pools, BP0, for the implicitly created table spaces. The following graphic shows this group and their privileges:

```
RACF Group ID:  DEVGROUP
    Privileges:  (All without GRANT)
                 CREATETAB on DB1
                 CREATETS on DB1
                 Use of SG1
                 Use of BP0
```

The database administrator, DBA01, owns database DB1 and has the privileges to use storage group SG1 and buffer pool BP0. The database administrator holds both of these privileges with the GRANT option. The database administrator issues the following statements:

```
1. GRANT CREATETAB, CREATETS ON DATABASE DB1 TO DEVGROUP;
2. GRANT USE OF STOGROUP SG1 TO DEVGROUP;
3. GRANT USE OF BUFFERPOOL BP0 TO DEVGROUP;
```

Because the system and database administrators at Spiffy still need to control the use of those resources, the preceding statements are issued without the GRANT option.

Three programmers in the Software Support department write and test a new program, PROGRAM1. Their IDs are PGMR01, PGMR02, and PGMR03. Each programmer needs to create test tables, use the SG1 storage group, and use one of the buffer pools. All of those resources are controlled by DEVGROUP, which is a RACF group ID.

Therefore, granting privileges over those resources specifically to PGMR01, PGMR02, and PGMR03 is unnecessary. Each ID should be associated with the RACF group DEVGROUP and receive the privileges that are associated with that

functional ID. The following graphic shows the DEVGROUP and its members:

```
RACF group ID:  DEVGROUP
Group members:  PGMR01, PGMR02, PGMR03
```

The security administrator connects as many members as desired to the group DEVGROUP. Each member can exercise all the privileges that are granted to the group ID.

This example assumes that the installed connection and sign-on procedures allow secondary authorization IDs. For examples of RACF commands that connect IDs to RACF groups, and for a description of the connection and sign-on procedures, see Chapter 11, “Controlling access to a DB2 subsystem,” on page 231.

Privileges for binding the plan

Three programmers can now share the tasks that are done by the ID DEVGROUP. Someone creates a test table, DEVGROUP.T1, in database DB1 and loads it with test data. Someone writes a program, PROGRAM1, to display bowling scores that are contained in T1. Someone must bind the plan and packages that accompany the program. Binding requires an additional privilege. ADMIN, who has SYSADM authority, grants the required privilege by issuing the following statement:

```
GRANT BINDADD TO DEVGROUP;
```

With that privilege, any member of the RACF group DEVGROUP can bind plans and packages that are to be owned by DEVGROUP. Any member of the group can rebind a plan or package that is owned by DEVGROUP. The following graphic shows the BINDADD privilege granted to the group:

```
RACF group ID:  DEVGROUP
     Privilege:  BINDADD
```

The Software Support department proceeds to create and test the program.

Moving PROGRAM1 into production

Spiffy has a different set of tables, which contain actual data that is owned by another group ID, PRODCTN. PROGRAM1 was written with unqualified table names. For example, table T1 was referred to as simply T1, not DEVGROUP.T1. The new packages and plan must refer to table PRODCTN.T1. To move the completed program into production, someone must perform the following steps:

- Rebind the application plan with the owner PRODCTN.
- Rebind the packages into the collection BOWLS, again with the owner PRODCTN.

Spiffy gives that job to a production binder with the ID BINDER. BINDER needs privileges to bind a plan or package that DEVGROUP owns, to bind a plan or package with OWNER (PRODCTN), and to add a package to the collection BOWLS. BINDER acquires these abilities through its RACF DEVGROUP and PRODCTN groups. These groups need to have all the necessary privileges.

Suppose that BINDER is a member of the RACF DEVGROUP and PRODCTN groups, and the PRODCTN group has the following privileges:

RACF group ID:	PRODCTN
Privileges:	BINDADD CREATE IN collection BOWLS Privileges on SQL objects referenced in application

BINDER can bind plans and packages for owner PRODCTN because it is a member of the RACF PRODCTN group.

PACKADM, the package administrator for BOWLS, can grant the CREATE privilege with the following statement:

```
GRANT CREATE ON COLLECTION BOWLS TO PRODCTN;
```

With the plan in place, the database administrator at Spiffy wants to make the PROGRAM1 plan available to all employees by issuing the following statement:

```
GRANT EXECUTE ON PLAN PROGRAM1 TO PUBLIC;
```

More than one ID has the authority or privileges that are necessary to issue this statement. For example, ADMIN has SYSADM authority and can grant the EXECUTE privilege. Also, BINDER (or any other members of the PRODCTN group) can set CURRENT SQLID to PRODCTN, which owns PROGRAM1, and issue the statement. When EXECUTE is granted to PUBLIC, other IDs do not need any explicit authority on T1.

Finally, the plan to display bowling scores at Spiffy Computer Company is complete. The production plan, PROGRAM1, is created, and all IDs have the authority to execute the plan.

Spiffy Computer Company's approach to distributed data

Some time after the system and database administrators at Spiffy install their security plan, the president of Spiffy Computer Company tells them that other applications on other systems must connect to the local DB2 subsystem. She wants people at every location to be able to access bowling scores through PROGRAM1 on the local subsystem.

To enable access from all Spiffy locations, administrators perform the following steps:

1. Add a CONNECT statement to the program, naming the location at which table PRODCTN.T1 resides. (In this case, the table and the package reside at only the central location.)
2. Issue the following statement:

   ```
   GRANT CREATE IN COLLECTION BOWLS TO DEVGROUP;
   ```

   PKA01, who has PACKADM authority, grants the required privileges to DEVGROUP by issuing this statement.
3. Bind the SQL statements in PROGRAM1 as a package.

After the steps are completed, the package owner can issue the following statement:

```
GRANT EXECUTE ON PACKAGE PROGRAM1 TO PUBLIC;
```

Any system that is connected to the original DB2 location can then run PROGRAM1 and execute the package by using DRDA access.

Restriction: If the remote system is another DB2, a plan must be bound there that includes the package in its package list.

The preceding steps represent a simplified scenario that focuses on granting appropriate privileges and authorities. In practice, you would also need to consider questions like these:

- Is the performance of a remote query acceptable for this application?
- If other DBMSs are not DB2 subsystems, will the non-SQL portions of PROGRAM1 run in their environments?

Examples using the REVOKE statement

An ID that has granted a privilege can revoke the privilege by issuing the REVOKE statement like the following statement:

```
REVOKE authorization-specification FROM auth-id
```

An ID with SYSADM or SYSCTRL authority can revoke a privilege that has been granted by another ID with the following statement:

```
REVOKE authorization-specification FROM auth-id BY auth-id
```

The BY clause specifies the authorization ID that originally granted the privilege. If two or more grantors grant the same privilege to an ID, executing a single REVOKE statement does not remove the privilege. To remove it, each grant of the privilege must be revoked.

The WITH GRANT OPTION clause of the GRANT statement allows an ID to pass the granted privilege to others. If the privilege is removed from the ID, its deletion can cascade to others, with side effects that are not immediately evident. For example, when a privilege is removed from authorization ID X, it is also removed from any ID to which X granted it, unless that ID also has the privilege from some other source.

Example: Suppose that DBA01 grants DBCTRL authority with the GRANT option on database DB1 to DBUTIL1. Then DBUTIL1 grants the CREATETAB privilege on DB1 to PGMR01. If DBA01 revokes DBCTRL from DBUTIL1, PGMR01 loses the CREATETAB privilege. If PGMR01 also granted the CREATETAB privilege to OPER1 and OPER2, they also lose the privilege.

Example: Suppose that PGMR01 from the preceding example created table T1 while holding the CREATETAB privilege. If PGMR01 loses the CREATETAB privilege, table T1 is not dropped, and the privileges that PGMR01 has as owner of the table are not deleted. Furthermore, the privileges that PGMR01 grants on T1 are not deleted. For example, PGMR01 can grant SELECT on T1 to OPER1 as long as PGMR01 owns of the table. Even when the privilege to create the table is revoked, the table remains, the privilege remains, and OPER1 can still access T1.

Example: Consider the following REVOKE scenario:

1. Grant #1: SYSADM, SA01, grants SELECT on TABLE1 to USER01 with the GRANT option.
2. Grant #2: USER01 grants SELECT on TABLE1 to USER02 with the GRANT option.
3. Grant #3: USER02 grants SELECT on TABLE1 back to SA01.
4. USER02 then revokes SELECT on TABLE1 from SA01.

The cascade REVOKE process of Grant #3 determines if SA01 granted SELECT to anyone else. It locates Grant #1. Because SA01 did not have SELECT from any

4. DB2 does not cascade a revoke of SYSADM authority from the installation SYSADM authorization IDs.

other source, this grant is revoked. The cascade REVOKE process then locates Grant #2 and revokes it for the same reason. In this scenario, the single REVOKE action by USER02 triggers and results in the cascade removal of all the grants even though SA01 has the SYSADM authority. The SYSADM authority is not considered.

Privileges granted from two or more IDs

In addition to the CREATETAB privilege that DBUTIL1 grants to PGRM01, suppose that DBUTIL2 also grants the CREATETAB privilege to PGMR01. The second grant is recorded in the catalog, with its date and time, but it has no other effect until the grant from DBUTIL1 to PGMR01 is revoked. After the first grant is revoked, DB2 must determine which authority PGMR01 used to grant CREATETAB to OPER1. Figure 15 diagrams the situation; arrows represent the granting of the CREATETAB privilege.

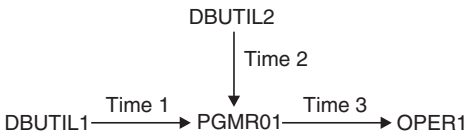


Figure 15. Authorization granted by two or more IDs

As in the diagram, suppose that DBUTIL1 issues the GRANT statement at Time 1 and that DBUTIL2 issues the GRANT statement at Time 2. DBUTIL1 and DBUTIL2 both use the following statement to issue the grant:

```
GRANT CREATETAB ON DATABASE DB1 TO PGMR01 WITH GRANT OPTION;
```

At Time 3, PGMR01 grants the privilege to OPER1 by using the following statement:

```
GRANT CREATETAB ON DATABASE DB1 TO OPER1;
```

After Time 3, DBUTIL1's authority is revoked, along with all of the privileges and authorities that DBUTIL1 granted. However, PGMR01 also has the CREATETAB privilege from DBUTIL2, so PGMR01 does not lose the privilege. The following criteria determine whether OPER1 loses the CREATETAB privilege when DBUTIL1’s authority is revoked:

- If Time 3 *comes after* Time 2, OPER1 does not lose the privilege. The recorded dates and times show that, at Time 3, PGMR01 could have granted the privilege entirely on the basis of the privilege that was granted by DBUTIL2. That privilege was not revoked.
- If Time 3 is *precedes* Time 2, OPER1 does lose the privilege. The recorded dates and times show that, at Time 3, PGMR01 could have granted the privilege only on the basis of the privilege that was granted by DBUTIL1. That privilege was revoked, so the privileges that are dependent on it are also revoked.

Revoking privileges granted by other IDs

An ID with SYSADM or SYSCTRL authority can revoke privileges that are granted by other IDs.

To revoke CREATETAB privileges that are granted to PGMR01 on database DB1 by all IDs, use the following statement:

```
REVOKE CREATETAB ON DATABASE DB1 FROM PGMR01 BY ALL;
```

However, you might want to revoke only privileges that are granted by a certain ID. To revoke privileges that are granted by DBUTIL1 and to leave intact the same privileges if they were granted by any other ID, use the following statement:

```
REVOKE CREATETAB, CREATETS ON DATABASE DB1 FROM PGMR01 BY DBUTIL1;
```

Restricting revocation of privileges

Whether specified or not, the RESTRICT clause of the REVOKE always applies to the following objects:

- User-defined functions
- JARs (Java classes for a routine)
- Stored procedures
- Distinct types
- Sequences

When an attempt is made to revoke a privilege on one of these objects, DB2 determines whether the revokee owns an object that is dependent on the privilege. If such a dependency exists, the REVOKE statement proceeds only if the revokee also holds this privilege from another grantor or holds this privilege indirectly (such as if PUBLIC has this privilege, or if the revokee has SYSADM authority).

Example: Consider the following scenario:

1. UserA creates a user-defined function named UserA.UDFA.
2. UserA grants EXECUTE on UserA.UDFA to UserB.
3. User B then creates a user-defined function UserB.UDFB that is sourced on UserA.UDFA.

At this point, UserA attempts to revoke the EXECUTE privilege on UserA.UDFA from UserB. The revoke succeeds or fails based on the following criteria:

- If UserB has the EXECUTE privilege on UserA.UDFA only from UserA, the revoke fails with an accompanying message that indicates that a dependency on this privilege.
- If UserB has the EXECUTE privilege on UserA.UDFA from another source, directly or indirectly, the EXECUTE privilege that was granted by UserA is revoked successfully.

For distinct types, the following objects that are owned by the revokee can have dependencies:

- A table that has a column that is defined as a distinct type
- A user-defined function that has a parameter that is defined as a distinct type
- A stored procedure that has a parameter that is defined as a distinct type
- A sequence that has a parameter that is defined as a distinct type

For user-defined functions, the following objects that are owned by the revokee can have dependencies:

- Another user-defined function that is sourced on the user-defined function
- A view that uses the user-defined function
- A table that uses the user-defined function in a check constraint or user-defined default clause
- A trigger package that uses the user-defined function

For JARs (Java classes for a routine), the following objects that are owned by the revokee can have dependencies:

- A Java user-defined function that uses a JAR
- A Java stored procedure that uses a JAR

For stored procedures, a trigger package that refers to the stored procedure in a CALL statement can have dependencies.

For sequences, the following objects that are owned by the revokee can have dependencies:

- Triggers that contain NEXT VALUE or PREVIOUS VALUE expressions that specify a sequence
- Inline SQL routines that contain NEXT VALUE or PREVIOUS VALUE expressions that specify a sequence

One way to ensure that the REVOKE statement succeeds is to drop the object that has a dependency on the privilege. To determine which objects are dependent on which privileges before attempting the revoke, use the following SELECT statements.

For a distinct type:

- List all tables that are owned by the revokee USRT002 that contain columns that use the distinct type USRT001.UDT1:

```
SELECT * FROM SYSIBM.SYSCOLUMNS WHERE
  TBCREATOR = 'USRT002'      AND
  TYPESCHEMA = 'USRT001'     AND
  TYPENAME = 'UDT1'          AND
  COLTYPE = 'DISTINCT';
```

- List the user-defined functions that are owned by the revokee USRT002 that contain a parameter that is defined as distinct type USRT001.UDT1:

```
SELECT * FROM SYSIBM.SYSPARMS WHERE
  OWNER = 'USRT002'          AND
  TYPESCHEMA = 'USRT001'     AND
  TYPENAME = 'UDT1'          AND
  ROUTINETYPE = 'F';
```

- List the stored procedures that are owned by the revokee USRT002 that contain a parameter that is defined as distinct type USRT001.UDT1:

```
SELECT * FROM SYSIBM.SYSPARMS WHERE
  OWNER = 'USRT002'          AND
  TYPESCHEMA = 'USRT001'     AND
  TYPENAME = 'UDT1'          AND
  ROUTINETYPE = 'P';
```

- List the sequences that are owned by the revokee USRT002 that contain a parameter that is defined as distinct type USRT001.UDT1:

```
SELECT SYSIBM.SYSSEQUENCES.SCHEMA, SYSIBM.SYSSEQUENCES.NAME
  FROM SYSIBM.SYSSEQUENCES, SYSIBM.SYSDATATYPES WHERE
    SYSIBM.SYSSEQUENCES.DATATYPEID = SYSIBM.SYSDATATYPES.DATATYPEID AND
    SYSIBM.SYSDATATYPES.SCHEMA ='USRT001' AND
    SYSIBM.SYSDATATYPES.NAME   ='UDT1';
```

For a user-defined function:

- List the user-defined functions that are owned by the revokee USRT002 that are sourced on user-defined function USRT001.SPECUDF1:

```
SELECT * FROM SYSIBM.SYSROUTINES WHERE
  OWNER = 'USRT002'            AND
  SOURCESCHEMA = 'USRT001'     AND
  SOURCESPECIFIC = 'SPECUDF1'  AND
  ROUTINETYPE = 'F';
```

- List the views that are owned by the revokee USRT002 that use user-defined function USRT001.SPECUDF1:

```
SELECT * FROM SYSIBM.SYSVIEWDEP WHERE
  DCREATOR = 'USRTOO2'       AND
  BSCHEMA = 'USRT001'        AND
  BNAME = 'SPECUDF1'         AND
  BTYPE = 'F';
```

- List the tables that are owned by the revokee USRT002 that use user-defined function USRT001.A_INTEGER in a check constraint or user-defined default clause:

```
SELECT * FROM SYSIBM.SYSCONSTDEP WHERE
  DTBCREATOR = 'USRT002'      AND
  BSCHEMA = 'USRT001'         AND
  BNAME = 'A_INTEGER'         AND
  BTYPE = 'F';
```

- List the trigger packages that are owned by the revokee USRT002 that use user-defined function USRT001.UDF4:

```
SELECT * FROM SYSIBM.SYSPACKDEP WHERE
  DOWNER = 'USRT002'         AND
  BQUALIFIER = 'USRT001'     AND
  BNAME = 'UDF4'             AND
  BTYPE = 'F';
```

For a JAR (Java class for a routine):

List the routines that are owned by the revokee USRT002 and that use a JAR named USRT001.SPJAR:

```
SELECT * FROM SYSIBM.SYSROUTINES WHERE
  OWNER = 'USRT002'          AND
  JARCHEMA = 'USRT001'       AND
  JAR_ID = 'SPJAR';
```

For a stored procedure that is used in a trigger package:

List the trigger packages that refer to the stored procedure USRT001.WLMOCN2 that is owned by the revokee USRT002:

```
SELECT * FROM SYSIBM.SYSPACKDEP WHERE
  DOWNER = 'USRT002'         AND
  BQUALIFIER = 'USRT001'     AND
  BNAME = 'WLMLOCN2'         AND
  BTYPE = 'O';
```

For a sequence:

- List the sequences that are owned by the revokee USRT002 and that use a trigger named USRT001.SEQ1:

```
SELECT * FROM SYSIBM.SYSPACKDEP WHERE
  BNAME = 'SEQ1'
  BQUALIFIER = 'USRT001'
  BTYPE = 'Q'
  DOWNER = 'USRT002'
  DTYPE = 'T';
```

- List the sequences that are owned by the revokee USRT002 and that use a inline SQL routine named USRT001.SEQ1:

```
SELECT * FROM SYSIBM.SYSSEQUENCESDEP WHERE
  DCREATOR = 'USRT002'
  DTYPE = 'F'
  BNAME = 'SEQ1'
  BSCHEMA = 'USRT001';
```

Other implications of the REVOKE statement

This section discusses some of the indirect implications of issuing a REVOKE statement.

Views and privileges: If a table privilege is revoked from the owner of a view on the table, the corresponding privilege on the view is revoked. The privilege on the view is revoked not only from the owner of the view, but also from all other IDs to which the owner granted the privilege.

If the SELECT privilege on the base table is revoked from the owner of the view, the view is dropped. However, if another grantor granted the SELECT privilege to the view owner before the view was created, the view is not dropped.

Example: Suppose that OPER2 has the SELECT and INSERT privileges on table T1 and creates a view of the table. If the INSERT privilege on T1 is revoked from OPER2, all insert privileges on the view are revoked. If the SELECT privilege on T1 is revoked from OPER2, and if OPER2 did not have the SELECT privilege from another grantor before the view was created, the view is dropped.

If a view uses a user-defined function, the view owner must have the EXECUTE privilege on the function. If the EXECUTE privilege is revoked, the revoke fails because the view is using the privilege and the RESTRICT clause prevents the revoke.

Materialized query tables and privileges:If the SELECT privilege on a source table is revoked from the owner of a materialized query table, the corresponding privilege on the materialized query table is revoked. The SELECT privilege on the materialized query table is revoked not only from the owner of the materialized query table, but also from all other IDs to which the owner granted the SELECT privilege.

If the SELECT privilege on the source table is revoked from the owner of a materialized query table, the materialized query table is dropped. However, if another grantor granted the SELECT privilege to the materialized query table owner before the materialized query table was created, the materialized query table is not dropped.

Example: Suppose that OPER7 has the SELECT privilege on table T1 and creates a materialized query table T2 by selecting from T1. If the SELECT privilege on T1 is revoked from OPER7, and if OPER7 did not have the SELECT privilege from another grantor before T2 was created, T2 is dropped.

If a materialized query table uses a user-defined function, the owner of the materialized query table must have the EXECUTE privilege on the function. If the EXECUTE privilege is revoked, the revoke fails because the materialized query table is using the privilege and the RESTRICT clause prevents the revoke.

Views created by SYSADM: An authorization ID with SYSADM authority can create a view for another authorization ID. In this case, the view could have both a creator and an owner. The owner is automatically given the SELECT privilege on the view. However, the privilege on the base table determines whether the view is dropped.

Example: Suppose that IDADM, with SYSADM authority, creates a view on TABLX with OPER as the owner of the view. OPER now has the SELECT privilege on the

view, but not necessarily any privileges on the base table. If SYSADM is revoked from IDADM, the SELECT privilege on TABLX is gone and the view is dropped.

If one ID creates a view for another ID, the catalog table SYSIBM.SYSTABAUTH needs either one or two rows to record the associated privileges. The number of rows that DB2 uses to record the privilege is determined by the following criteria:

- If IDADM creates a view for OPER when OPER has enough privileges to create the view by itself, only one row is inserted in SYSTABAUTH. The row shows only that OPER granted the required privileges.
- If IDADM creates a view for OPER when OPER does not have enough privileges to create the view by itself, two rows are inserted in SYSTABAUTH. One row shows IDADM as GRANTOR and OPER as GRANTEE of the SELECT privilege. The other row shows any other privileges that OPER might have on the view because of privileges that are held on the base table.

Invalidated and inoperative application plans and packages: If the owner of an application plan or package loses a privilege that is required by the plan or package, and the owner does not have that privilege from another source, DB2 invalidates the plan or package.

Example: Suppose that OPER2 has the SELECT and INSERT privileges on table T1 and creates a plan that uses SELECT, but not INSERT. When privileges are revoked from OPER2, the plan is affected in the following ways:

- If the SELECT privilege is revoked, DB2 invalidates the plan.
- If the INSERT privilege is revoked, the plan is unaffected.
- If the revoked privilege was EXECUTE on a user-defined function, DB2 marks the plan or package **inoperative** instead of invalid.

Implications for caching: If authorization data is cached for a package and an ID loses EXECUTE authority on the package, that ID is removed from the cache.

Similarly, if authorization data is cached for routines, a revoke or cascaded revoke of EXECUTE authority on a routine, or on all routines in a schema (schema.*), from any ID causes the ID to be removed from the cache.

If authorization data is cached for plans, a revoke of EXECUTE authority on the plan from any ID causes the authorization cache to be invalidated.

If an application is caching dynamic SQL statements, and a privilege is revoked that was needed when the statement was originally prepared and cached, that statement is removed from the cache. Subsequent PREPARE requests for that statement do not find it in the cache and therefore execute a full PREPARE. If the plan or package is bound with KEEPDYNAMIC(YES), which means that the application does not need to explicitly re-prepare the statement after a commit operation, you might get an error on an OPEN, DESCRIBE, or EXECUTE of that statement following the next commit operation. The error can occur because a prepare operation is performed implicitly by DB2. If you no longer have sufficient authority for the prepare, the OPEN, DESCRIBE, or EXECUTE request fails.

Revoking SYSADM from installation SYSADM: If you REVOKE SYSADM authority from an ID with installation SYSADM authority, DB2 does not cascade the revoke. Therefore, you can change the ID that holds installation SYSADM authority or delete extraneous IDs with SYSADM authority without cascading the revoke that these processes required.

To change the ID that holds installation SYSADM authority, perform the following steps:

1. Select a new ID that you will grant installation SYSADM authority.
2. Grant SYSADM authority to the ID that you selected.
3. Revoke SYSADM authority from the ID that currently holds installation SYSADM authority.
4. Update the SYSTEM ADMIN 1 or SYSTEM ADMIN 2 field on installation panel DSNTIPP with the new ID that you want to grant installation SYSADM authority.

To delete extraneous IDs with SYSADM authority, perform the following steps:

1. Write down the ID that currently holds installation SYSADM authority.
2. Change the authority of the ID that you want to delete from SYSADM to installation SYSADM. You can change the authority by updating the SYSTEM ADMIN 1 or SYSTEM ADMIN 2 field on installation panel DSNTIPP. Replace the ID that you wrote down in step 1 with the ID that you want to delete.
3. Revoke SYSADM authority from the ID that you want to delete.
4. Change the SYSTEM ADMIN 1 or SYSTEM ADMIN 2 field on installation panel DSNTIPP back to its original value.

For more information on installation panel DSNTIPP, see *DB2 Installation Guide*.

Finding catalog information about privileges

The following catalog tables contain information about the privileges that IDs can hold.

Table 51. Privileges information in DB2 catalog tables

Table name	Records privileges held for
SYSIBM.SYSCOLAUTH	Updating columns
SYSIBM.SYSDBAUTH	Databases
SYSIBM.SYSPLANAUTH	Plans
SYSIBM.SYSPACKAUTH	Packages
SYSIBM.SYSRESAUTH	Buffer pools, storage groups, collections, table spaces, JARs, and distinct types
SYSIBM.SYSROUTINEAUTH	User-defined functions and stored procedures
SYSIBM.SYSSCHEMAAUTH	Schemas
SYSIBM.SYSTABAUTH	Tables and views
SYSIBM.SYSUSERAUTH	System authorities
SYSIBM.SYSSEQUENCEAUTH	Sequences

For descriptions of the columns of each table, see Appendix F of *DB2 SQL Reference*.

Retrieving information in the catalog

You can query the DB2 catalog tables by using SQL SELECT statements. Executing these SQL statements requires appropriate privileges and authorities. You can control access to the catalog by granting and revoking these privileges and authorities. For suggestions about controlling the security of the catalog, see "Creating views of the DB2 catalog tables" on page 191.

Retrieving all DB2 authorization IDs with granted privileges

Some of the catalog tables listed in Table 51 on page 187 include columns named GRANTEE and GRANTEETYPE. If GRANTEETYPE is blank, the value of GRANTEE is an ID that has been granted a privilege. No single catalog table contains information about all privileges. However, to retrieve all IDs with privileges, you can issue SQL code as shown in the following example:

```
SELECT GRANTEE, 'PACKAGE ' FROM SYSIBM.SYSPACKAUTH
    WHERE GRANTEETYPE = ' '
  UNION
SELECT GRANTEE, 'TABLE   ' FROM SYSIBM.SYSTABAUTH
    WHERE GRANTEETYPE = ' '
  UNION
SELECT GRANTEE, 'COLUMN  ' FROM SYSIBM.SYSCOLAUTH
    WHERE GRANTEETYPE = ' '
  UNION
SELECT GRANTEE, 'ROUTINE ' FROM SYSIBM.SYSROUTINEAUTH
    WHERE GRANTEETYPE = ' '
  UNION
SELECT GRANTEE, 'PLAN    ' FROM SYSIBM.SYSPLANAUTH
  UNION
SELECT GRANTEE, 'SYSTEM  ' FROM SYSIBM.SYSUSERAUTH
  UNION
SELECT GRANTEE, 'DATABASE' FROM SYSIBM.SYSDBAUTH
  UNION
SELECT GRANTEE, 'SCHEMA  ' FROM SYSIBM.SYSSCHEMAAUTH
  UNION
SELECT GRANTEE, 'USE     ' FROM SYSIBM.SYSRESAUTH
 UNION
SELECT GRANTEE, 'SEQUENCE' FROM SYSIBM.SYSSEQUENCEAUTH;
```

Periodically, you should compare the list of IDs that is retrieved by this SQL code with the following lists:

- Lists of users from subsystems that connect to DB2 (such as IMS, CICS, and TSO)
- Lists of RACF groups
- Lists of users from other DBMSs that access your DB2

If DB2 lists IDs that do not exist elsewhere, you should revoke their privileges.

Retrieving multiple grants of the same authorization

If several grantors grant the same privilege to the same grantee, the catalog can become cluttered with similar data. This similar data is often unnecessary, and it might cause poor performance.

Example: Suppose that Judy, Kate, and Patti all grant the SELECT privilege on TABLE1 to Chris. If you care that Chris's ID has the privilege but not who granted the privilege, you might consider two of the SELECT grants to be redundant and unnecessary performance liabilities.

However, you might want to maintain information about authorities that are granted from several different IDs, especially when privileges are revoked.

Example: Suppose that the SELECT privilege from the previous example is revoked from Judy. If Chris has the SELECT privilege from only Judy, Chris loses the SELECT privilege. However, Chris retains the SELECT privilege because Kate and Patti also granted the SELECT privilege to Chris. In this case, the similar grants prove not to be redundant.

You can query the catalog to find duplicate grants on objects. If multiple grants clutter your catalog, consider eliminating unnecessary grants. You can use the following SQL statement to retrieve duplicate grants on plans:

```
SELECT GRANTEE, NAME, COUNT(*)
  FROM SYSIBM.SYSPLANAUTH
  GROUP BY GRANTEE, NAME
  HAVING COUNT(*) > 2
  ORDER BY 3 DESC;
```

This statement orders the duplicate grants by frequency, so that you can easily identify the most duplicated grants. Similar statements for other catalog tables can retrieve information about multiple grants on other types of objects.

Retrieving all IDs with DBADM authority

To retrieve all IDs that have DBADM authority, issue the following statement:

```
SELECT DISTINCT GRANTEE
  FROM SYSIBM.SYSDBAUTH
  WHERE DBADMAUTH <>' ' AND GRANTEETYPE = ' ';
```

Retrieving IDs authorized to access a table

To retrieve all IDs that are explicitly authorized to access the sample employee table (DSN8810.EMP in database DSN8D81A), issue the following statement:

```
SELECT DISTINCT GRANTEE FROM SYSIBM.SYSTABAUTH
  WHERE TTNAME = 'EMP' AND TCREATOR = 'DSN8810'
  AND GRANTEETYPE = ' ';
```

To retrieve all IDs that can change the sample employee table (IDs with administrative authorities and IDs to which authority is explicitly granted), issue the following statement:

```
SELECT DISTINCT GRANTEE FROM SYSIBM.SYSTABAUTH
  WHERE TTNAME = 'EMP' AND
        TCREATOR = 'DSN8810' AND
        GRANTEETYPE = ' ' AND
          (ALTERAUTH  <> ' ' OR
           DELETEAUTH <> ' ' OR
           INSERTAUTH <> ' ' OR
           UPDATEAUTH <> ' ')
UNION
SELECT GRANTEE FROM SYSIBM.SYSUSERAUTH
  WHERE SYSADMAUTH <> ' '
UNION
SELECT GRANTEE FROM SYSIBM.SYSDBAUTH
  WHERE DBADMAUTH <> ' ' AND NAME = 'DSN8D81A';
```

To retrieve the columns of DSN8810.EMP for which update privileges have been granted on a specific set of columns, issue the following statement:

```
SELECT DISTINCT COLNAME, GRANTEE, GRANTEETYPE FROM SYSIBM.SYSCOLAUTH
  WHERE CREATOR='DSN8810' AND TNAME='EMP'
  ORDER BY COLNAME;
```

The character in the GRANTEETYPE column shows whether the privileges have been granted to an authorization ID (blank) or are used by an application plan or package (P).

To retrieve the IDs that have been granted the privilege of updating one or more columns of DSN8810.EMP, issue the following statement:

```
SELECT DISTINCT GRANTEE FROM SYSIBM.SYSTABAUTH
  WHERE TTNAME = 'EMP' AND
        TCREATOR='DSN8810' AND
        GRANTEETYPE=' ' AND
        UPDATEAUTH <> ' ';
```

The query returns only the IDs to which update privileges have been specifically granted. It does not return IDs that have the privilege because of SYSADM or DBADM authority. You could include them by forming a union with additional queries, as shown in the following example:

```
SELECT DISTINCT GRANTEE FROM SYSIBM.SYSTABAUTH
  WHERE TTNAME = 'EMP' AND
        TCREATOR = 'DSN8810' AND
        GRANTEETYPE = ' ' AND
           UPDATEAUTH <> ' '
UNION
SELECT GRANTEE FROM SYSIBM.SYSUSERAUTH
  WHERE SYSADMAUTH <> ' '
UNION
SELECT GRANTEE FROM SYSIBM.SYSDBAUTH
  WHERE DBADMAUTH <> ' ' AND NAME = 'DSN8D81A';
```

Retrieving IDs authorized to access a routine

To retrieve the IDs that are authorized to access stored procedure PROCA in schema SCHEMA1, issue the following statement:

```
SELECT DISTINCT GRANTEE FROM SYSIBM.SYSROUTINEAUTH
  WHERE SPECIFICNAME='PROCA' AND
        SCHEMA='SCHEMA1' AND
        GRANTEETYPE=' ' AND
        ROUTINETYPE ='P';
```

You can write a similar statement to retrieve the IDs that are authorized to access a user-defined function. To retrieve the IDs that are authorized to access user-defined function UDFA in schema SCHEMA1, issue the following statement:

```
SELECT DISTINCT GRANTEE FROM SYSIBM.SYSROUTINEAUTH
  WHERE SPECIFICNAME='UDFA' AND
        SCHEMA='SCHEMA1' AND
        GRANTEETYPE=' ' AND
        ROUTINETYPE ='F';
```

Retrieving the tables an ID is authorized to access

To retrieve the list of tables and views that PGMR001 can access, issue the following statement:

```
SELECT DISTINCT TCREATOR, TTNAME FROM SYSIBM.SYSTABAUTH
  WHERE GRANTEE = 'PGMR001' AND GRANTEETYPE =' ';
```

To retrieve the tables, views, and aliases that PGMR001 owns, issue the following statement:

```
SELECT NAME FROM SYSIBM.SYSTABLES
  WHERE CREATOR = 'PGMR001';
```

Retrieving the plans and packages that access a table

To retrieve the names of application plans and packages that refer to table DSN8810.EMP directly, issue the following statement:

```
SELECT DISTINCT GRANTEE FROM SYSIBM.SYSTABAUTH
  WHERE GRANTEETYPE = 'P' AND
        TCREATOR = 'DSN8810'  AND
        TTNAME = 'EMP';
```

The preceding query does not distinguish between plans and packages. To identify a package, use the COLLID column of table SYSTABAUTH, which names the collection in which a package resides and is blank for a plan.

A plan or package can refer to the table indirectly, through a view. To find all views that refer to the table, perform the following steps:

1. Issue the following query:

```
SELECT DISTINCT DNAME FROM SYSIBM.SYSVIEWDEP
  WHERE BTYPE = 'T' AND
        BCREATOR = 'DSN8810'  AND
        BNAME = 'EMP';
```

2. Write down the names of the views that satisfy the query. These values are instances of *DNAME_list*.
3. Find all plans and packages that refer to those views by issuing a series of SQL statements. For each instance of *DNAME_list*, issue the following statement:

```
SELECT DISTINCT GRANTEE FROM SYSIBM.SYSTABAUTH
  WHERE GRANTEETYPE = 'P' AND
        TCREATOR = 'DSN8810'  AND
        TTNAME = DNAME_list;
```

Creating views of the DB2 catalog tables

Only an ID with SYSADM or SYSCTRL authority automatically has the privilege of retrieving data from catalog tables. If you do not want to grant the SELECT privilege on all catalog tables to PUBLIC, consider using views to let each ID retrieve information about its own privileges.

For example, the following view includes the owner and the name of every table on which a user's primary authorization ID has the SELECT privilege:

```
CREATE VIEW MYSELECTS AS
  SELECT TCREATOR, TTNAME FROM SYSIBM.SYSTABAUTH
    WHERE SELECTAUTH <> ' ' AND
          GRANTEETYPE = ' ' AND
          GRANTEE IN (USER, 'PUBLIC', 'PUBLIC*', CURRENT SQLID);
```

The keyword USER in that statement is equal to the value of the primary authorization ID. To include tables that can be read by a secondary ID, set the current SQLID to that secondary ID before querying the view.

To make the view available to every ID, issue the following GRANT statement:

```
GRANT SELECT ON MYSELECTS TO PUBLIC;
```

Similar views can show other privileges. This view shows privileges over columns:

```
CREATE VIEW MYCOLS (OWNER, TNAME, CNAME, REMARKS, LABEL)
  AS SELECT DISTINCT TBCREATOR, TBNAME, NAME, REMARKS, LABEL
  FROM SYSIBM.SYSCOLUMNS, SYSIBM.SYSTABAUTH
    WHERE TCREATOR = TBCREATOR AND
          TTNAME = TBNAME AND
          GRANTEETYPE = '  ' AND
          GRANTEE IN (USER,'PUBLIC',CURRENT SQLID,'PUBLIC*');
```

Multilevel security

> **Important**
> The following information about multilevel security is specific to DB2. It does not describe all aspects of multilevel security. However, this specific information assumes that you have general knowledge of multilevel security. Before implementing multilevel security on your DB2 subsystem, read *z/OS Planning for Multilevel Security and the Common Criteria*.

Introduction to multilevel security

Multilevel security is a security policy that allows you to classify objects and users based on a system of hierarchical security levels and a system of non-hierarchical security categories. Multilevel security provides the capability to prevent unauthorized users from accessing information at a higher classification than their authorization, and prevents users from declassifying information.

Advantages of multilevel security: Multilevel security offers the following advantages:

- Multilevel security enforcement is mandatory and automatic.
- Multilevel security can use methods that are difficult to express through traditional SQL views or queries.
- Multilevel security does not rely on special views or database variables to provide row-level security control.
- Multilevel security controls are consistent and integrated across the system, so that you can avoid defining users and authorizations more than once.
- Multilevel security does not allow users to declassify information.

Using multilevel security, you can define security for DB2 objects and perform other checks, including row-level security checks. Row-level security checks allow you to control which users have authorization to view, modify, or perform other actions on specific rows of data. For complete information about RACF and multilevel security, including defining security labels, and establishing, auditing, and operating a multilevel-secure environment, see *z/OS Planning for Multilevel Security and the Common Criteria*.

Users and objects in multilevel security

In multilevel security, the relationship between users and objects is important. In the context of multilevel security, a user is any entity that requires access to system resources. The term *user* includes human users, but is not limited to people. Examples of users include:

- Human users
- Stored procedures
- Batch jobs

In the context of multilevel security, an object is any system resource to which access must be controlled. Examples of objects include:

- Data sets
- Tables
- Rows
- Commands

Security labels

Multilevel security restricts access to an object or a row based on the security label of the object or row and the security label of the user.

For local connections, the security label of the user is the security label that the user signed on with. It is the security label that is associated with the DB2 primary authorization ID and that is accessed from the RACF ACEE control block.

For TCP/IP connections, the security label of the user can be defined by the security zone. IP addresses are grouped into security zones on the DB2 server. Users that come in on an IP address have the security label that is associated with the security zone that the IP address is grouped under.

For SNA connections, the default security label for the user is used instead of the security label that the user signed on with.

Security labels are based on security levels and security categories. You can use the Common Criteria (COMCRIT) environment's subsystem parameter to require that all tables created in the subsystem are defined with security labels.

Security levels: Along with security categories, hierarchical security levels are used as a basis for mandatory access checking decisions. When you define the security level of an object, you define the degree of sensitivity of that object. Security levels ensure that an object of a certain security level is protected from access by a user of a lower security level.

For information about defining security levels, see *z/OS Security Server RACF Security Administrator's Guide*.

Security categories: Security categories are the non-hierarchical basis for mandatory access checking decisions. When making security decisions, mandatory access control checks whether one set of security categories includes the security categories that are defined in a second set of security categories.

Mandatory access checking

Mandatory access checking evaluates dominance and determines whether to allow certain actions, based on the following rules:

- If the security label of the user dominates the security label of the object, the user can read from the object.
- If the security label of a user and the security label of the object are equivalent, the user can read from and write to the object.
- If the security label of the object dominates the security label of the user or the security labels are incompatible, the user cannot read or write from the object.

Exception: IDs with Install SYSADM authority bypass mandatory access checking at the DB2 object level because actions by Install SYSADM do not invoke the external access control exit routine (DSNX@XAC). However, multilevel security with row-level granularity is enforced for IDs with Install SYSADM authority.

Discretionary access checking: Once the user passes the mandatory access check, a discretionary check follows. The discretionary access check restricts access to objects based on the identity of a user and the groups to which the user belongs. The discretionary access check ensures that the user is identified as having a "need

to know" for the requested resource. The check is discretionary because a user with a certain access permission is capable of passing that permission to any other user.

Dominant, reverse dominant, equivalent, and disjoint security labels

Mandatory access checking is based on the *dominance* relationships between user security labels and object security labels. One security label dominates another security label when both of the following conditions are true:

- The security level that defines the first security label is greater than or equal to the security level that defines the second security label.
- The set of security categories that defines one security label includes the set of security categories that defines the other security label.

Comparisons between user security labels and object security labels can result in four types of relationships:

Dominant

One security label dominates another security label when both of the following conditions are true:

- The security level that defines the first security label is greater than or equal to the security level that defines the second security label.
- The set of security categories that defines the first security label includes the set of security categories that defines the other security label.

Reading data requires that the user security label dominates the data security label.

Reverse dominant

One security label reverse dominates another security label when both of the following conditions are true:

- The security level that defines the first security label is less than or equal to the security level that defines the second security label.
- The set of security categories that defines the first security label is a subset of the security categories that defines the other security label.

Equivalent

One security label is equivalent to another security label when they are the same or have the same level and set of categories. If both dominance and reverse dominance are true for two security labels, they are equivalent. The user security label must be equivalent to the data security label to be able to read and write data without being able to write down.

Disjoint

A security label is disjoint or incompatible with another security label if incompatible security categories cause neither security label to dominate the other security label. Two security labels are disjoint when each of them has at least one category that the other does not have. Disjoint access is not allowed, even when a user is allowed to write down. If a user security label that is disjoint to the data security label issues an INSERT, UPDATE, or LOAD command, DB2 issues an error.

Example: Suppose that the security level "secret" for the security label HIGH is greater than the security level "sensitive" for the security label MEDIUM. Also, suppose that the security label HIGH includes the security categories Project_A, Project_B, and Project_C, and that the security label MEDIUM includes the security

categories Project_A and Project_B. The security label HIGH dominates the security label MEDIUM because both conditions for dominance are true.

Example: Suppose that the security label HIGH includes the security categories Project_A, Project_B, and Project_C, and that the security label MEDIUM includes the security categories Project_A and Project_Z. In this case, the security label HIGH does not dominate the security label MEDIUM because the set of security categories that define the security label HIGH does not contain the security category Project_Z. The security labels are disjoint.

Write-down control

Mandatory access checking prevents users from declassifying information by not allowing a user to write to an object unless the security label of the user and the security label of the object are equivalent. You can override this security feature, known as write-down control, for specific users by granting write-down privilege to those users.

Example: Suppose that user1 has a security label of HIGH and that row_x has a security label of MEDIUM. Because the security label of the user and the security label of the row are not equivalent, user_1 cannot write to row_x. Therefore, write-down control prevents user1 from declassifying the information that is in row_x.

Example: Suppose that user2 has a security label of MEDIUM and that row_x has a security label of MEDIUM. Because the security label of the user and the security label of the row are equivalent, user1 can read from and write to row_x. However, user2 cannot change the security label for row_x unless user2 has write-down privilege. Therefore write-down control prevents user2 from declassifying the information that is in row_x.

Granting write-down privilege: To grant write-down privilege to users, you need to define a profile and then allow users to access the profile.

Example: To grant write down privilege to users, perform the following steps:

1. Define a profile. The following RACF command defines the IRR.WRITEDOWN.BYUSER profile:

   ```
   RDEFINE FACILITY IRR.WRITEDOWN.BYUSER UACC(NONE)
   ```

2. Allow specified users to access the profile. The following RACF command allows a group of users to access the IRR.WRITEDOWN.BYUSER profile:

   ```
   PERMIT IRR.WRITEDOWN.BYUSER ID(USRT051 USRT052 USRT054 USRT056 -
      USRT058 USRT060 USRT062 USRT064 USRT066 USRT068 USRT041) -
      ACCESS(UPDATE) CLASS(FACILITY)
   ```

Implementing multilevel security with DB2

You can implement multilevel security with the following combinations:

DB2 authorization with multilevel security with row-level granularity
In this combination, DB2 grants are used for authorization at the DB2 object level (database, table, and so forth). Multilevel security is implemented only on the row level within DB2.

External access control and multilevel security with row-level granularity
In this combination, external access control (such as the RACF access control module) is used for authorization at the DB2 object level. External access control also uses security labels to perform mandatory access checking on DB2 objects as part of multilevel security. Multilevel security is

also implemented on the row level within DB2. For information about the access control authorization exit, see "Access control authorization exit routine" on page 1061 and *DB2 RACF Access Control Module Guide*.

Implementing multilevel security at the object level

To implement multilevel security with DB2 at the object level, perform the following steps:

1. Define security labels in RACF for all DB2 objects that require mandatory access checking by using the RDEFINE command. Define security labels for the following RACF resource classes:
   - DSNADM (administrative authorities)
   - DSNR (access to DB2 subsystems)
   - MDSNBP and GSNBP (buffer pools)
   - MDSNCL and GDSNCL (collections)
   - MDSNJR and MDSNJR (JAR)
   - MDSNPN and GDSNPN (plans)
   - MDSNSC and GDSNSC (schema)
   - MDSNSG and GDSNSG (storage groups)
   - MDSNSM and GDSNSM (system privileges)
   - MDSNSP and GDSNSP (stored procedures)
   - MDSNSQ and GDSNSQ (sequences)
   - MDSNTB and GDSNTB (tables, views, indexes)
   - MDSNTS and GDSNTS (table spaces)
   - MDSNUF and GDSNUF (user-defined functions)

   You are responsible for ensuring a proper hierarchy of security labels. In general, the security label of an object that is higher in the object hierarchy should dominate the security labels of objects that are lower in the hierarchy. RACF and DB2 do not enforce the hierarchy; they merely enforce the dominance rules that you establish.

   You can use RACF to define security labels for the DB2 objects in the following object hierarchy:
   - Subsystem or data sharing group
     - Database
       - Table space
         - Table
           - Column
           - Row
     - View
     - Storage group
     - Buffer pool
     - Plan
     - Collection
       - Package
     - Schema
       - Stored procedure or user-defined function
       - Java ARchive (JAR)
       - Distinct type
       - Sequence

   The following examples suggest dominance relationships among objects in the DB2 object hierarchy.

   Example: A collection should dominate a package.

   Example: A subsystem should dominate a database. That database should dominate a table space. That table space should dominate a table. That table should dominate a column.

Example: If a view is based on a single table, the table should dominate the view. However, if a view is based on multiple tables, the view should dominate the tables.

Recommendation: Define the security label SYSMULTI for DB2 subsystems that are accessed by users with different security labels and tables that require row-level granularity.

2. Define security labels and associate users with the security labels in RACF. If you are using a TCP/IP connection, you need to define security labels in RACF for the security zones into which IP addresses are grouped. These IP addressed represent remote users.

   Recommendation: Give users with SYSADM, SYSCTRL, and SYSOPR authority the security label of SYSHIGH.

3. Activate the SECLABEL class in RACF. If you want to enforce write-down control, turn on write-down control in RACF.
4. Install the external security access control authorization exit routine (DSNX@XAC), such as the RACF access control module.

Implementing multilevel security with row-level granularity

Many applications need row-level security within the relational database, so that access can be restricted to a specific set of rows. This security control often needs to be mandatory, so that users, including application programmers and database administrators, are unable to bypass the row-level security mechanism. Using mandatory controls with z/OS and RACF provides consistency across the system.

You can implement multilevel security with row-level granularity with or without implementing multilevel security on the object level. If you implement multilevel security on the object level, you must perform step 1 and step 4 in "Implementing multilevel security at the object level" on page 196. If you do not use the access control authorization exit routine or RACF access control, you can use DB2 native authorization control.

Recommendation: Use multilevel security at the object level with multilevel security with row-level granularity. Using RACF with multilevel security provides an independent check at run time and always checks the authorization of a user to the data.

DB2 performs multilevel security with row-level granularity by comparing he security label of the user to the security label of the row that is accessed. Because security labels can be equivalent without being identical, DB2 uses the RACROUTE REQUEST=DIRAUTH macro to make this comparison when the two security labels are not the same. For read operations, such as SELECT, DB2 uses ACCESS=READ. For update operations, DB2 uses ACCESS=READWRITE.

The write-down privilege for multilevel security with row-level granularity has the following properties:

- A user with the write-down privilege can update the security label of a row to any valid value. The user can make this update independent of the user's dominance relationship with the row.
- DB2 requires that a user have the write-down privilege to perform certain utilities.
- If write-down control is not enabled, all users with valid security labels are equivalent to users with the write-down privilege.

Requirement: You must have z/OS Version 1 Release 5 or later to use DB2 authorization with multilevel security with row-level granularity.

Defining multilevel security on tables: You can use multilevel security with row-level checking to control table access by creating or altering a table to have a column with the AS SECURITY LABEL attribute. Tables with multilevel security in effect can be dropped by using the DROP TABLE statement. Users must have a valid security label to execute CREATE TABLE, ALTER TABLE, and DROP TABLE statements on tables with multilevel security enabled. For information about defining security labels and enabling the security label class, see *z/OS Planning for Multilevel Security*.

Indexing the security label column: The performance of tables that you create and alter can suffer if the security label is not included in indexes. The security label column is used whenever a table with multilevel security enabled is accessed. Therefore, the security label column should be included in indexes on the table. If you do not index the security label column, you cannot maintain index-only access.

CREATE TABLE: When a user with a valid security label creates a table, the user can implement row-level security by including a security label column. The security label column can have any name, but it must be defined as CHAR(8) and NOT NULL WITH DEFAULT. It also must be defined with the AS SECURITY LABEL clause.

Example: To create a table that is named TABLEMLS1 and that has row-level security enabled, issue the following statement:

```
CREATE TABLE TABLEMLS1
  (EMPNO      CHAR(6)       NOT NULL,
   EMPNAME    VARCHAR(20)   NOT NULL,
   DEPTNO     VARCHAR(5)
   SECURITY   CHAR(8)       NOT NULL WITH DEFAULT AS SECURITY LABEL,
   PRIMARY KEY (EMPNO)             )
IN DSN8D71A.DSN8S71D;
```

After the user specifies the AS SECURITY LABEL clause on a column, users can indicate the security label for each row by entering values in that column. When a user creates a table and includes a security label column, SYSIBM.SYSTABLES indicates that the table has row-level security enabled. Once a user creates a table with a security label column, the security on the table cannot be disabled. The table must be dropped and recreated to remove this protection.

ALTER TABLE: A user with a valid security label can implement row-level security on an existing table by adding a security label column to the existing table. The security label column can have any name, but it must be defined as CHAR(8) and NOT NULL WITH DEFAULT. It also must be defined with the AS SECURITY LABEL clause.

Example: Suppose that the table EMP does not have row-level security enabled. To alter EMP so that it has row-level security enabled, issue the following statement:

```
ALTER TABLE EMP
  ADD SECURITY   CHAR(8)       NOT NULL WITH DEFAULT AS SECURITY LABEL;
```

After a user specifies the AS SECURITY LABEL clause on a column, row-level security is enabled on the table and cannot be disabled. The security label for existing rows in the table at the time of the alter is the same as the security label of the user that issued the ALTER TABLE statement.

Important: Plans, packages, and dynamic statements are invalidated when a table is altered to add a security label column.

DROP TABLE: When a user with a valid security label drops a table that has row-level security in effect, the system generates an audit record. Row-level security does not affect the success of a DROP statement; the user's privilege on the table determines whether the statement succeeds.

For more information about these SQL statements and multilevel security, see the *DB2 SQL Reference*.

Working with data in a multilevel-secure environment

Multilevel security with row-level checking affects the results of SELECT, INSERT, UPDATE, and DELETE statements and some utilities. For example, row-level checking ensures that DB2 does not return rows that have a HIGH security label to a user that has a LOW security label. Users must have a valid security label to execute SELECT, INSERT, UPDATE, and DELETE statements and some utilities on tables with multilevel security enabled.

In addition to SQL statements and utilities, this section discusses the following topics:

- "Using views to restrict access" on page 205
- "Global temporary tables with multilevel security" on page 205
- "Materialized query tables with multilevel security" on page 205
- "Constraints and multilevel security" on page 206
- "Field procedures, edit procedures, validation procedures, and multilevel security" on page 206
- "Triggers and multilevel security" on page 206

For information about defining security labels and enabling the security label class, see *z/OS Planning for Multilevel Security*.

Using the SELECT statement with multilevel security

When a user with a valid security label selects data from a table or tables with row-level security enabled, DB2 compares the security label of the user to the security label of each row. Results are returned according to the following rules:

- If the security label of the user dominates the security label of the row, DB2 returns the row.
- If the security label of the user does not dominate the security label of the row, DB2 does not return the data from that row, and DB2 does not generate an error report.

Example: Suppose that Alan has a security label of HIGH, Beth has a security label of MEDIUM, and Carlos has a security label of LOW. Suppose that DSN8710.EMP contains the data that is shown in Table 52 and that the SECURITY column has been declared with the AS SECURITY LABEL clause.

Table 52. Sample data from DSN8710.EMP

EMPNO	LASTNAME	WORKDEPT	SECURITY
000010	HAAS	A00	LOW
000190	BROWN	D11	HIGH
000200	JONES	D11	MEDIUM

Table 52. Sample data from DSN8710.EMP (continued)

EMPNO	LASTNAME	WORKDEPT	SECURITY
000210	LUTZ	D11	LOW
000330	LEE	E21	MEDIUM

Now, suppose that Alan, Beth, and Carlos each submit the following SELECT statement:

```
SELECT LASTNAME
  FROM EMP
  ORDER BY LASTNAME;
```

Because Alan has the security label HIGH, he receives the following result:

```
BROWN
HAAS
JONES
LEE
LUTZ
```

Because Beth has the security label MEDIUM, she receives the following result:

```
HAAS
JONES
LEE
LUTZ
```

Beth does not see BROWN in her result set because the row with that information has a security label of HIGH.

Because Carlos has the security label LOW, he receives the following result:

```
HAAS
LUTZ
```

Carlos does not see BROWN, JONES, or LEE in his result set because the rows with that information have security labels that dominate Carlos's security label.

Although Beth and Carlos do not receive the full result set for the query, DB2 does not return an error code to Beth or Carlos.

Using the INSERT statement with multilevel security

When a user with a valid security label inserts data into a table with row-level security enabled, the security label of the row is determined according to the following rules:

- If the user has write-down privilege or write-down control is not enabled, the user can set the security label for the row to any valid security label. If the user does not specify a value for the security label, the security label of the row becomes the same as the security label of the user.
- If the user does not have write-down privilege and write-down control is enabled, the security label of the row becomes the same as the security label of the user.

Example: Suppose that Alan has a security label of HIGH, that Beth has a security label of MEDIUM and write-down privilege defined in RACF, and that Carlos has a security label of LOW. Write-down control is enabled.

Now, suppose that Alan, Beth, and Carlos each submit the following INSERT statement:

```
INSERT INTO DSN8710.EMP(EMPNO, LASTNAME, WORKDEPT, SECURITY)
    VALUES('099990', 'SMITH', 'C01', 'MEDIUM');
```

Because Alan does not have write-down privilege, Alan cannot choose the security label of the row that he inserts. Therefore DB2 ignores the security label of MEDIUM that is specified in the statement. The security label of the row becomes HIGH because Alan's security label is HIGH.

Because Beth has write-down privilege on the table, she can specify the security label of the new row. In this case, the security label of the new row is MEDIUM. If Beth submits a similar INSERT statement that specifies a value of LOW for the security column, the security label for the row becomes LOW.

Because Carlos does not have write-down privilege, Carlos cannot choose the security label of the row that he inserts. Therefore DB2 ignores the security label of MEDIUM that is specified in the statement. The security label of the row becomes LOW because Carlos's security label is LOW.

Considerations for INSERT from a fullselect: For statements that insert the result of a fullselect, DB2 does not return an error code if the fullselect contains a table with a security label column.

Considerations for SELECT...FROM...INSERT statements: For statements that insert rows and select the inserted rows, DB2 avoids returning some error codes. If the fullselect includes a table with a security label column and the object of the insert does not contain a security label column, DB2 does not return an error code.

If the user has write-down privilege or write-down is not in effect, the security label of the user might not dominate the security label of the row. For statements that insert rows and select the inserted rows, the INSERT statement succeeds in this. However, the inserted row is not returned.

Considerations for INSERT with subselect: If you insert data into a table that does not have a security label column, but a subselect in the INSERT statement does include a table with a security label column, row-level checking is performed for the subselect. However, the inserted rows will not be stored with a security label column.

Using the UPDATE statement with multilevel security

When a user with a valid security label updates a table with row-level security enabled, DB2 compares the security label of the user to the security label of the row. The update proceeds according to the following rules:

- If the security label of the user and the security label of the row are equivalent, the row is updated and the value of the security label is determined by whether the user has write-down privilege:
  - If the user has write-down privilege or write-down control is not enabled, the user can set the security label of the row to any valid security label.
  - If the user does not have write-down privilege and write-down control is enabled, the security label of the row is set to the value of the security label of the user.
- If the security label of the user dominates the security label of the row, the result of the UPDATE statement is determined by whether the user has write-down privilege:

- If the user has write-down privilege or write-down control is not enabled, the row is updated and the user can set the security label of the row to any valid security label.
- If the user does not have write-down privilege and write-down control is enabled, the row is not updated.

- If the security label of the row dominates the security label of the user, the row is not updated.

Example: Suppose that Alan has a security label of HIGH and write-down privilege defined in RACF, that Beth has a security label of MEDIUM and write-down privilege defined in RACF, and that Carlos has a security label of LOW. Write-down control is enabled.

Suppose that DSN8710.EMP contains the data that is shown in Table 53 and that the SECURITY column has been declared with the AS SECURITY LABEL clause.

Table 53. Sample data from DSN8710.EMP

EMPNO	LASTNAME	WORKDEPT	SECURITY
000190	BROWN	D11	HIGH
000200	JONES	D11	MEDIUM
000210	LUTZ	D11	LOW

Now, suppose that Alan, Beth, and Carlos each submit the following UPDATE statement:

```
UPDATE DSN8710.EMP
  SET DEPTNO='X55', SECURITY='MEDIUM'
  WHERE DEPTNO='D11';
```

Because Alan has a security label that is equivalent to the security label of the row with HIGH security, the update on that row succeeds. Because Alan has a security label that dominates the rows with security labels of MEDIUM and LOW, his write-down privilege determines whether these rows are updated. Alan has the write-down privilege that is required to set the security label to any value, so the update succeeds for these rows and the security label for all of the rows becomes MEDIUM. The results of Alan's update are shown in Table 54.

Table 54. Sample data from DSN8710.EMP after Alan's update

EMPNO	EMPNAME	DEPTNO	SECURITY
000190	BROWN	X55	MEDIUM
000200	JONES	X55	MEDIUM
000210	LUTZ	X55	MEDIUM

Because the row with the security label of HIGH dominates Beth's security label, the update fails for that row, which causes the entire update to fail.

Because the rows with the security labels of MEDIUM and HIGH dominate Carlos's security label, the update fails for those rows, which causes the entire update to fail.

Recommendation: To avoid failed updates, qualify the rows that you want to update with the following predicate, for the security label column SECLABEL:

```
WHERE SECLABEL=GETVARIABLE('SYSIBM.SECLABEL');
```

Using this predicate avoids failed updates because it ensures that the user's security label is equivalent to the security label of the rows that DB2 attempts to update.

Using the DELETE statement with multilevel security

When a user with a valid security label deletes data from a table with row-level security enabled, DB2 compares the security label of the user to the security label of the row. The delete proceeds according to the following rules:

- If the security label of the user and the security label of the row are equivalent, the row is deleted.
- If the security label of the user dominates the security label of the row, the user's write-down privilege determines the result of the DELETE statement:
  - If the user has write-down privilege or write-down control is not enabled, the row is deleted.
  - If the user does not have write-down privilege and write-down control is enabled, the row is not deleted.
- If the security label of the row dominates the security label of the user, the row is not deleted.

Example: Suppose that Alan has a security label of HIGH, that Beth has a security label of MEDIUM and write-down privilege defined in RACF, and that Carlos has a security label of LOW. Write-down control is enabled.

Suppose that DSN8710.EMP contains the data that is shown in Table 55 and that the SECURITY column has been declared with the AS SECURITY LABEL clause.

Table 55. Sample data from DSN8710.EMP

EMPNO	LASTNAME	WORKDEPT	SECURITY
000190	BROWN	D11	HIGH
000200	JONES	D11	MEDIUM
000210	LUTZ	D11	LOW

Now, suppose that Alan, Beth, and Carlos each submit the following DELETE statement:

```
DELETE FROM DSN8710.EMP
   WHERE DEPTNO='D11';
```

Because Alan has a security label that dominates the rows with security labels of MEDIUM and LOW, his write-down privilege determines whether these rows are deleted. Alan does not have write-down privilege, so the delete fails for these rows. Because Alan has a security label that is equivalent to the security label of the row with HIGH security, the delete on that row succeeds. The results of Alan's delete are shown in Table 56.

Table 56. Sample data from DSN8710.EMP after Alan's delete

EMPNO	EMPNAME	DEPTNO	SECURITY
000200	JONES	D11	MEDIUM
000210	LUTZ	D11	LOW

Because Beth has a security label that dominates the row with a security label of LOW, her write-down privilege determines whether this row is deleted. Beth has write-down privilege, so the delete succeeds for this row. Because Beth has a

security label that is equivalent to the security label of the row with MEDIUM security, the delete succeeds for that row. Because the row with the security label of HIGH dominates Beth's security label, the delete fails for that row. The results of Beth's delete are shown in Table 57.

Table 57. Sample data from DSN8710.EMP after Beth's delete

EMPNO	EMPNAME	DEPTNO	SECURITY
000190	BROWN	D11	HIGH

Because Carlos's security label is LOW, the delete fails for the rows with security labels of MEDIUM and HIGH. Because Carlos has a security label that is equivalent to the security label of the row with LOW security, the delete on that row succeeds. The results of Carlos's delete are shown in Table 58.

Table 58. Sample data from DSN8710.EMP after Carlos's delete

EMPNO	EMPNAME	DEPTNO	SECURITY
000190	BROWN	D11	HIGH
000200	JONES	D11	MEDIUM

Important: Do not omit the WHERE clause from DELETE statements. If you omit the WHERE clause from the DELETE statement, checking occurs for rows that have security labels. This checking behavior might have a negative impact on performance.

For more information about these SQL statements and multilevel security, see the *DB2 SQL Reference*.

Using utilities with multilevel security

You need a valid security label and additional authorizations to run certain LOAD, UNLOAD, and REORG TABLESPACE jobs on tables that have multilevel security enabled. All other utilities check only for authorization to operate on the table space; they do not check for row-level authorization.

LOAD: You must have the write-down privilege to run LOAD REPLACE on a table space that contains a table with multilevel security enabled. In this case, you can specify the values for the security label column.

When you run LOAD RESUME, you must have the write-down privilege to specify values for the security label column. If you run a LOAD RESUME job and do not have the write-down privilege, DB2 assigns your security label as the value for each row in the security label column.

UNLOAD: Additional restrictions apply to UNLOAD jobs on tables that have multilevel security enabled. Each row is unloaded only if the security label of the user dominates the security label of the row. If security label of the user does not dominate the security label of the row, the row is not unloaded and DB2 does not issue an error message.

REORG TABLESPACE: REORG TABLESPACE jobs on tables that have multilevel security enabled have the following restrictions:

- For jobs with the UNLOAD EXTERNAL option, each row is unloaded only if the security label of the user dominates the security label of the row. If the

security label of the user does not dominate the security label of the row, the row is not unloaded and DB2 does not issue an error message.

- For jobs with the DISCARD option, a qualifying row is discarded only if the user has the write-down privilege and the security label of the user dominates the security label of the row.

For more information about utilities and multilevel security, see *DB2 Utility Guide and Reference*.

Using views to restrict access

If you do not want users to see a security label column, create views that do not include the column.

Example: Suppose that the ORDER table has the following columns: ORDERNO, PRODNO, CUSTNO, SECURITY. Suppose that SECURITY is the security label column, and that you do not want users to see the SECURITY column. Use the following statement to create a view that hides the security label column from users:

```
CREATE VIEW V1 AS
  SELECT ORDERNO, PRODNO, CUSTNO FROM ORDER;
```

Alternatively, you can create views that give each user access only to the rows that include that user's security label column. To do that, retrieve the value of the SYSIBM.SECLABEL session variable, and create a view that includes only the rows that match the session variable value.

Example: To allow access only to the rows that match the user's security label, use the following CREATE statement:

```
CREATE VIEW V2 AS SELECT * FROM ORDER
    WHERE SECURITY=GETVARIABLE('SYSIBM.SECLABEL');
```

Global temporary tables with multilevel security

For a declared temporary table with a column definition, no syntax exists to specify a security label on a DECLARE GLOBAL TEMPORARY TABLE statement. An attempt to specify a security label results in an error.

If a DECLARE GLOBAL TEMPORARY TABLE statement uses a fullselect or a LIKE predicate or a CREATE GLOBAL TEMPORARY TABLE statement uses a LIKE predicate, the resulting temporary table can inherit the security label column from the referenced table or view. However, the temporary table **does not** inherit any security attributes on that column. That means that the inherited column in the temporary table is not defined AS SECURITY LABEL. The column in the temporary table is defined as NOT NULL, with no default. Therefore, any statements that insert data in the temporary table must provide a value for the inherited column.

Materialized query tables with multilevel security

Materialized query tables are tables that contain information that is derived and summarized from other tables. If one or more of the source tables for a materialized query table has multilevel security with row-level granularity enabled, some additional rules apply to working with the materialized query table and the source tables. For information about materialized query tables and these multilevel security rules, see "Using multilevel security with materialized query tables" on page 888.

Constraints and multilevel security

Constraints operate in an multilevel-secure environment in the following ways:

- A unique constraint is allowed on a security label column.
- A referential constraint is not allowed on a security label column.
- A check constraint is not allowed on a security label column.

Multilevel security with row-level checking is not enforced when DB2 checks a referential constraint. Although a referential constraint is not allowed for the security label column, DB2 enforces referential constraints for other columns in the table that are not defined with a security label.

Field procedures, edit procedures, validation procedures, and multilevel security

Field procedures, edit procedures, and validation procedures operate in an multilevel-secure environment in the following ways:

- Field procedures and edit procedures are not allowed on a security label column.
- Validation procedures are allowed on a table that is defined with a security label column. When an authorized user with write-down privilege makes an INSERT or UPDATE request for a row, the validation procedure passes the new row with the security label of the user. If the authorized user does not have write-down privilege, the security label of the row remains the same.

Triggers and multilevel security

When a transition table is generated as the result of a trigger, the security label of the table or row from the original table is not inherited by the transition table. Therefore, multilevel security with row-level checking is not enforced for transition tables and transition values.

If an ALTER TABLE statement is used to add a security label column to a table with a trigger on it, the same rules apply to the new security label column that would apply to any column that is added to the table with the trigger on it.

When a BEFORE trigger is activated, the value of the NEW transition variable that corresponds to the security label column is set to the security label of the user if either of the following criteria are met:

- Write-down control is in effect and the user does not have the write-down privilege
- The value of the security label column is not specified

Implementing multilevel security in a distributed environment

SQL statements that originate from remote requesters can participate in a multilevel secure environment if all information on the requester has the same security label and all users of the requester are permitted to that security label. Management of multilevel security in a distributed environment requires physical control of the participating systems and careful management of the network. Managed systems must be prevented from communicating with other systems that do not have equivalent security labels. For more information about configuring a DB2 server to use multilevel security in a distributed environment, see *z/OS Communications Server: IP Configuration Guide*.

TCP/IP support for multilevel security

A Communications Server IP stack that runs in a multilevel secure environment can be configured as either a restricted stack or an unrestricted stack.

Recommendation: Use an unrestricted stack for DB2. An unrestricted stack is configured with an ID that is defined with a security label of SYSMULTI. A single z/OS system can concurrently run a mix of restricted and unrestricted stacks. Unrestricted stacks allow DB2 to use any security label to open sockets.

All users on a TCP/IP connection have the security label that is associated with the IP address that is defined on the server. If a user requires a different security label, the user must enter through an IP address that has that security label associated with it. If you require multiple IP addresses on a remote z/OS server, a workstation, or a gateway, you can configure multiple virtual IP addresses. This strategy can increase the number of security labels that are available on a client.

Remote users that access DB2 by using a TCP/IP network connection use the security label that is associated with the RACF SERVAUTH class profile when the remote user is authenticated. Security labels are assigned to the database access thread when the DB2 server authenticates the remote server by using the RACROUTE REQUEST = VERIFY service.

SNA support for multilevel security

Security labels are assigned to the database access thread when the DB2 server authenticates the remote server by using the RACROUTE REQUEST = VERIFY service. The service establishes a security label to the authorization ID that is associated with the database access thread. For SNA connections, this security label is the default security label that is defined for the remote user.

Data encryption through built-in functions

Prerequisite: To use built-in functions for data encryption and decryption, the Integrated Cryptographic Service Facility must be installed. For more specific information about this prerequisite, see *DB2 Program Directory*.

DB2 provides built-in data encryption functions that you can use to encrypt sensitive data, such as credit card numbers and medical record numbers. When you use data encryption, DB2 requires the correct password to retrieve the data in a decrypted format. If an incorrect password is provided, DB2 does not decrypt the data.

Attention: DB2 cannot decrypt data without the encryption password, and DB2 does not store encryption passwords in an accessible format. If you forget the encryption password, you cannot decrypt the data.

The ENCRYPT keyword encrypts data. The DECRYPT_BIT, DECRYPT_CHAR, and DECRYPT_DB keywords decrypt data. These functions work like other built-in functions. To use these functions on data, the column that holds the data must be properly defined, as described in “Defining columns for encrypted data” on page 208.

You can encrypt data by using two different methods. These methods are described in the following sections:

- “Defining encryption at the column level” on page 208
- “Defining encryption at the value level” on page 210

This section also discusses the following topics:

- “Ensuring accurate predicate evaluation for encrypted data” on page 211
- “Encrypting non-character values” on page 211
- “Performance recommendations for data encryption” on page 212

Important: Built-in encryption functions work for data that is stored within DB2 subsystem and is retrieved from within that same DB2 subsystem. The encryption functions do not work for data that is passed into and out of a DB2 subsystem. This task is handled by DRDA data encryption, and it is separate from built-in data encryption functions.

Defining columns for encrypted data

When data is encrypted, it is stored as a binary data string. Therefore, encrypted data should be stored in columns that are defined as VARCHAR FOR BIT DATA. Columns that hold encrypted data also require additional bytes to hold a header and to reach a multiple of 8 bytes.

Example: Suppose that you have non-encrypted data in a column that is defined as VARCHAR(6). Use the following calculation to determine the column definition for storing the data in encrypted format:

```
Maximum length of non-encrypted data        6 bytes
Number of bytes to the next multiple of 8   2 bytes
24 bytes for encryption key                24 bytes
                                           --------
Encrypted data column length               32 bytes
```

Therefore, define the column for encrypted data as VARCHAR(32) FOR BIT DATA.

If you use a password hint, DB2 requires an additional 32 bytes to store the hint.

Example: Suppose that you have non-encrypted data in a column that is defined as VARCHAR(10). Use the following calculation to determine the column definition for storing the data in encrypted format with a password hint:

```
Maximum length of non-encrypted data       10 bytes
Number of bytes to the next multiple of 8   6 bytes
24 bytes for encryption key                24 bytes
32 bytes for password hint                 32 bytes
                                           --------
Encrypted data column length               72 bytes
```

Therefore, define the column for encrypted data as VARCHAR(72) FOR BIT DATA.

For more information about encryption password hints, see “Using password hints with column-level encryption” on page 210 and “Using password hints with value-level encryption” on page 211.

Defining encryption at the column level

For *column-level encryption*, all encrypted values in a column are encrypted with the same password. Column-level encryption uses the SET ENCRYPTION PASSWORD statement to manage passwords and hints, the ENCRYPT keyword to indicate which data should be encrypted, and the DECRYPT_BIT, DECRYPT_CHAR, or DECRYPT_DB keyword to decrypt data. The following statement and keywords are used with column-level encryption:

SET ENCRYPTION PASSWORD

Sets the password (and optionally sets the password hint) that DB2 holds for encryption and decryption. DB2 holds this value until it is replaced or until the application finishes.

Recommendation: Use host variables instead of literal values for all passwords and password hints. If statements contain literal values for

passwords and password hints, the security of the encrypted data can be compromised in the DB2 catalog and in a trace report.

ENCRYPT
Indicates which column or columns require encryption. DB2 sets the password on the indicated data to the password that DB2 holds at the time a statement with the ENCRYPT keyword is issued.

DECRYPT_BIT, DECRYPT_CHAR, DECRYPT_DB
Checks for the correct password and decrypts data when the data is selected. For more information about the different decryption functions, see *DB2 SQL Reference*.

When encrypted data is selected, DB2 must hold the same password that was held at the time of encryption to decrypt the data. To ensure that DB2 holds the correct password, issue a SET ENCRYPTION PASSWORD statement with the correct password immediately before selecting encrypted data.

Example: Suppose that you need to create an employee table EMP that contains employee ID numbers in encrypted format. Suppose also that you want to set the password for all rows in an encrypted column to the host variable hv_pass. Finally, suppose that you want to select employee ID numbers in decrypted format. Perform the following steps:

1. Create the EMP table with the EMPNO column. The EMPNO column must be defined with the VARCHAR data type, must be defined FOR BIT DATA, and must be long enough to hold the encrypted data. The following statement creates the EMP table:
```
CREATE TABLE EMP (EMPNO VARCHAR(32) FOR BIT DATA);
```
2. Set the encryption password. The following statement sets the encryption password to the host variable :hv_pass:
```
SET ENCRYPTION PASSWORD = :hv_pass;
```
3. Use the ENCRYPT keyword to insert encrypted data into the EMP table by issuing the following statements:
```
INSERT INTO EMP (EMPNO) VALUES(ENCRYPT('47138'));
INSERT INTO EMP (EMPNO) VALUES(ENCRYPT('99514'));
INSERT INTO EMP (EMPNO) VALUES(ENCRYPT('67391'));
```
4. Select the employee ID numbers in decrypted format:
```
SELECT DECRYPT_CHAR(EMPNO) FROM EMP;
```
   If you provide the correct password, DB2 returns the employee ID numbers in decrypted format.

Using column-level encryption with views

You can use column-level encryption in combination with views to create a view that selects decrypted data from a table. To do this, define the view with a decryption function in the defining fullselect. If the correct password is provided when the view is queried, DB2 will return decrypted data.

Example: Suppose that you want to create a view that contains decrypted employee ID numbers from the EMP table.

1. Create a view on the EMP table by using the following statement:
```
CREATE VIEW CLR_EMP (EMPNO) AS SELECT DECRYPT_CHAR(EMPNO) FROM EMP;
```
2. Set the encryption password, so that the fullselect in the view definition can retrieve decrypted data. Use the following statement:
```
SET ENCRYPTION PASSWORD = :hv_pass;
```

3. Select the desired data from the view by using the following statement:

```
SELECT EMPNO FROM CLR_EMP;
```

Using password hints with column-level encryption

DB2 can store encryption password hints to help with forgotten encryption passwords. Each password hint uses 32 bytes in the encrypted column. For column-level encryption, the password hint is set with the SET ENCRYPTION PASSWORD statement. The GETHINT function returns the password hint.

Example: Use the following statement to set the password hint to the host variable hv_hint:

```
SET ENCRYPTION PASSWORD = :hv_pass WITH HINT = :hv_hint;
```

Example: Suppose that the EMPNO column in the EMP table contains encrypted data and that you submitted a password hint when you inserted the data. Suppose that you cannot remember the encryption password for the data. Use the following statement to return the password hint:

```
SELECT GETHINT (EMPNO) FROM EMP;
```

Defining encryption at the value level

When you use value-level encryption, each value in a given column can be encrypted with a different password. You set the password for each value by using the ENCRYPT keyword with the password. The following keywords are used with value-level encryption:

ENCRYPT

Indicates which data requires encryption. Also, encryption passwords, and optionally password hints, are indicated as part of the ENCRYPT keyword for value-level encryption.

Recommendation: Use host variables instead of literal values for all passwords and password hints. If statements contain literal values for passwords and password hints, the security of the encrypted data can be compromised in the DB2 catalog and in a trace report.

DECRYPT_BIT, DECRYPT_CHAR, DECRYPT_DB

Checks for the correct password and decrypts data when the data is selected. For more information about the different decryption functions, see *DB2 SQL Reference*.

Example: Suppose that a web application collects user information about a customer. This information includes the customer name, which is stored in host variable custname; the credit card number, which is stored in a host variable cardnum; and the password for the card number value, which is stored in a host variable userpswd. The application uses the following statement to insert the customer information:

```
INSERT INTO CUSTOMER (CCN, NAME)
  VALUES(ENCRYPT(:cardnum, :userpswd), :custname);
```

Before the application displays the credit card number for a customer, the customer must enter the password. The application retrieves the credit card number by using the following statement:

```
SELECT DECRYPT_CHAR(CCN, :userpswd) FROM CUSTOMER WHERE NAME = :custname;
```

Using password hints with value-level encryption

DB2 can store encryption password hints to help with forgotten encryption passwords. Each password hint uses 32 bytes in the encrypted column. For value-level encryption, the password hint is set with the ENCRYPT keyword. The GETHINT function returns the password hint.

Recommendation: Use host variables instead of literal values for all passwords and password hints. If the statements contain literal values for passwords and password hints, the security of the encrypted data can be compromised in the DB2 catalog and in a trace report.

Example: Suppose that you want the application from the previous example to use a hint to help customers remember their passwords. The application stores the hint in the host variable pswdhint. For this example, assume the values 'Tahoe' for userpswd and 'Ski Holiday' for pswdhint. The application uses the following statement to insert the customer information:

```
INSERT INTO CUSTOMER (CCN, NAME)
  VALUES(ENCRYPT(:cardnum, :userpswd, :pswdhint), :custname);
```

If the customer requests a hint about the password, the following query is used:

```
SELECT GETHINT(CCN) INTO :pswdhint FROM CUSTOMER WHERE NAME = :custname;
```

The value for pswdhint is set to 'Ski Holiday' and returned to the customer. Hopefully the customer can remember the password 'Tahoe' from this hint.

Ensuring accurate predicate evaluation for encrypted data

When data is encrypted, only = and <> predicates provide accurate results. Predicates such as >, <, and LIKE will return inaccurate results for encrypted data.

Example: Suppose that the value 1234 is encrypted as H71G. Also suppose that the value 5678 is encrypted as BF62. If you use a <> predicate to compare these two values in encrypted format, you receive the same result as you will if you compare these two values in decrypted format:

```
Decrypted:  1234 <> 5678    True
Encrypted:  H71G <> BF62    True
```

In both case, they are not equal.

Example: However, if you use a < predicate to compare these values in encrypted format, you receive a different result than you will if you compare these two values in decrypted format:

```
Decrypted:  1234 < 5678    True
Encrypted:  H71G < BF62    False
```

To ensure that predicates such as >, <, and LIKE return accurate results, you must first decrypt the data.

Encrypting non-character values

DB2 supports encryption for numeric and datetime data types indirectly through casting. If non-character data is cast as VARCHAR or CHAR, the data can be encrypted. For more information about casting, see *DB2 SQL Reference*.

Example: Suppose that you need to encrypt timestamp data and retrieve it in decrypted format. Perform the following steps:

1. Create a table to store the encrypted values and set the column-level encryption password by using the following statements:

```
CREATE TABLE ETEMP (C1 VARCHAR(124) FOR BIT DATA);
SET ENCRYPTION PASSWORD :hv_pass;
```

2. Cast, encrypt, and insert the timestamp data by using the following statement:

```
INSERT INTO ETEMP VALUES ENCRYPT(CHAR(CURRENT TIMESTAMP));
```

3. Recast, decrypt, and select the timestamp data by using the following statement:

```
SELECT TIMESTAMP(DECRYPT_CHAR(C1)) FROM ETEMP;
```

Performance recommendations for data encryption

Encryption, by its nature, degrades the performance of most SQL statements. Decryption requires extra processing, and encrypted data requires more space in DB2. If a predicate requires decryption, the predicate is a stage 2 predicate, which can degrade performance. Encrypted data can also impact your database design, which can indirectly impact performance. To minimize performance degradation, use encryption only in cases that require encryption.

Recommendation: Encrypt only a few highly sensitive data elements, such credit card numbers and medical record numbers.

Some data values are poor candidates for encryption. For example, boolean values and other small value sets, such as the integers 1 through 10, are poor candidates for encryption. Because few values are possible, these types of data can be easy to guess even when they are encrypted. In most cases, encryption is not a good security option for this type of data.

Data encryption and indexes: Creating indexes on encrypted data can improve performance in some cases. Exact matches and joins of encrypted data (if both tables use the same encryption key to encrypt the same data) can use the indexes that you create. Because encrypted data is binary data, range checking of encrypted data requires table space scans. Range checking requires all the row values for a column to be decrypted. Therefore, range checking should be avoided, or at least tuned appropriately.

Encryption performance scenario: The following scenario contains a series of examples that demonstrate how to improve performance while working with encrypted data.

Example: Suppose that you must store EMPNO in encrypted form in the EMP table and in the EMPPROJ table. To define tables and indexes for the encrypted data, use the following statements:

```
CREATE TABLE EMP (EMPNO VARCHAR(48) FOR BIT DATA, NAME VARCHAR(48));
CREATE TABLE EMPPROJ(EMPNO VARCHAR(48) FOR BIT DATA, PROJECTNAME VARCHAR(48));
CREATE INDEX IXEMPPRJ ON EMPPROJ(EMPNO);
```

Example: Next, suppose that one employee can work on multiple projects, and that you want to insert employee and project data into the table. To set the encryption password and insert data into the tables, use the following statements:

```
SET ENCRYPTION PASSWORD = :hv_pass;
SELECT INTO :hv_enc_val FROM FINAL TABLE
  (INSERT INTO EMP VALUES (ENCRYPT('A7513'),'Super Prog'));
INSERT INTO EMPPROJ VALUES (:hv_enc_val,'UDDI Project');
INSERT INTO EMPPROJ VALUES (:hv_enc_val,'DB2 UDB Version 10');
```

```
SELECT INTO :hv_enc_val FROM FINAL TABLE
  (INSERT INTO EMP VALUES (ENCRYPT('4NF18'),'Novice Prog'));
INSERT INTO EMPPROJ VALUES (:hv_enc_val,'UDDI Project');
```

You can improve the performance of INSERT statements by avoiding unnecessary repetition of encryption processing. Note how the host variable hv_enc_val is defined in the SELECT INTO statement and then used in subsequent INSERT statements. If you need to insert a large number of rows that contain the same encrypted value, you might find that the repetitive encryption processing degrades performance. However, you can dramatically improve performance by encrypting the data, storing the encrypted data in a host variable, and inserting the host variable.

Example: Next, suppose that you want to find the programmers who are working on the UDDI Project. Consider the following pair of SELECT statements:

- ***Poor performance:*** The following query shows how not to write the query for good performance:

  ```
  SELECT A.NAME, DECRYPT_CHAR(A.EMPNO) FROM EMP A, EMPPROJECT B
    WHERE DECRYPT_CHAR(A.EMPNO) = DECRYPT_CHAR(B.EMPNO) AND
          B.PROJECT ='UDDI Project';
  ```

  Although the preceding query returns the correct results, it decrypts every EMPNO value in the EMP table and every EMPNO value in the EMPPROJ table where PROJECT = 'UDDI Project' to perform the join. For large tables, this unnecessary decryption is a significant performance problem.
- ***Good performance:*** The following query produces the same result as the preceding query, but with significantly better performance. To find the programmers who are working on the UDDI Project, use the following statement :

  ```
  SELECT A.NAME, DECRYPT_CHAR(A.EMPNO) FROM EMP A, EMPPROJ B
    WHERE A.EMPNO = B.EMPNO AND B.PROJECT ='UDDI Project';
  ```

Example: Next, suppose that you want to find the projects that the programmer with employee ID A7513 is working on. Consider the following pair of SELECT statements:

- ***Poor performance:*** The following query requires DB2 to decrypt every EMPNO value in the EMPPROJ table to perform the join:

  ```
  SELECT PROJECTNAME FROM EMPPROJ WHERE DECRYPT_CHAR(EMPNO) = 'A7513';
  ```
- ***Good performance:*** The following query encrypts the literal value in the predicate so that DB2 can compare it to encrypted values that are stored in the EMPNO column without decrypting the whole column. To find the projects that the programmer with employee ID A7513 is working on, use the following statement :

  ```
  SELECT PROJECTNAME FROM EMPPROJ WHERE EMPNO = ENCRYPT('A7513');
  ```

Chapter 10. Controlling access through a closed application

A *closed application* is an application that requires DB2 objects to be managed solely through external interfaces.

Example: Consider an application process that uses DB2 as a repository for changing data. The application process does more than write to and read from a fixed set of tables. The application process also creates, alters, and drops DB2 objects to deal with new data formats. If an application is not closed, a database administrator has the privileges to do these operations at any time. However, in a closed application, the operations must be performed through a specific application. The application is "closed" because it requires exclusive control over data definition statements for some set of objects.

If you install *data definition control support* on the DSNTIPZ installation panel, you can control how specific plans or package collections can use data definition statements. Data definition control support does not replace existing authorization checks; it imposes additional checks. Table 59 lists the specific statements that are controlled by using data definition control support.

Table 59. Statements that are controlled by data definition control support

Object	CREATE statement	ALTER statement	DROP statement
Alias	`CREATE ALIAS`		`DROP ALIAS`
Database	`CREATE DATABASE`	`ALTER DATABASE`	`DROP DATABASE`
Index	`CREATE INDEX`	`ALTER INDEX`	`DROP INDEX`
Storage group	`CREATE STOGROUP`	`ALTER STOGROUP`	`DROP STOGROUP`
Synonym	`CREATE SYNONYM`		`DROP SYNONYM`
Table	`CREATE TABLE`	`ALTER TABLE`	`DROP TABLE`
Table space	`CREATE TABLESPACE`	`ALTER TABLESPACE`	`DROP TABLESPACE`
View	`CREATE VIEW`		`DROP VIEW`

Note: Data definition control support also controls COMMENT statements and LABEL statements.

The statements in Table 59 are a subset of statements that are referred to as "data definition language." In this chapter, data definition language refers only to this subset of statements. For information about how to impose several degrees of control over data definition in applications and objects, see "Controlling data definition" on page 218.

This chapter provides information on the following topics:

- "Registration tables" on page 216 describes the columns of the ART and ORT.
- "Controlling data definition" on page 218 describes various methods for controlling data definition.
- "Managing the registration tables and their indexes" on page 226 desrcibes how to create and manage the ART and ORT.

© Copyright IBM Corp. 1982, 2008

Registration tables

If you use data definition control support, you must create and maintain the *application registration table* (ART) and the *object registration table* (ORT). You register plans and package collections in the ART. You register the objects that are associated with the plans and collections in the ORT. DB2 consults these two registration tables before accepting a data definition statement from a process. If the registration tables indicate that the process is not allowed to create, alter, or drop a particular object, DB2 does not allow it.

"Columns of the ART" and "Columns of the ORT" on page 217 describe the columns of the two registration tables. For more information about maintaining the ART and the ORT, see "Managing the registration tables and their indexes" on page 226.

Columns of the ART

Table 60 shows the columns of the ART. For information about creating the ART, see "Creating the tables and indexes" on page 226.

Table 60. Columns of the ART

Column name	Description
APPLIDENT	Indicates the collection-ID of the package that executes the data definition language. If no package exists, it indicates the name of the plan that executes the data definition language.
APPLIDENTTYPE	Indicates the type of application identifier.
APPLICATIONDESC[1]	Optional data. Provides a more meaningful description of each application than the eight-byte APPLIDENT column can contain.
DEFAULTAPPL	Indicates whether all data definition language should be accepted from this application.
QUALIFIEROK	Indicates whether the application can supply a missing name part for objects that are named in the ORT. Applies only if REQUIRE FULL NAMES = NO.
CREATOR[1, 2]	Optional data. Indicates the authorization ID that created the row.
CREATETIMESTAMP[1]	Optional data. Indicates when a row was created. If you use CURRENT TIMESTAMP, DB2 automatically enters the value of CURRENT TIMESTAMP When you load or insert a row.
CHANGER[1, 2]	Optional data. Indicates the authorization ID that last changed the row.
CHANGETIMESTAMP[1]	Optional data. Indicates when a row was changed. If you use CURRENT TIMESTAMP, DB2 automatically enters the value of CURRENT TIMESTAMP When you update a row.

Table 60. Columns of the ART (continued)

Column name	Description

Notes:

1. Optional columns are for administrator use. DB2 does not use these columns.
2. Because the CREATOR and CHANGER columns are CHAR(26), they are large enough for a three-part authorization ID. Separate each 8-byte part of the ID with a period in byte 9 and in byte 18. If you enter only the primary authorization ID, consider entering it right-justified in the field (that is, preceded by 18 blanks).

Columns of the ORT

Table 61 shows the columns of the ORT. For information about creating the ORT, see "Creating the tables and indexes" on page 226.

Table 61. Columns of the ORT

Column name	Description
QUALIFIER	Indicates the object name qualifier.
NAME	Indicates the unqualified object name.
TYPE	Indicates the type of object.
APPLMATCHREQ	Indicates whether an application that names this object must match the one that is named in the APPLIDENT column.
APPLIDENT	Collection-ID of the plan or package that executes the data definition language.
APPLIDENTTYPE	Indicates the type of application identifier.
APPLICATIONDESC[1]	Optional data. Provides a more meaningful description of each application than the eight-byte APPLIDENT column can contain.
CREATOR[1, 2]	Optional data. Indicates the authorization ID that created the row.
CREATETIMESTAMP[1]	Optional data. Indicates when a row was created. If you use CURRENT TIMESTAMP, DB2 automatically enters the value of CURRENT TIMESTAMP When you load or insert a row.
CHANGER[1, 2]	Optional data. Indicates the authorization ID that last changed the row.
CHANGETIMESTAMP[1]	Optional data. Indicates when a row was changed. If you use CURRENT TIMESTAMP, DB2 automatically enters the value of CURRENT TIMESTAMP When you update a row.

Notes:

1. Optional columns are for administrator use. DB2 does not use these columns.
2. Because the CREATOR and CHANGER columns are CHAR(26), they are large enough for a three-part authorization ID. Separate each 8-byte part of the ID with a period in byte 9 and in byte 18. If you enter only the primary authorization ID, consider entering it right-justified in the field (that is, preceded by 18 blanks).

Controlling data definition

You can control the use of data definition through several installation options and through the ART and ORT. To install data definition control support and to control data definition behavior, specify the appropriate values on the DSNTIPZ installation panel, as shown with default values in Figure 16.

```
DSNTIPZ      INSTALL DB2 - DATA DEFINITION CONTROL SUPPORT
===>

Enter data below:

 1 INSTALL DD CONTROL SUPT. ===> NO        YES - activate the support
                                           NO  - omit DD control support
 2 CONTROL ALL APPLICATIONS ===> NO        YES or NO
 3 REQUIRE FULL NAMES       ===> YES       YES or NO
 4 UNREGISTERED DDL DEFAULT ===> ACCEPT    Action for unregistered DDL:
                                           ACCEPT - allow it
                                           REJECT - prohibit it
                                           APPL   - consult ART
 5 ART/ORT ESCAPE CHARACTER ===>           Used in ART/ORT Searches
 6 REGISTRATION OWNER       ===> DSNRGCOL          Qualifier for ART and ORT
 7 REGISTRATION DATABASE    ===> DSNRGFDB          Database name
 8 APPL REGISTRATION TABLE  ===> DSN_REGISTER_APPL Table name
 9 OBJT REGISTRATION TABLE  ===> DSN_REGISTER_OBJT Table name

Note: ART = Application Registration Table
      ORT = Object      Registration Table

PRESS:  ENTER to continue   RETURN to exit   HELP for more information
```

Figure 16. DSNTIPZ installation panel with default values

After you specify values on the installation panel, you enter the appropriate information in the ART and ORT to enable data definition control support.

This section explains what values to enter on the DSNTIPZ installation panel and what data to enter in the ART and ORT. This section first explains the basic installation options in “Installing data definition control support” on page 219. Then, the section explains the specific options for the following four methods of controlling data definition control support:

- “Controlling data definition by application name” on page 220 describes the simplest of the four methods for implementing data definition control.
- “Controlling data definition by application name with exceptions” on page 221 describes how to give one or more applications almost total control over data definition.
- “Controlling data definition by object name” on page 222 describes how to register all of the objects in a subsystem and have several applications control specific sets of objects.
- “Controlling data definition by object name with exceptions” on page 224 describes how to control registered and unregistered objects.

Finally, the section describes the optional task of registering sets of objects in “Registering sets of objects” on page 225.

Recommendation: As you read through the relevant sections in this chapter, use the template in Figure 17 on page 219 to record the values that you specify on the DSNTIPZ installation panel.

```
DSNTIPZ      INSTALL DB2 - DATA DEFINITION CONTROL SUPPORT
===>

Enter data below:

 1 INSTALL DD CONTROL SUPT. ===>          YES - activate the support
                                          NO  - omit DD control support
 2 CONTROL ALL APPLICATIONS ===>          YES or NO
 3 REQUIRE FULL NAMES       ===>          YES or NO
 4 UNREGISTERED DDL DEFAULT ===>          Action for unregistered DDL:
                                          ACCEPT - allow it
                                          REJECT - prohibit it
                                          APPL   - consult ART
 5 ART/ORT ESCAPE CHARACTER ===>          Used in ART/ORT Searches
 6 REGISTRATION OWNER       ===>                  Qualifier for ART and ORT
 7 REGISTRATION DATABASE    ===>                  Database name
 8 APPL REGISTRATION TABLE  ===>                  Table name
 9 OBJT REGISTRATION TABLE  ===>                  Table name

Note: ART = Application Registration Table
      ORT = Object      Registration Table

PRESS:  ENTER to continue   RETURN to exit   HELP for more information
```

Figure 17. DSNTIPZ installation panel

Installing data definition control support

To install data definition control support, perform the following steps:

1. Enter YES for the first option on the DSNTIPZ installation panel, as shown:

   ```
   1 INSTALL DD CONTROL SUPT. ===> YES
   ```

2. Enter the names for the registration tables in your DB2 subsystem, their owners, and the databases in which they reside for options 6, 7, 8, and 9 on the DSNTIPZ installation panel. The default names are as follows:

   ```
   6 REGISTRATION OWNER       ===> DSNRGCOL
   7 REGISTRATION DATABASE    ===> DSNRGFDB
   8 APPL REGISTRATION TABLE  ===> DSN_REGISTER_APPL
   9 OBJT REGISTRATION TABLE  ===> DSN_REGISTER_OBJT
   ```

   You can accept the default names or assign names of your own. If you specify your own table names, each name can have a maximum of 17 characters. This chapter uses the default names.

3. If you want to use the percent character (%) or the underscore character (_) as a regular character in the ART or ORT, enter an escape character for option 5 on the DSNTIPZ installation panel. You can use any special character other than underscore or percent as the escape character.

   Example: To use the pound sign (#) as an escape character, fill in option 5 as follows:

   ```
   5 ART/ORT ESCAPE CHARACTER ===> #
   ```

   After you specify the pound sign as an escape character, the pound sign can be used in names in the same way that an escape character is used in an SQL LIKE predicate.

   For more information about escape characters and the percent and underscore characters, see *DB2 SQL Reference*.

4. Register plans, packages, and objects in the ART and ORT, and enter values for the three other options on the DSNTIPZ installation panel as follows:

```
2 CONTROL ALL APPLICATIONS ===>
3 REQUIRE FULL NAMES       ===>
4 UNREGISTERED DDL DEFAULT ===>
```

Choose the values to enter and the plans, packages, and objects to register based on the control method that you plan to use:

- If you want to control data definition by **application name**, perform the steps in one of the following sections:
  - For registered applications that have total control over all data definition language in the DB2 subsystem, see "Controlling data definition by application name."
  - For registered applications that have total control with some exceptions, see "Controlling data definition by application name with exceptions" on page 221. If you also want to control data definition by **registering sets of objects**, perform the steps in "Registering sets of objects" on page 225.
- If you want to control data definition by **object name**, perform the steps in one of the following sections:
  - For subsystems in which all objects are registered and controlled by name, see "Controlling data definition by object name" on page 222. If you also want to control data definition by **registering sets of objects**, perform the steps in "Registering sets of objects" on page 225.
  - For subsystems in which some specific objects are registered and controlled, and data definition language is accepted for objects that are not registered, see "Controlling data definition by object name with exceptions" on page 224. If you also want to control data definition by **registering sets of objects**, perform the steps in "Registering sets of objects" on page 225.

Controlling data definition by application name

The simplest of the four methods for implementing data definition control is to give one or more applications total control over the use of data definition language in the subsystem. To control data definition by application name, perform the following steps:

1. Enter YES for the first option on the DSNTIPZ installation panel, as shown:

   ```
   2 CONTROL ALL APPLICATIONS ===> YES
   ```

   When you specify YES, only package collections or plans that are registered in the ART are allowed to use data definition statements.
2. In the ART, register all package collections and plans that you will allow to issue DDL statements, and enter the value Y in the DEFAULTAPPL column for these package collections. You must supply values for the APPLIDENT, APPLIDENTTYPE, and DEFAULTAPPL columns of the ART. You can enter information in other columns for your own use as indicated in Table 60 on page 216.

Example: Suppose that you want all data definition language in your subsystem to be issued only through certain applications. The applications are identified by the following application plan names, collection-IDs, and patterns:

PLANA	The name of an application plan
PACKB	The collection-ID of a package
TRULY%	A pattern name for any plan name beginning with TRULY
TR%	A pattern name for any plan name beginning with TR

Table 62 shows the entries that you need in your ART.

Table 62. Table DSN_REGISTER_APPL for total subsystem control

APPLIDENT	APPLIDENTTYPE	DEFAULTAPPL
PLANA	P	Y
PACKB	C	Y
TRULY%	P	Y
TR%	P	Y

An inactive table entry: If the row with TR% for APPLIDENT in Table 62 contains the value Y for DEFAULTAPPL, any plan with a name beginning with TR can execute data definition language. If DEFAULTAPPL is later changed to N to disallow that use, the changed row **does not prevent** plans beginning with TR from using data definition language; the row merely fails to allow that specific use. In this case, the plan TRXYZ is not allowed to use data definition language. However, the plan TRULYXYZ is allowed to use data definition language, by the row with TRULY% specified for APPLIDENT.

Controlling data definition by application name with exceptions

Registering application names with some exceptions is one of four methods for controlling data definition. If you want to give one or more applications **almost** total control over data definition language on all objects, you can reserve a few objects that can be created, altered, or dropped by applications that are not registered in the ART. To control data definition by application name with exceptions, perform the following steps:

1. Choose not to control all applications. On the DSNTIPZ installation panel, specify the following value for option 2:

   ```
   2 CONTROL ALL APPLICATIONS ===> NO
   ```

   When you specify NO, you allow unregistered applications to use data definition statements on some objects.
2. On the DSNTIPZ installation panel, specify the following for option 4:

   ```
   4 UNREGISTERED DDL DEFAULT ===> APPL
   ```

   When you specify APPL, you restrict the use of data definition statements for objects that are **not** registered in the ORT. If an object is registered in the ORT, any applications that are not registered in the ART can use data definition language on the object. However, if an object is not registered in the ORT, only applications that are registered in the ART can use data definition language on the object.
3. In the ART, register package collections and plans that you will allow to issue data definition statements on any object. Enter the value Y in the DEFAULTAPPL column for these package collections. Applications that are registered in the ART retain almost total control over data definition. Objects that are registered in the ORT are the only exceptions. See step 2 of "Controlling data definition by application name" on page 220 for more information about registering package collections and plans in the ART.
4. In the ORT, register all objects that are exceptions to the subsystem data definition control that you defined in the ART. You must supply values for the QUALIFIER, NAME, TYPE, APPLMATCHREQ, APPLIDENT, and

APPLIDENTTYPE columns of the ORT. You can enter information in other columns of the ORT for your own use as indicated in Table 61 on page 217.

Example: Suppose that you want almost all of the data definition language in your subsystem to be issued only through an application plan (PLANA) and a package collection (PACKB).

Table 63 shows the entries that you need in your ART.

Table 63. Table DSN_REGISTER_APPL for total subsystem control with exceptions

APPLIDENT	APPLIDENTTYPE	DEFAULTAPPL
PLANA	P	Y
PACKB	C	Y

However, you also want the following specific exceptions:

- Object KIM.VIEW1 can be created, altered, or dropped by the application plan PLANC.
- Object BOB.ALIAS can be created, altered, or dropped only by the package collection PACKD.
- Object FENG.TABLE2 can be created, altered, or dropped by **any** plan or package collection.
- Objects with names that begin with SPIFFY.MSTR and exactly one following character can be created, altered, or dropped by any plan that matches the name pattern TRULY%. For example, the plan TRULYJKL can create, alter, or drop the object SPIFFY.MSTRA.

Table 64 shows the entries that are needed to register these exceptions in the ORT.

Table 64. Table DSN_REGISTER_OBJT for subsystem control with exceptions

QUALIFIER	NAME	TYPE	APPLMATCHREQ	APPLIDENT	APPLIDENTTYPE
KIM	VIEW1	C	Y	PLANC	P
BOB	ALIAS	C	Y	PACKD	C
FENG	TABLE2	C	N		
SPIFFY	MSTR_	C	Y	TRULY%	P

You can register objects in the ORT individually, or you can register sets of objects. For information about registering sets of objects, see “Registering sets of objects” on page 225.

Controlling data definition by object name

Registering object names is one of four methods for controlling data definition.If you want all objects in the subsystem to be registered and you want several applications to control specific sets of objects, you need to control by object name. When you control by object name, all objects are registered regardless of whether they are controlled by specific applications. To control data definition by object name, perform the following steps:

1. Choose not to control all applications. On the DSNTIPZ installation panel, specify the following value for option 2:

   ```
   2 CONTROL ALL APPLICATIONS ===> NO
   ```

When you specify NO, you allow unregistered applications to use data definition statements on some objects.

2. On the DSNTIPZ installation panel, fill in option 4 as follows:

   ```
   4 UNREGISTERED DDL DEFAULT ===> REJECT
   ```

   When you specify REJECT for option 4, you totally restrict the use of data definition statements for objects that are not registered in the ORT. Therefore, **no** application can use data definition language for any unregistered object.

3. In the ORT, register all of the objects in the subsystem, and enter Y in the APPLMATCHREQ column. You must supply values for the QUALIFIER, NAME, TYPE, APPLMATCHREQ, APPLIDENT, and APPLIDENTTYPE columns of the ORT. You can enter information in other columns of the ORT for your own use as indicated in Table 61 on page 217.

4. In the ART, register any plan or package collection that can use a set of objects that you register in the ORT with an incomplete name. Enter the value Y in the QUALIFIEROK column. These plans or package collections can use data definition lanaguage on sets of objects regardless of whether a set of objects has a value of Y in the APPLMATCHREQ column.

Example: Table 65 on page 224 shows entries in the ORT for a DB2 subsystem that contains the following objects that are controlled by object name:

- Two storage groups (STOG1 and STOG2) and a database (DATB1) that are not controlled by a specific application. These objects can be created, altered, or dropped by a user with the appropriate authority by using any application, such as SPUFI or QMF.
- Two table spaces (TBSP1 and TBSP2) that are not controlled by a specific application. Their names are qualified by the name of the database in which they reside (DATB1).
- Three objects (OBJ1, OBJ2, and OBJ3) whose names are qualified by the authorization IDs of their owners. Those objects might be tables, views, indexes, synonyms, or aliases. Data definition statements for OBJ1 and OBJ2 can be issued only through the application plan named PLANX. Data definition statements for OBJ3 can be issued only through the package collection named PACKX.
- Objects that match the qualifier pattern E%D and the name OBJ4 can be created, altered, or deleted by application plan SPUFI. For example, the objects EDWARD.OBJ4, ED.OBJ4, and EBHARD.OBJ4, can be created, altered, or deleted by application plan SPUFI. Entry E%D in the QUALIFIER column represents all three objects.
- Objects with names that begin with TRULY.MY_, where the underscore character is actually part of the name. Assuming that you specify # as the escape character, all of the objects with this name pattern can be created, altered, or dropped only by plans with names that begin with TRULY.

Assume the following installation option:

```
3 REQUIRE FULL NAMES       ===> YES
```

Entries in Table 65 on page 224 do not specify incomplete names. Hence, objects that are not represented in the table cannot be created in the subsystem, except by an ID with installation SYSADM authority.

Table 65. Table DSN_REGISTER_OBJT for total control by object

QUALIFIER	NAME	TYPE	APPLMATCHREQ	APPLIDENT	APPLIDENTTYPE
	STOG1	S	N		
	STOG2	S	N		
	DATB1	D	N		
DATB1	TBSP1	T	N		
DATB1	TBSP2	T	N		
KIM	OBJ1	C	Y	PLANX	P
FENG	OBJ2	C	Y	PLANX	P
QUENTIN	OBJ3	C	Y	PACKX	C
E%D	OBJ4	C	Y	SPUFI	P
TRULY	MY#_%	C	Y	TRULY%	P

You can register objects in the ORT individually, or you can register sets of objects. For information about registering sets of objects, see "Registering sets of objects" on page 225.

Controlling data definition by object name with exceptions

Registering object names with some exceptions is one of four methods for controlling data definition. If you want several applications to control specific sets of registered objects and you want to allow other applications to use data definition statements for unregistered objects, perform the following steps:

1. Choose not to control all applications. On the DSNTIPZ installation panel, specify the following value for option 2:

   ```
   2 CONTROL ALL APPLICATIONS ===> NO
   ```

   When you specify NO, you allow unregistered applications to use data definition statements on some objects.
2. On the DSNTIPZ installation panel, fill in option 4 as follows:

   ```
   4 UNREGISTERED DDL DEFAULT ===> ACCEPT
   ```

   This option **does not** restrict the use of data definition statements for objects that are not registered in the ORT. Therefore, **any** application can use data definition language for any unregistered object.
3. Register all controlled objects in the ORT. Use a name and qualifier to identify a single object. Use only one part of a two-part name to identify a set of objects that share just that part of the name. For each controlled object, use APPLMATCHREQ = Y. Enter the name of the plan or package collection that controls the object in the APPLIDENT column.
4. For each set of controlled objects (identified by only a simple name in the ORT), register the controlling application in the ART. You must supply values for the APPLIDENT, APPLIDENTTYPE, and QUALIFIEROK columns of the ART.

Example: The following two tables assume that the installation option REQUIRE FULL NAMES is set to NO, as described in "Registering sets of objects" on page 225. Table 66 on page 225 shows entries in the ORT for the following controlled objects:

- The objects KIM.OBJ1, FENG.OBJ2, QUENTIN.OBJ3, and EDWARD.OBJ4, all of which are controlled by PLANX or PACKX, as described under "Controlling data definition by object name" on page 222. DB2 cannot interpret the object names as incomplete names because the objects that control them, PLANX and PACKX, are registered in Table 67 with QUALIFIEROK=N.
- Two sets of objects, *.TABA and *.TABB, which are controlled by PLANA and PACKB, respectively.

Table 66. Table DSN_REGISTER_OBJT for object control with exceptions

QUALIFIER	NAME	TYPE	APPLMATCHREQ	APPLIDENT	APPLIDENTTYPE
KIM	OBJ1	C	Y	PLANX	P
FENG	OBJ2	C	Y	PLANX	P
QUENTIN	OBJ3	C	Y	PACKX	C
EDWARD	OBJ4	C	Y	PACKX	C
	TABA	C	Y	PLANA	P
	TABB	C	Y	PACKB	C

Table 67 shows the entries in the corresponding ART.

Table 67. Table DSN_REGISTER_APPL for object control with exceptions

APPLIDENT	APPLIDENTTYPE	DEFAULTAPPL	QUALIFIEROK
PLANX	P	N	N
PACKX	C	N	N
PLANA	P	N	Y
PACKB	C	N	Y

In this situation, with the combination of installation options shown previously, any application can use data definition language for objects that are **not** covered by entries in the ORT. For example, if HOWARD has the CREATETAB privilege, HOWARD can create the table HOWARD.TABLE10 through any application.

You can register objects in the ORT individually, or you can register sets of objects. For information about registering sets of objects, see "Registering sets of objects."

Registering sets of objects

Registering sets of objects is not a data definition control method; you must install one of the following data definition control methods before registering sets of objects:

- "Controlling data definition by application name with exceptions" on page 221
- "Controlling data definition by object name" on page 222
- "Controlling data definition by object name with exceptions" on page 224

Registering sets of objects allows you to save time and to simplify object registration. Because complete two-part names are not required for every object that is registered in the ORT, you can use incomplete names to register sets of objects. To use incomplete names and register sets of objects, fill in option 3 on the DSNTIPZ installation panel as follows:

```
3 REQUIRE FULL NAMES       ===> NO
```

The default value YES requires you to use both parts of the name for each registered object. If you specify the value NO, an incomplete name in the ORT

represents a set of objects that all share the same value for one part of a two-part name. Objects that are represented by incomplete names in the ORT require an authorizing entry in the ART.

Example: If you specify NO for option 3, you can include entries with incomplete names in the ORT. Table 68 shows entries in the ORT for the following objects:

- Two sets of objects, *.TABA and *.TABB, which are controlled by PLANX and PACKY, respectively. Only PLANX can create, alter, or drop any object whose name is *.TABA. Only PACKY can create, alter, or drop any object whose name is *.TABB. PLANX and PACKY must also be registered in the ART with QUALIFIEROK set to Y, as shown in Table 69. That setting allows the applications to use sets of objects that are registered in the ORT with an incomplete name.
- Tables, views, indexes, or aliases with names like SYSADM.*.
- Table spaces with names like DBSYSADM.*; that is, table spaces in database DBSYSADM.
- Tables with names like USER1.* **and** tables with names like *.TABLEX.

Table 68. Table DSN_REGISTER_OBJT for objects with incomplete names

QUALIFIER	NAME	TYPE	APPLMATCHREQ	APPLIDENT	APPLIDENTTYPE
	TABA	C	Y	PLANX	P
	TABB	C	Y	PACKY	C
SYSADM		C	N		
DBSYSADM		T	N		
USER1	TABLEX	C	N		

ART entries for objects with incomplete names in the ORT: APPLMATCHREQ=N and objects SYSAMD.*, DBSYSADM.*, USER1.*, and *.TABLEX can be created, altered, or dropped by any package collection or application plan. However, the collection or plan that creates, alters, or drops such an object must be registered in the ART with QUALIFIEROK=Y to allow it to use incomplete object names.

Table 69 shows that PLANA and PACKB are registered in the ART to use sets of objects that are registered in the ORT with incomplete names.

Table 69. Table DSN_REGISTER_APPL for plans that use sets of objects

APPLIDENT	APPLIDENTTYPE	DEFAULTAPPL	QUALIFIEROK
PLANA	P	N	Y
PACKB	C	N	Y

Managing the registration tables and their indexes

This section explains how to create and maintain the ART and the ORT. For a description of all of the columns in these tables, see “Columns of the ART” on page 216 and “Columns of the ORT” on page 217.

Creating the tables and indexes

The ART, the ORT, and the required unique indexes on them are created when you install data definition control support. If you drop any of these objects, you can re-create them using the CREATE statements shown here:

CREATE statements for the ART and its index:

```
CREATE TABLE DSNRGCOL.DSN_REGISTER_APPL
  (APPLIDENT          VARCHAR(128) NOT NULL WITH DEFAULT,
   APPLIDENTTYPE      CHAR(1)      NOT NULL WITH DEFAULT,
   APPLICATIONDESC    VARCHAR(30)  NOT NULL WITH DEFAULT,
   DEFAULTAPPL        CHAR(1)      NOT NULL WITH DEFAULT,
   QUALIFIEROK        CHAR(1)      NOT NULL WITH DEFAULT,
   CREATOR            CHAR(26)     NOT NULL WITH DEFAULT,
   CREATETIMESTAMP    TIMESTAMP    NOT NULL WITH DEFAULT,
   CHANGER            CHAR(26)     NOT NULL WITH DEFAULT,
   CHANGETIMESTAMP    TIMESTAMP    NOT NULL WITH DEFAULT)
  IN DSNRGFDB.DSNRGFTS;

CREATE UNIQUE INDEX DSNRGCOL.DSN_REGISTER_APPLI
  ON DSNRGCOL.DSN_REGISTER_APPL
  (APPLIDENT, APPLIDENTTYPE, DEFAULTAPPL DESC, QUALIFIEROK DESC)
  CLUSTER;
```

CREATE statements for the ORT and its index:

```
CREATE TABLE DSNRGCOL.DSN_REGISTER_OBJT
  (QUALIFIER          CHAR(8)      NOT NULL WITH DEFAULT,
   NAME               CHAR(18)     NOT NULL WITH DEFAULT,
   TYPE               CHAR(1)      NOT NULL WITH DEFAULT,
   APPLMATCHREQ       CHAR(1)      NOT NULL WITH DEFAULT,
   APPLIDENT          VARCHAR(128) NOT NULL WITH DEFAULT,
   APPLIDENTTYPE      CHAR(1)      NOT NULL WITH DEFAULT,
   APPLICATIONDESC    VARCHAR(30)  NOT NULL WITH DEFAULT,
   CREATOR            CHAR(26)     NOT NULL WITH DEFAULT,
   CREATETIMESTAMP    TIMESTAMP    NOT NULL WITH DEFAULT,
   CHANGER            CHAR(26)     NOT NULL WITH DEFAULT,
   CHANGETIMESTAMP    TIMESTAMP    NOT NULL WITH DEFAULT)
  IN DSNRGFDB.DSNRGFTS;

CREATE UNIQUE INDEX DSNRGCOL.DSN_REGISTER_OBJTI
  ON DSNRGCOL.DSN_REGISTER_OBJT
  (QUALIFIER, NAME, TYPE) CLUSTER;
```

You can alter these CREATE statements in the following ways:
- Add columns to the ends of the tables
- Assign an auditing status
- Choose buffer pool or storage options for indexes
- Declare table check constraints to limit the types of entries that are allowed

Naming registration tables and indexes

Every member of a data sharing group must have the same names for the ART and ORT tables.

Recommendation: Avoid changing ART and ORT table names. If you change either of the table names, their owner, or their database, you must reinstall DB2 in update mode and make the corresponding changes on the DSNTIPZ installation panel.

Name the required index by adding the letter I to the corresponding table name. For example, suppose that you are naming a required index for the ART named ABC. You should name the required index ABCI.

Dropping registration tables and indexes

If you drop any of the registration tables or their indexes, most data definition statements are rejected until the dropped objects are re-created. The only data definition statements that are allowed under such circumstances are those that create the following objects:

- Registration tables that are defined during installation

- Indexes of the registration tables that are defined during installation
- Table spaces that contain the registration tables that are defined during installation
- The database that contains the registration tables that are defined during installation

Creating a table space for the ART and ORT

The installation job DSNTIJSG creates a segmented table space to hold the ART and ORT, using this statement:

```
CREATE TABLESPACE DSNRGFTS IN DSNRGFDB SEGSIZE 4 CLOSE NO;
```

If you want to use a table space with a different name or different attributes, you can modify job DSNTIJSG before installing DB2. Alternatively, you can drop the table space and re-create it, the two tables, and their indexes.

Adding columns

You can add columns to either registration table for your own use, by using the ALTER TABLE statement. If you add columns, the additional columns must come at the end of the table, after existing columns.

Recommendation: Use a special character, such as the plus sign (+), in your column names to avoid possible conflict. If IBM adds columns to the ART or the ORT in future releases, the column names will contain only letters and numbers.

Updating the tables

You can update either the ART or the ORT with the LOAD utility, or with SQL INSERT, UPDATE, or DELETE statements.

Recommendation: Because security provisions are important, allow only a restricted set of authorization IDs, or perhaps only those with SYSADM authority, to update the ART. Consider assigning a validation exit routine to the ORT, to allow applications to change only those rows that have the same application identifier in the APPLIDENT column. A registration table cannot be updated until all jobs whose data definition statements are controlled by the table have completed.

Stopping data definition control

When data definition control is active, only IDs with installation SYSADM or installation SYSOPR authority are able to stop the database, a table space, or an index space that contains a registration table or index. When the object is stopped, only an ID with one of those authorities can start it again.

Bypassing data definition control: An ID with install SYSADM authority can execute data definition statements regardless of whether data definition control is active, and regardless of whether the ART or ORT is available. To bypass data definition control, an ID with install SYSDAM authority can use the following methods:

- If the ID is the owner of the plan or package that contains the statement, the ID can bypass data definition control by using a static SQL statement.
- If the ID is the current SQL ID, the ID can bypass data definition control through a dynamic CREATE statement.

- If the ID is the current SQL ID, the primary ID, or any secondary ID of the executing process, the ID can bypass data definition control through a dynamic ALTER or DROP statement.

Chapter 11. Controlling access to a DB2 subsystem

This chapter explains how to control access to the DB2 subsystem from different environments and how to associate a process with an intended set of authorization IDs.

Recommendation for external security system: Control access through an external security system, for which Resource Access Control Facility (RACF) is the model. "Establishing RACF protection for DB2" on page 264 describes how to make DB2 and its IDs known to RACF.

If you are using RACF, you can also define multilevel security for DB2 resources, which is described in "Multilevel security" on page 192.

Control by RACF is not strictly necessary, and some alternatives are described under "Other methods of controlling access" on page 279. However, most of the information in this chapter assumes that RACF, or an equivalent product, is already in place.

Local requests only: If you do not accept requests from or send requests to remote locations, begin reading this chapter with "Controlling local requests" on page 232. When you reach "Controlling requests from remote applications" on page 238, you can skip forward to "Establishing RACF protection for DB2" on page 264.

Remote requests: If you accept requests from remote applications, read "Controlling requests from remote applications" on page 238. "Controlling requests from remote applications" on page 238 describes the security checks that a remote request is subject to before it can access your DB2 subsystem. The level of security differs depending on whether the requesting application is using SNA or Transmission Control Protocol/Internet Protocol (TCP/IP) protocols to access DB2. After the local system authenticates the incoming ID, it treats the ID like a local connection request or a local sign-on request. You can process the ID with your connection or sign-on exit routine and associate secondary authorization IDs with the ID. For more information about local connection requests, see "Controlling local requests" on page 232.

If you are sending requests to a remote DB2 subsystem, that subsystem can subject your requests to various security checks. For suggestions on how to plan for these checks, see "Planning to send remote requests" on page 252. If you send requests to a remote DBMS that is not DB2 UDB for z/OS, use the documentation for that DRDA database server.

Topics covered in this chapter:

- "Controlling local requests" on page 232
- "Processing connections" on page 232
- "Processing sign-ons" on page 236
- "Controlling requests from remote applications" on page 238
- "Planning to send remote requests" on page 252
- "Establishing RACF protection for DB2" on page 264
- "Establishing Kerberos authentication through RACF" on page 278
- "Other methods of controlling access" on page 279

© Copyright IBM Corp. 1982, 2008

Controlling local requests

Different local processes enter the access control procedure at different points, depending on the environment in which they originate. The following processes go through connection processing only:

- Requests originating in TSO foreground and background (including online utilities and requests through the call attachment facility)
- JES-initiated batch jobs
- Requests through started task control address spaces (from the z/OS START command)

The following processes go through connection processing and can later go through the sign-on exit also.

- The IMS control region.
- The CICS recovery coordination task.
- DL/I batch.
- Applications that connect using the Resource Recovery Services attachment facility (RRSAF). (See Part 6 of *DB2 Application Programming and SQL Guide* for more information.)

The following processes go through sign-on processing:

- Requests from IMS dependent regions (including MPP, BMP, and Fast Path)
- CICS transaction subtasks

For instructions on controlling the IDs that are associated with connection requests, see "Processing connections." For instructions on controlling the IDs that are associated with sign-on requests, see "Processing sign-ons" on page 236.

IMS, CICS, RRSAF, and DDF-to-DDF connections can send a sign-on request, typically to execute an application plan. That request must provide a primary ID, and can also provide secondary IDs. After a plan is allocated, it need not be deallocated until a new plan is required. A different transaction can use the same plan by issuing a new sign-on request with a new primary ID.

Processing connections

A connection request makes a new connection to DB2; it does not reuse an application plan that is already allocated. Therefore, an essential step in processing the request is to check that the ID is authorized to use DB2 resources, as shown in Figure 18.

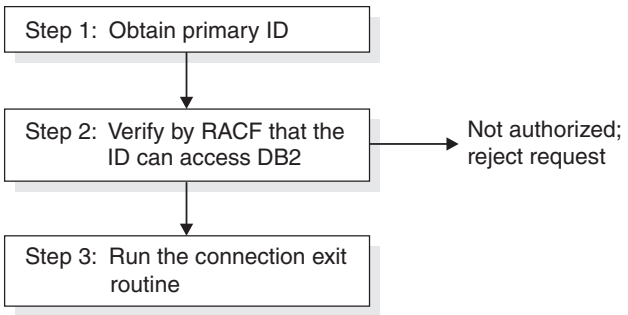


Figure 18. Connection processing

Steps in processing connections

Processing connections involves the following steps:

1. DB2 obtains the initial primary authorization ID. Table 70 shows how the source of the ID depends on the type of address space from which the connection was made.

Table 70. Sources of initial primary authorization IDs

Source	Initial primary authorization ID
TSO	TSO logon ID.
BATCH	USER parameter on JOB statement.
IMS control region or CICS	USER parameter on JOB statement.
IMS or CICS started task	Entries in the started task control table.
Remote access requests	Depends on the security mechanism used. See "Overview of security mechanisms for DRDA and SNA" on page 238 for more details.

2. RACF is called through the z/OS system authorization facility (SAF) to check whether the ID that is associated with the address space is authorized to use the following resources:
   - The DB2 resource class (CLASS=DSNR)
   - The DB2 subsystem (SUBSYS=*ssnm*)
   - The requested connection type

   For instructions on authorizing the use of these resources, see "Permitting RACF access" on page 267.

   The SAF return code (RC) from the invocation determines the next step, as follows:
   - **If RC > 4**, RACF determined that the RACF user ID is not valid or does not have the necessary authorization to access the resource name. DB2 rejects the request for a connection.
   - **If RC = 4**, the RACF return code is checked.
     - If RACF return code value is **equal to 4**, the resource name is not defined to RACF and DB2 rejects the request with reason code X'00F30013'. For instructions on defining the resource name, see "Defining DB2 resources to RACF" on page 265.
     - If RACF return code value is **not equal to 4**, RACF is not active. DB2 continues with the next step, but the connection request and the user are not verified.
   - **If RC = 0**, RACF is active and has verified the RACF user ID; DB2 continues with the next step.
3. If RACF is active and has verified the RACF user ID, DB2 runs the connection exit routine. To use DB2 secondary IDs, you must replace the exit routine. See "Supplying secondary IDs for connection requests" on page 234.

   If you do not want to use secondary IDs, do nothing. The IBM-supplied default connection exit routine continues the connection processing. The process has the following effects:
   - The DB2 primary authorization ID is set based on the following rules:
     - If a value for the initial primary authorization ID exists, the value becomes the DB2 primary ID.
     - If no value exists (the value is blank), the primary ID is set by default, as shown in Table 71 on page 234.

Table 71. Sources of default authorization identifiers

Source	Default primary authorization ID
TSO	TSO logon ID
BATCH	USER parameter on JOB statement
Started task, or batch job with no USER parameter	Default authorization ID set when DB2 was installed (UNKNOWN AUTHID on installation panel DSNTIPP)
Remote request	None. The user ID is required and is provided by the DRDA requester.

- The SQL ID is set equal to the primary ID.
- No secondary IDs exist.

If you want to use secondary IDs, see the description in "Supplying secondary IDs for connection requests." Of course, you can also replace the exit routine with one that provides different default values for the DB2 primary ID. If you have written such a routine for an earlier release of DB2, it will probably work for this release with no change.

Supplying secondary IDs for connection requests

If you want to use DB2 secondary authorization IDs, you must replace the default connection exit routine. If you want to use RACF group names as DB2 secondary IDs, as illustrated in "Examples of granting and revoking privileges" on page 173,, the easiest method is to use the IBM-supplied sample routine.

The difference between the default connection exit routine and the sample connection exit routine are shown in Table 72.

Table 72. Differences between default connection exit routine and sample connection exit routine

Default connection exit routine	Sample connection exit routine
Supplied as object code.	Supplied as source code. You can change the code.
Installed as part of the normal DB2 installation procedure.	Must be compiled and placed in the DB2 library.
Provides values for primary IDs and SQL IDs, but does not provide values for secondary IDs.	Provides values for primary IDs, secondary IDs, and SQL IDs.

Installation job DSNTIJEX replaces the default connection exit routine with the sample connection exit routine; for more information, see Part 2 of *DB2 Installation Guide*.

The sample connection exit routine has the following effects:

- The sample connection exit routine sets the DB2 primary ID in the same way that the default routine sets the DB2 primary ID, and according to the following rules:
  - If the initial primary ID is not blank, the initial ID becomes the DB2 primary ID.
  - If the initial primary ID is blank, the sample routine provides the same default value as does the default routine.

  - If the sample routine cannot find a nonblank primary ID, DB2 uses the default ID (UNKNOWN AUTHID) from the DSNTIPP installation panel. In this case, no secondary IDs are supplied.
- The sample connection exit routine sets the SQL ID based on the following criteria:
  - The routine sets the SQL ID to the TSO data set name prefix in the TSO user profile table if the following conditions are true:
    - The connection request is from a TSO-managed address space, including the call attachment facility, the TSO foreground, and the TSO background.
    - The TSO data set name prefix is equal to the primary ID or one of the secondary IDs.
  - In all other cases, the routine sets the SQL ID equal to the primary ID.
- The secondary authorization IDs depend on RACF options:
  - If RACF is not active, no secondary IDs exist.
  - If RACF is active but its "list of groups" option is not active, one secondary ID exists (the default connected group name) if the attachment facility supplied the default connected group name.
  - If RACF is active and the "list of groups" option is active, the routine sets the list of DB2 secondary IDs to the list of group names to which the RACF user ID is connected. Those RACF user IDs that are in REVOKE status do not become DB2 secondary IDs. The maximum number of groups is 1012. The list of group names is obtained from RACF and includes the default connected group name.

If the default connection exit routine and the sample connection exit routine do not provide the flexibility and features that your subsystem requires, you can write your own exit routine. For instructions on writing your own exit routine, see Appendix B, "Writing exit routines," on page 1051.

Required CICS specifications

For a CICS transaction to use the sample connection or sign-on exit routines, the external security system, such as RACF, must be defined to CICS with the following specifications:

- The CICS system initialization table must specify external security.
  - For CICS Version 4 or later, specify SEC=YES.
  - For earlier releases of CICS, specify EXTSEC=YES.

  If you are using the CICS multiple region option (MRO), you must specify SEC=YES or EXTSEC=YES for every CICS system that is connected by interregion communication (IRC).
- If your version of CICS uses a sign-on table (SNT), the CICS sign-on table must specify EXTSEC=YES for each signed on user that uses the sign-on exit.
- When the user signs on to a CICS terminal-owning region, the terminal-owning region must propagate the authorization ID to the CICS application-owning region. For more information on that propagation, see the description of ATTACHSEC in the applicable version of the *CICS Intercommunication Guide*.

You must change the sample sign-on exit routine (DSN3SSGN) before using it if the following conditions are all true:

- You have the RACF list-of-groups option active.
- You have transactions whose initial primary authorization ID is not defined to RACF.

For instructions on changing the sample sign-on exit routine, see “Sample connection and sign-on routines” on page 1052.

Processing sign-ons

For requests from IMS dependent regions, CICS transaction subtasks, or RRS connections, the initial primary ID is not obtained until just before allocating a plan for a transaction. A new sign-on request can run the same plan without deallocating the plan and reallocating it. Nevertheless, the new sign-on request can change the primary ID.

Unlike connection processing, sign-on processing does not check the RACF user ID of the address space. The steps in processing sign-ons are shown in Figure 19.

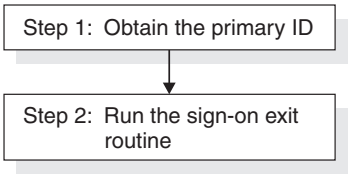


Figure 19. Sign-on processing

Steps in processing sign-ons

DB2 takes the following steps in processing sign-ons:

1. DB2 determines the initial primary ID as follows:

   For IMS sign-ons from message-driven regions, if the user has signed on, the initial primary authorization ID is the user's sign-on ID.

   IMS passes to DB2 the IMS sign-on ID and the associated RACF connected group name, if one exists.

   If the user has not signed on, the primary ID is the LTERM name, or if that is not available, the PSB name.

   For a batch-oriented region, the primary ID is the value of the USER parameter on the job statement, if that is available. If that is not available, the primary ID is the program's PSB name.

   For information about authorization IDs, CICS, and RDO, see the following books:

   - *CICS Transaction Server for OS/390 Resource Definition Guide*
   - *CICS Transaction Server for z/OS DB2 Guide*

   For remote requests, the source of the initial primary ID is determined by entries in the SYSIBM.USERNAMES table. “Accepting a remote attachment request” on page 242 explains how to control the ID.

   For connections using Resource Recovery Services attachment facility, the processing depends on the type of signon request:
   - SIGNON
   - AUTH SIGNON
   - CONTEXT SIGNON

   For SIGNON, the primary authorization ID is retrieved from ACEEUSRI if an ACEE is associated with the TCB (TCBSENV). This is the normal case. However, if an ACEE is not associated with the TCB, SIGNON uses the primary authorization ID that is associated with the address space, that is, from the ASXB. If the new primary authorization ID was retrieved from the ACEE

that is associated with the TCB and ACEEGRPN is not null, DB2 uses ACEEGRPN to establish secondary authorization IDs.

With AUTH SIGNON, an APF-authorized program can pass a primary authorization ID for the connection. If a primary authorization ID is passed, AUTH SIGNON also uses the value that is passed in the secondary authorization ID parameter to establish secondary authorization IDs. If the primary authorization ID is not passed, but a valid ACEE is passed, AUTH SIGNON uses the value in ACEEUSRI for the primary authorization ID if ACEEUSRL is not 0. If ACEEUSRI is used for the primary authorization ID, AUTH SIGNON uses the value in ACEEGRPN as the secondary authorization ID if ACEEGRPL is not 0.

For CONTEXT SIGNON, the primary authorization ID is retrieved from data that is associatedail with the current RRS context using the context_key, which is supplied as input. CONTEXT SIGNON uses the CTXSDTA and CTXRDTA functions of RRS context services. An authorized function must use CTXSDTA to store a primary authorization ID prior to invoking CONTEXT SIGNON. Optionally, CTXSDTA can be used to store the address of an ACEE in the context data that has a context_key that was supplied as input to CONTEXT SIGNON. DB2 uses CTXRDTA to retrieve context data. If an ACEE address is passed, CONTEXT SIGNON uses the value in ACEEGRPN as the secondary authorization ID if ACEEGRPL is not 0.

For more information, see Part 6 of *DB2 Application Programming and SQL Guide*.

2. DB2 runs the sign-on exit routine. **User action:** To use DB2 secondary IDs, you must replace the exit routine.

   If you **do not** want to use secondary IDs, do nothing. Sign-on processing is then continued by the IBM-supplied default sign-on exit routine, which has the following effects:
   - The initial primary authorization ID remains the primary ID.
   - The SQL ID is set equal to the primary ID.
   - No secondary IDs exist.

   You can replace the exit routine with one of your own, even if it has nothing to do with secondary IDs. If you do, remember that IMS and CICS recovery coordinators, their dependent regions, and RRSAF take the exit routine only if they have provided a user ID in the sign-on parameter list.

   If you *do* want to use secondary IDs, see the description that follows.

Supplying secondary IDs for sign-on requests

If you want the primary authorization ID to be associated with DB2 secondary authorization IDs, you must replace the default sign-on exit routine. The procedure is like that for connection processing: If you want to use RACF group names as DB2 secondary IDs, the easiest method is to use the IBM-supplied sample routine. An installation job can automatically replace the default routine with the sample routine; to run it, see “Installation Step 6: Define User Authorization Exit Routines: DSNTIJEX” in Part 2 of *DB2 Installation Guide*.

Distinguish carefully between the two routines. The default sign-on routine provides no secondary IDs and has the effects described in step 2 of “Processing sign-ons” on page 236. The sample sign-on routine supports DB2 secondary IDs, and is like the sample connection routine.

The sample sign-on routine has the following effects:

- The initial primary authorization ID is left unchanged as the DB2 primary ID.

- The SQL ID is made equal to the DB2 primary ID.
- The secondary authorization IDs depend on RACF options:
  - If RACF is not active, no secondary IDs exist.
  - If RACF is active but its "list of groups" option is not active, one secondary ID exists; it is the name passed by CICS or by IMS.
  - If RACF is active and you have selected the option for a list of groups, the routine sets the list of DB2 secondary IDs to the list of group names to which the RACF user ID is connected, up to a limit of 1012 groups. The list of group names includes the default connected group name.

Controlling requests from remote applications

If you are controlling requests from remote applications, your DB2 subsystem might be accepting requests from applications that use SNA network protocols, TCP/IP network protocols, or both. This section describes the methods that the DB2 server can use to control access from those applications. To understand what is described here, you must be familiar with the communications database, which is part of the DB2 catalog. This section describes the following topics:

- "Overview of security mechanisms for DRDA and SNA"
- "The communications database for the server" on page 240
- "Controlling inbound connections that use SNA protocols" on page 242
- "Controlling inbound connections that use TCP/IP protocols" on page 249

Overview of security mechanisms for DRDA and SNA

SNA and DRDA have different security mechanisms. DRDA lets a user be authenticated using SNA security mechanisms or DRDA mechanisms, which are independent of the underlying network protocol. For an SNA network connection, a DRDA requester can send security tokens using a SNA attach or using DRDA commands. DB2 UDB for z/OS as a requester uses SNA security mechanisms if it uses a SNA network connection (except for Kerberos) and DRDA security mechanisms for TCP/IP network connections (or when Kerberos authentication is chosen, regardless of the network type).

Mechanisms used by DB2 UDB for z/OS as a requester

DB2 UDB for z/OS as a requester chooses SNA or DRDA security mechanisms based on the network protocol and the authentication mechanisms you use. If you use SNA protocols, DB2 supports the following SNA authentication mechanisms:

- User ID only (already verified)
- User ID and password, described in "Sending passwords" on page 262
- User ID and PassTicket, described in "Sending RACF PassTickets" on page 263

Authentication is performed based on SNA protocols, which means that the authentication tokens are sent in an SNA attach (FMH-5).

If you use TCP/IP protocols, DB2 supports the following DRDA authentication mechanisms:

- User ID only (already verified)
- User ID and password, described in "Sending passwords" on page 262
- User ID and PassTicket, described in "Sending RACF PassTickets" on page 263

If you use TCP/IP protocols with the z/OS Integrated Cryptographic Service Facility, DB2 also supports the following DRDA authentication mechanisms:

- Encrypted user ID and encrypted password
- Encrypted user ID and encrypted security-sensitive data
- Encrypted user ID, encrypted password, and encrypted security-sensitive data

- Encrypted user ID, encrypted password, encrypted new password, and encrypted security-sensitive data

Authentication is performed based on DRDA protocols, which means that the authentication tokens are sent in DRDA security flows.

If you use a requester other than DB2 UDB for z/OS, refer to that product's documentation.

Mechanisms accepted by DB2 UDB for z/OS as a server

As a server, DB2 UDB for z/OS can accept either SNA or DRDA authentication mechanisms. This means that DB2 can authenticate remote users from either the security tokens obtained from the SNA ATTACH (FMH-5) or from the DRDA security commands described by each of the protocols. The following authentication methods are supported by DB2 UDB for z/OS as a server:

- User ID only (already verified at the requester)
- User ID and password, described in "Sending passwords" on page 262
- User ID and PassTicket, described in "Sending RACF PassTickets" on page 263
- Kerberos tickets, described in "Establishing Kerberos authentication through RACF" on page 278
- Unencrypted user ID and encrypted password, described in "Sending encrypted passwords from a workstation" on page 264
- Encrypted user ID and encrypted password, described in "Sending encrypted passwords from a workstation" on page 264
- User ID, password, and new password, described in "Allowing users to change expired passwords"

DB2 UDB for z/OS as a server also supports the following authentication mechanisms if the z/OS Integrated Cryptographic Service Facility is installed and active:

- Encrypted user ID and encrypted security-sensitive data
- Encrypted user ID, encrypted password, and encrypted security-sensitive data

Allowing users to change expired passwords: DB2 can return to the DRDA requester information about errors and expired passwords. To allow this, specify YES in the EXTENDED SECURITY field of installation panel DSNTIPR.

When the DRDA requester is notified that the RACF password has expired, and the requester has implemented function to allow passwords to be changed, the requester can prompt the end user for the old password and a new password. The requester sends the old and new passwords to the DB2 server. This function is supported through DB2 Connect.

With the extended security option, DB2 passes the old and new passwords to RACF. If the old password is correct, and the new password meets the installation's password requirements, the end user's password is changed and the DRDA connection request is honored.

When a user changes a password, the user ID, the old password, and the new password are sent to DB2 by the client system. The client system can optionally encrypt these three tokens before they are sent.

Detecting authorization failures (EXTENDED SECURITY): If the DB2 server is installed with YES for the EXTENDED SECURITY field of installation panel DSNTIPR, detailed reason codes are returned to a DRDA client when a DDF

connection request fails because of security errors. When using SNA protocols, the requester must have included support for extended security sense codes. One such product is DB2 Connect.

If the proper requester support is present, the requester generates SQLCODE -30082 (SQLSTATE '08001') with a specific indication for the failure. Otherwise, a generic security failure code is returned.

The communications database for the server

The information in this section, up to "Controlling inbound connections that use SNA protocols" on page 242, is General-use Programming Interface and Associated Guidance Information, as defined in "Notices" on page 1301.

The communications database (CDB) is a set of DB2 catalog tables that let you control aspects of how requests leave this DB2 and how requests come in. This section concentrates on the columns of the communications database that pertain to security on the inbound side (the server).

The SYSIBM.IPNAMES table is not described in this section, because that table is not used to control inbound TCP/IP requests.

Columns used in SYSIBM.LUNAMES

This table is used only for requests that use SNA protocols.

LUNAME CHAR(8)
: The LUNAME of the remote system. A blank value identifies a default row that serves requests from any system that is not specifically listed elsewhere in the column.

SECURITY_IN CHAR(1)
: The *acceptance option* for a remote request from the corresponding LUNAME:

V
: The option is "verify." An incoming request must include one of the following authentication entities:
- User ID and password
- User ID and RACF PassTicket, described in "Sending RACF PassTickets" on page 263
- User ID and RACF encrypted password (not recommended)
- Kerberos security tickets, described in "Establishing Kerberos authentication through RACF" on page 278
- User ID and DRDA encrypted password, described in "Sending encrypted passwords from a workstation" on page 264
- User ID, password, and new password, described in "Allowing users to change expired passwords" on page 239
- User ID and encrypted password, or encrypted user ID and encrypted password, described in "Allowing users to change expired passwords" on page 239

A
: The option is "already verified." This is the default. With A, a request does not need an authentication token, although the token is checked if it is sent.

With this option, an incoming connection request is accepted if it includes any of the following authentication tokens:
- User ID only

- All authentication methods that option V supports

If the USERNAMES column of SYSIBM.LUNAMES contains I or B, RACF is not invoked to validate incoming connection requests that contain only a user ID.

ENCRYPTPSWDS CHAR(1)
This column only applies to DB2 UDB for z/OS or DB2 UDB for z/OS partners when passwords are used as authentication tokens. It indicates whether passwords received from and sent to the corresponding LUNAME are encrypted:

Y Yes, passwords are encrypted. For outbound requests, the encrypted password is extracted from RACF and sent to the server. For inbound requests, the password is treated as if it is encrypted.

N No, passwords are not encrypted. This is the default; any character other than Y is treated as N. Specify N for CONNECT statements that contain a USER parameter.

Recommendation: When you connect to a DB2 UDB for z/OS partner that is at Version 5 or a subsequent release, use RACF PassTickets (SECURITY_OUT='R') instead of using passwords.

USERNAMES CHAR(1)
This column indicates whether an ID accompanying a remote request, sent from or to the corresponding LUNAME, is subject to translation and "come from" checking. When you specify I, O, or B, use the SYSIBM.USERNAMES table to perform the translation.

I An inbound ID is subject to translation.

O An outbound ID, sent to the corresponding LUNAME, is subject to translation.

B Both inbound and outbound IDs are subject to translation.

blank No IDs are translated.

Columns used in SYSIBM.USERNAMES

This table is used by both SNA and TCP/IP connections.

TYPE CHAR(1)
Indicates whether the row is used for inbound or outbound translation:

I The row applies to inbound IDs (not applicable for TCP/IP connections).

O The row applies to outbound IDs.

The field should contain only I or O. Any other character, including blank, causes the row to be ignored.

AUTHID VARCHAR(128)
An authorization ID that is permitted and perhaps translated. If blank, any authorization ID is permitted with the corresponding LINKNAME; all authorization IDs are translated in the same way. Outbound translation is not performed on CONNECT statements that contain an authorization ID for the value of the USER parameter.

LINKNAME CHAR(8)
Identifies the VTAM or TCP/IP network locations that are associated with this row. A blank value in this column indicates that this name translation rule applies to any TCP/IP or SNA partner.

If you specify a nonblank value for this column, one or both of the following situations must be true:

- A row exists in table SYSIBM.LUNAMES that has an LUNAME value that matches the LINKNAME value that appears in this column.
- A row exists in table SYSIBM.IPNAMES that has a LINKNAME value that matches the LINKNAME value that appears in this column.

NEWAUTHID VARCHAR(128)
The translated authorization ID. If blank, no translation occurs.

Controlling inbound connections that use SNA protocols

Requests from a remote LU are subject to two security checks before they come into contact with DB2. Those checks control what LUs can attach to the network and verify the identity of a partner LU.

Finally, DB2 itself imposes several checks before accepting an attachment request.

If using private protocols, the LOCATIONS table controls the locations that can access DB2. To allow a remote location to access DB2, the remote location name must be specified in the SYSIBM.LOCATIONS table. This check is only supported for connections using private protocols.

Controlling what LUs can attach to the network

This check is carried out by VTAM, to prevent an unauthorized LU from attaching to the network and presenting itself to other LUs as an acceptable partner in communication. It requires each LU that attaches to the network to identify itself by a password. If that requirement is in effect for your network, your DB2 subsystem, like every other LU on the network, must:

1. Choose a VTAM password.
2. Code the password with the PRTCT parameter of the VTAM APPL statement, when you define your DB2 to VTAM. The APPL statement is described in detail in Part 3 of *DB2 Installation Guide*.

Verifying a partner LU

This check is carried out by RACF and VTAM, to check the identity of an LU sending a request to your DB2.

Recommendation: Specify partner-LU verification, which requires the following steps:

1. Code VERIFY=REQUIRED on the VTAM APPL statement, when you define your DB2 to VTAM. The APPL statement is described in detail in Part 3 of *DB2 Installation Guide*.
2. Establish a RACF profile for each LU from which you permit a request. For the steps required, see “Enable partner-LU verification” on page 267.

Accepting a remote attachment request

When VTAM has established a conversation for a remote application, that application sends a *remote request*, which is a request to attach to your local DB2. (Do not confuse the *remote* request with a *local* attachment request that comes through one of the DB2 attachment facilities—IMS, CICS, TSO, and so on. A remote attachment request is defined by Systems Network Architecture and LU 6.2 protocols; specifically, it is an SNA Function Management Header 5.)

Thissection tells what security checks you can impose on remote attachment requests.

Conversation-level security: This section assumes that you have defined your DB2 to VTAM with the conversation-level security set to "already verified". (To do that, you coded SECACPT=ALREADYV on the VTAM APPL statement, as described in Part 3 of *DB2 Installation Guide*. That value provides more options than does "conversation" (SECACPT=CONV), which is not recommend.

Steps, tools, and decisions: The steps an attachment request goes through before acceptance allow much flexibility in choosing security checks. Scan Figure 20 on page 245 to see what is possible.

The primary tools for controlling remote attachment requests are entries in tables SYSIBM.LUNAMES and SYSIBM.USERNAMES in the communications database. You need a row in SYSIBM.LUNAMES for each system that sends attachment requests, a dummy row that allows *any* system to send attachment requests, or both. You might need rows in SYSIBM.USERNAMES to permit requests from specific IDs or specific LUNAMES, or to provide translations for permitted IDs.

When planning to control remote requests, answer the questions posed by the following topics for each remote LU that can send a request.

- "Do you permit access?"
- "Do you manage inbound IDs through DB2 or RACF?"
- "Do you trust the partner LU?" on page 244
- "If you use passwords, are they encrypted?" on page 244
- "If you use Kerberos, are users authenticated?" on page 244
- "Do you translate inbound IDs?" on page 247
- "How do you associate inbound IDs with secondary IDs?" on page 249

Do you permit access?

To permit attachment requests from a particular LU, you need a row in your SYSIBM.LUNAMES table. The row must either give the specific LUNAME or it must be a dummy row with the LUNAME blank. (The table can have only one dummy row, which is used by all LUs for which no specific row exists, when making requests.) Without one of those rows, the attachment request is rejected.

Do you manage inbound IDs through DB2 or RACF?

If you manage incoming IDs through RACF, you must register every acceptable ID with RACF, and DB2 must call RACF to process every request. If you manage incoming IDs through RACF, either RACF or Kerberos can authenticate the user. Kerberos cannot be used if you do not have RACF on the system.

If you manage incoming IDs through DB2, you can avoid calls to RACF and can specify acceptance of many IDs by a single row in the SYSIBM.USERNAMES table.

To manage incoming IDs through DB2, put an I in the USERNAMES column of SYSIBM.LUNAMES for the particular LU. (Or, if an O is there already because you are also sending requests *to* that LU, change O to B.) Attachment requests from that LU now go through sign-on processing, and its IDs are subject to translation. (For more information about translating IDs, see "Do you translate inbound IDs?" on page 247.)

To manage incoming IDs through RACF, leave USERNAMES blank for that LU (or leave the O unchanged). Requests from that LU go through connection processing, and its IDs are not subject to translation.

Do you trust the partner LU?

Presumably, RACF has already validated the identity of the other LU (described in "Verifying a partner LU" on page 242). If you trust incoming IDs from that LU, you do not need to validate them by an authentication token. Put an A in the SECURITY_IN column of the row in SYSIBM.LUNAMES that corresponds to the other LU; your acceptance level for requests from that LU is now "already verified". Requests from that LU are accepted without an authentication token. (In order to use this option, you must have defined DB2 to VTAM with SECACPT=ALREADYV, as described in 243.)

If an authentication token **does** accompany a request, DB2 calls RACF to check the authorization ID against it. To require an authentication token from a particular LU, put a V in the SECURITY_IN column in SYSIBM.LUNAMES; your acceptance level for requests from that LU is now "verify". You must also register every acceptable incoming ID and its password with RACF.

Performance considerations: Each request to RACF to validate authentication tokens results in an I/O operation, which has a high performance cost.

Recommendation: To eliminate the I/O, allow RACF to cache security information in VLF. To activate this option, add the IRRACEE class to the end of z/OS VLF member COFVLFxx in SYS1.PARMLIB, as follows:

```
CLASS NAME(IRRACEE)
EMAJ (ACEE)
```

If you use passwords, are they encrypted?

Passwords can be encrypted through:

- RACF using PassTickets, described in "Sending RACF PassTickets" on page 263.
- DRDA password encryption support. DB2 UDB for z/OS as a server supports DRDA encrypted passwords and encrypted user IDs with encrypted passwords. See "Sending encrypted passwords from a workstation" on page 264 for more information.
- The SET ENCRYPTION PASSWORD statement. This encryption method should not be used for distributed access because the unencrypted passwords in the SET ENCRYPTION PASSWORD statement flow from the client to the server. For more information about the SET ENCRYPTION PASSWORD statement, see *DB2 SQL Reference*.

If you use Kerberos, are users authenticated?

If your distributed environment uses Kerberos to manage users and perform user authentication, DB2 UDB for z/OS can use Kerberos security services to authenticate remote users. See "Establishing Kerberos authentication through RACF" on page 278.

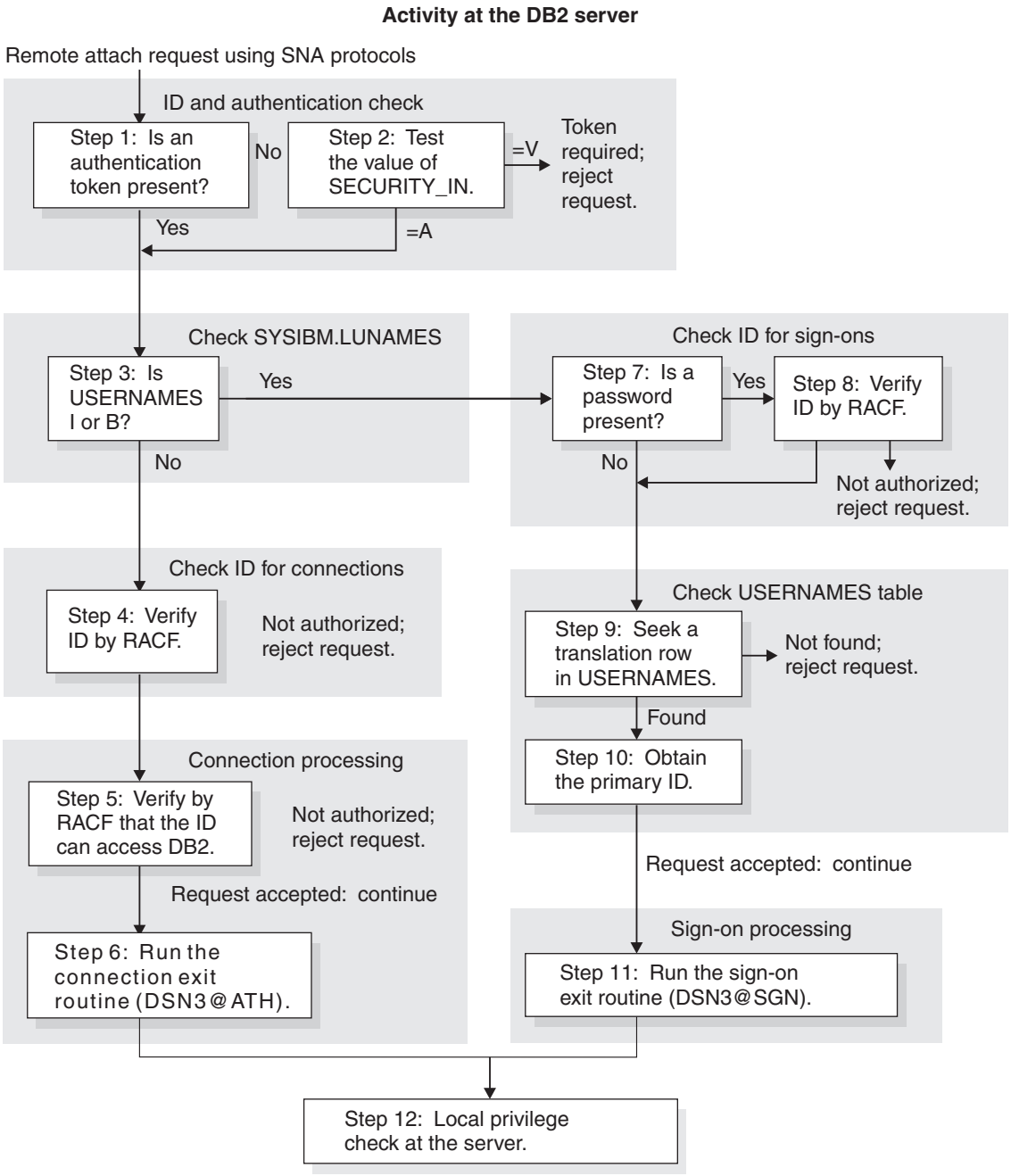


Figure 20. Steps in accepting a remote attachment request from requester that is using SNA

Details of remote attachment request processing:

1. If the remote request has no authentication token, DB2 checks the security acceptance option in the SECURITY_IN column of table SYSIBM.LUNAMES. No password is sent or checked for the plan or package owner that is sent from a DB2 subsystem.
2. If the acceptance option is “verify” (SECURITY_IN = V), a security token is required to authenticate the user. DB2 rejects the request if the token missing.
3. If the USERNAMES column of SYSIBM.LUNAMES contains I or B, the authorization ID, and the plan or package owner that is sent by a DB2

subsystem, are subject to translation under control of the SYSIBM.USERNAMES table. If the request is allowed, it eventually goes through sign-on processing.

If USERNAMES does not contain I or B, the authorization ID is not translated.

4. DB2 calls RACF by the RACROUTE macro with REQUEST=VERIFY to check the ID. DB2 uses the PASSCHK=NO option if no password is specified and ENCRYPT=YES if the ENCRYPTPSWDS column of SYSIBM.LUNAMES contains Y. If the ID, password, or PassTicket cannot be verified, DB2 rejects the request.

   In addition, depending on your RACF environment, the following RACF checks may also be performed:

   - If the RACF APPL class is active, RACF verifies that the ID has been given access to the DB2 APPL. The APPL resource that is checked is the LU name that the requester used when the attachment request was issued. This is either the local DB2 LU name or the generic LU name.
   - If the RACF APPCPORT class is active, RACF verifies that the ID is authorized to access z/OS from the Port of Entry (POE). The POE that RACF uses in the verify call is the requesting LU name.
5. The remote request is now treated like a local connection request with a DIST environment for the DSNR resource class; for details, see "Processing connections" on page 232. DB2 calls RACF by the RACROUTE macro with REQUEST=AUTH, to check whether the authorization ID is allowed to use DB2 resources that are defined to RACF.

   The RACROUTE macro call also verifies that the user is authorized to use DB2 resources from the requesting system, known as the port of entry (POE); for details, see "Allowing access from remote requesters" on page 272.
6. DB2 invokes the connection exit routine. The parameter list that is passed to the routine describes where a remote request originated.
7. If no password exists, RACF is not called. The ID is checked in SYSIBM.USERNAMES.
8. If a password exists, DB2 calls RACF through the RACROUTE macro with REQUEST=VERIFY to verify that the ID is known with the password. ENCRYPT=YES is used if the ENCRYPTPSWDS column of SYSIBM.LUNAMES contains Y. If DB2 cannot verify the ID or password, the request is rejected.
9. DB2 searches SYSIBM.USERNAMES for a row that indicates how to translate the ID. The need for a row that applies to a particular ID and sending location imposes a "come-from" check on the ID: If no such row exists, DB2 rejects the request.
10. If an appropriate row is found, DB2 translates the ID as follows:
    - If a nonblank value of NEWAUTHID exists in the row, that value becomes the primary authorization ID.
    - If NEWAUTHID is blank, the primary authorization ID remains unchanged.
11. The remote request is now treated like a local sign-on request; for details, see "Processing sign-ons" on page 236. DB2 invokes the sign-on exit routine. The parameter list that is passed to the routine describes where a remote request originated. For details, see "Connection routines and sign-on routines" on page 1051.
12. The remote request now has a primary authorization ID, possibly one or more secondary IDs, and an SQL ID. A request from a remote DB2 is also known by a plan or package owner. Privileges and authorities that are granted to those IDs at the DB2 server govern the actions that the request can take.

Do you translate inbound IDs?

Ideally, each of your authorization IDs has the same meaning throughout your entire network. In practice, that might not be so, and the duplication of IDs on different LUs is a security exposure.

Example: Suppose that the ID DBADM1 is known to the local DB2 and has DBADM authority over certain databases there; suppose also that the same ID exists in some remote LU. If an attachment request comes in from DBADM1, and if nothing is done to alter the ID, the wrong user can exercise privileges of DBADM1 in the local DB2. The way to protect against that exposure is to translate the remote ID into a different ID before the attachment request is accepted.

You must be prepared to translate the IDs of plan owners, package owners, and the primary IDs of processes that make remote requests. For the IDs that are sent to you by other DB2 LUs, see "What IDs you send" on page 257. (Do not plan to translate all IDs in the connection exit routine—the routine does not receive plan and package owner IDs.)

If you have decided to manage inbound IDs through DB2, you can translate an inbound ID to some other value. Within DB2, you grant privileges and authorities only to the translated value. As Figure 20 on page 245 shows, that "translation" is not affected by anything you do in your connection or sign-on exit routine. The *output* of the translation becomes the *input* to your sign-on exit routine.

Recommendation: Do not translate inbound IDs in an exit routine; translate them only through the SYSIBM.USERNAMES table.

The examples in Table 73 shows the possibilities for translation and how to control translation by SYSIBM.USERNAMES. You can use entries to allow requests only from particular LUs or particular IDs, or from combinations of an ID and an LU. You can also translate any incoming ID to another value. Table 75 on page 249 shows the search order of the SYSIBM.USERNAMES table.

Performance considerations: In the process of accepting remote attachment requests, any step that calls RACF is likely to have a relatively high performance cost. To trade some of that cost for a somewhat greater security exposure, have RACF check the identity of the other LU just once, as described under "Verifying a partner LU" on page 242. Then trust the partner LU, translating the inbound IDs and not requiring or using passwords. In this case, no calls are made to RACF from within DB2; the penalty is only that you make the partner LU responsible for verifying IDs.

Update considerations: If you update tables in the CDB while the distributed data facility is running, the changes might not take effect immediately. For details, see Part 3 of *DB2 Installation Guide*.

Example: Table 73 shows how USERNAMES translates inbound IDs.

Table 73. Your SYSIBM.USERNAMES table. (Row numbers are added for reference.)

Row	TYPE	AUTHID	LINKNAME	NEWAUTHID
1	I	blank	LUSNFRAN	blank
2	I	BETTY	LUSNFRAN	ELIZA
3	I	CHARLES	blank	CHUCK
4	I	ALBERT	LUDALLAS	blank

Table 73. Your SYSIBM.USERNAMES table (continued). (Row numbers are added for reference.)

Row	TYPE	AUTHID	LINKNAME	NEWAUTHID
5	I	BETTY	blank	blank

DB2 searches SYSIBM.USERNAMES to determine how to translate for each of the requests that are listed in Table 74.

Table 74. How DB2 translates inbound authorization ids

Request	How DB2 translates request
ALBERT requests from LUDALLAS	DB2 searches for an entry for AUTHID=ALBERT and LINKNAME=LUDALLAS. DB2 finds one in row 4, so the request is accepted. The value of NEWAUTHID in that row is blank, so ALBERT is left unchanged.
BETTY requests from LUDALLAS	DB2 searches for an entry for AUTHID=BETTY and LINKNAME=LUDALLAS; none exists. DB2 then searches for AUTHID=BETTY and LINKNAME=blank. It finds that entry in row 5, so the request is accepted. The value of NEWAUTHID in that row is blank, so BETTY is left unchanged.
CHARLES requests from LUDALLAS	DB2 searches for AUTHID=CHARLES and LINKNAME=LUDALLAS; no such entry exists. DB2 then searches for AUTHID=CHARLES and LINKNAME=blank. The search ends at row 3; the request is accepted. The value of NEWAUTHID in that row is CHUCK, so CHARLES is translated to CHUCK.
ALBERT requests from LUSNFRAN	DB2 searches for AUTHID=ALBERT and LINKNAME=LUSNFRAN; no such entry exists. DB2 then searches for AUTHID=ALBERT and LINKNAME=blank; again no entry exists. Finally, DB2 searches for AUTHID=blank and LINKNAME=LUSNFRAN, finds that entry in row 1, and the request is accepted. The value of NEWAUTHID in that row is blank, so ALBERT is left unchanged.
BETTY requests from LUSNFRAN	DB2 finds row 2, and BETTY is translated to ELIZA.
CHARLES requests from LUSNFRAN	DB2 finds row 3 before row 1; CHARLES is translated to CHUCK.
WILBUR requests from LUSNFRAN	No provision is made for WILBUR, but row 1 of the SYSIBM.USERNAMES table allows any ID to make a request from LUSNFRAN and to pass without translation. The acceptance level for LUSNFRAN is “already verified”, so WILBUR can pass without a password check by RACF. After accessing DB2, WILBUR can use only the privileges that are granted to WILBUR and to PUBLIC (for DRDA access) or to PUBLIC AT ALL LOCATIONS (for DB2 private-protocol access).
WILBUR requests from LUDALLAS	Because the acceptance level for LUDALLAS is “verify” as recorded in the SYSIBM.LUNAMES table, WILBUR must be known to the local RACF. DB2 searches in succession for one of the combinations WILBUR/LUDALLAS, WILBUR/blank, or blank/LUDALLAS. None of those is in the table, so the request is rejected. The absence of a row permitting WILBUR to request from LUDALLAS imposes a “come-from” check: WILBUR can attach from some locations (LUSNFRAN), and some IDs (ALBERT, BETTY, and CHARLES) can attach from LUDALLAS, but WILBUR cannot attach if coming from LUDALLAS.

Table 75 shows the search order for the SYSIBM.USERNAMES table:

Table 75. Precedence search order for SYSIBM.USERNAMES table

AUTHID	LINKNAME	Result
Name	Name	If NEWAUTHID is specified, AUTHID is translated to NEWAUTHID for the specified LINKNAME.
Name	Blank	If NEWAUTHID is specified, AUTHID is translated to NEWAUTHID for all LINKNAMEs.
Blank	Name	If NEWAUTHID is specified, it is substituted for AUTHID for the specified LINKNAME.
Blank	Blank	Unavailable resource message (SQLCODE -904) is returned.

Encryption considerations: If incoming authorization IDs are managed through DB2 and if the ICSF is installed and properly configured, you can use the DSNLEUSR stored procedure to encrypt translated authorization IDs and store them in the NEWAUTHID column of the SYSIBM.USERNAMES table. DB2 decrypts the translated authorization IDs during connection processing. For more information about the DSNLEUSR stored procedure, see Appendix J, "DB2-supplied stored procedures," on page 1255.

How do you associate inbound IDs with secondary IDs?

Your decisions on the previous questions determine what value is used for the primary authorization ID on an attachment request. They also determine whether those requests are next treated as connection requests or as sign-on requests. That means that the remote request next goes through the same processing as a local request, and that you have the opportunity to associate the primary ID with a list of secondary IDs in the same way you do for local requests. For more information about processing connections and sign-ons, see "Processing connections" on page 232 and "Processing sign-ons" on page 236.

Controlling inbound connections that use TCP/IP protocols

DRDA connections that use TCP/IP have fewer security controls than do connections that use SNA protocols. When planning to control inbound TCP/IP connections, consider the following issues:

Do you permit access by TCP/IP? If the serving DB2 UDB for z/OS subsystem has a DRDA port and resynchronization port specified in the BSDS, DB2 is enabled for TCP/IP connections.

Do you manage inbound IDs through DB2 or RACF? All IDs must be passed to RACF or Kerberos for processing. No option exists to handle incoming IDs through DB2.

Do you trust the partner? TCP/IP does not verify partner LUs as SNA does. If your requesters support mutual authentication, use Kerberos to handle this on the requester side.

If you use passwords, are they encrypted? Passwords can be encrypted through:

- RACF using PassTickets, described in "Sending RACF PassTickets" on page 263.

- DRDA password encryption support. DB2 UDB for z/OS as a server supports DRDA encrypted passwords and encrypted user IDs with encrypted passwords. See "Sending encrypted passwords from a workstation" on page 264 for more information.

If you use Kerberos, are users authenticated? If your distributed environment uses Kerberos to manage users and perform user authentication, DB2 UDB for z/OS can use Kerberos security services to authenticate remote users. See "Establishing Kerberos authentication through RACF" on page 278.

Do you translate inbound IDs? Inbound IDs are not translated when you use TCP/IP.

How do you associate inbound IDs with secondary IDs? To associate an inbound ID with secondary IDs, modify the default connection exit routine (DSN3@ATH). TCP/IP requests do not use the sign-on exit routine.

Steps, tools, and decisions

See Figure 21 on page 251 for an overview of how incoming requests are handled. See "Detecting authorization failures (EXTENDED SECURITY)" on page 239 for information about security diagnostics.

1. You must first decide whether you want incoming requests to have authentication information, such as RACF passwords, RACF PassTickets, and Kerberos tickets, passed along with the authorization ID.

   To indicate that you require this authentication information, specify NO on the TCP/IP ALREADY VERIFIED field of installation panel DSNTIP5, which is the default option. If you do not specify NO, all incoming TCP/IP requests can connect to DB2 without any authentication.
2. If you require authentication, ensure that the security subsystem at your server is properly configured to handle the authentication information that is passed to it.
   - For requests that use RACF passwords or PassTickets, enter the following RACF command to indicate which user IDs that use TCP/IP are authorized to access DDF (the distributed data facility address space):

     ```
     PERMIT ssnm.DIST CLASS(DSNR) ID(yyy) ACCESS(READ)
       WHEN(APPCPORT(TCPIP))
     ```

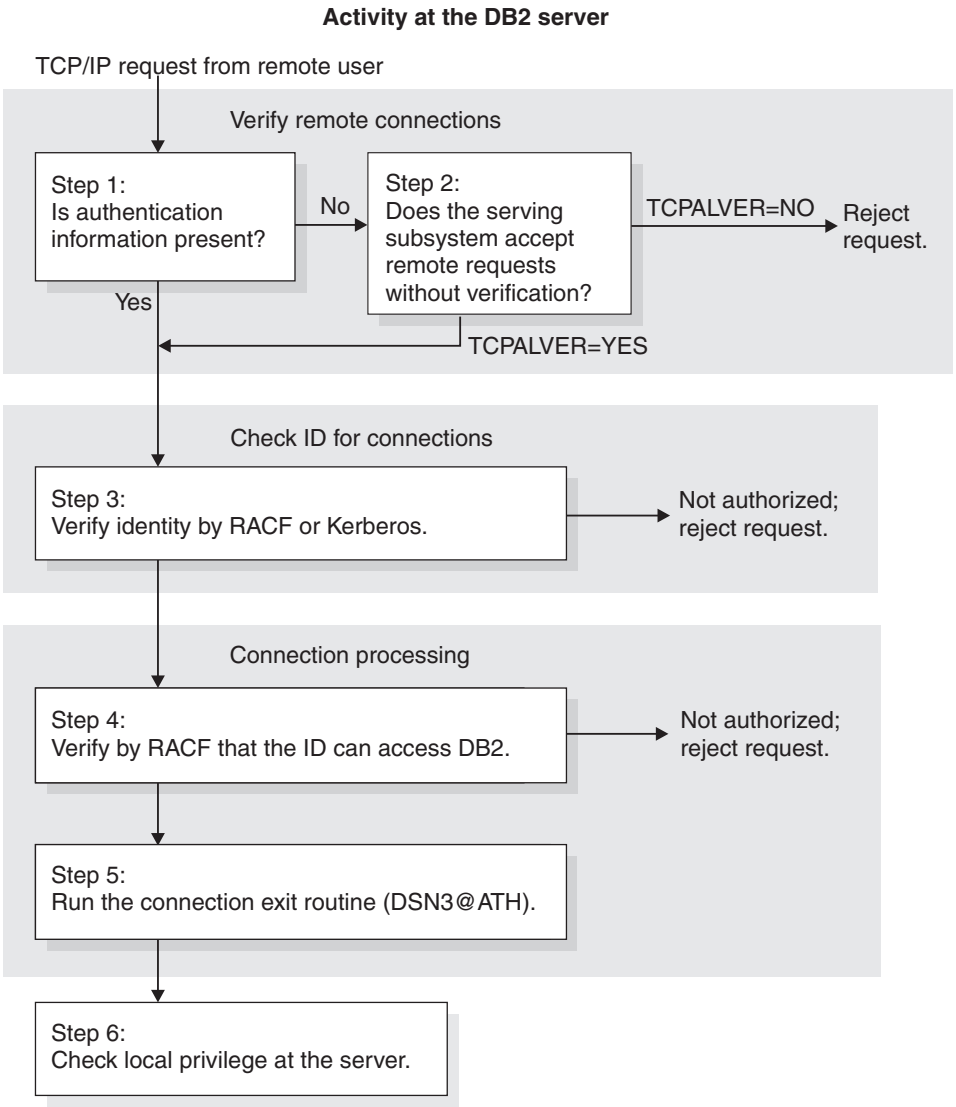


Figure 21. Steps in accepting a request from TCP/IP.

Details of steps: These notes explain the steps shown in Figure 21.

1. DB2 checks to see if an authentication token (RACF encrypted password, RACF PassTicket, DRDA encrypted password, or Kerberos ticket) accompanies the remote request.
2. If no authentication token is supplied, DB2 checks the TCPALVER subsystem parameter to see if DB2 accepts IDs without authentication information. If TCPALVER=NO, authentication information must accompany all requests, and DB2 rejects the request. If TCPALVER=YES, DB2 accepts the request without authentication.
3. The identity is a RACF ID that is authenticated by RACF if a password or PassTicket is provided, or the identity is a Kerberos principal that is validated by Kerberos Security Server, if a Kerberos ticket is provided. Ensure that the ID is defined to RACF in all cases. When Kerberos tickets are used, the RACF ID is derived from the Kerberos principal identity. To use Kerberos tickets, ensure that you map Kerberos principal names with RACF IDs, as described in "Establishing Kerberos authentication through RACF" on page 278.

   In addition, depending on your RACF environment, the following RACF checks may also be performed:

- If the RACF APPL class is active, RACF verifies that the ID has access to the DB2 APPL. The APPL resource that is checked is the LU name that the requester used when the attachment request was issued. This is either the local DB2 LU name or the generic LU name.
- If the RACF APPCPORT class is active, RACF verifies that the ID is authorized to access z/OS from the port of entry (POE). The POE that RACF uses in the RACROUTE VERIFY call depends on whether all the following conditions are true:
  - The current operating system is z/OS V1.5 or later
  - The TCP/IP Network Access Control is configured
  - The RACF SERVAUTH class is active

  If all these conditions are true, RACF uses the remote client's POE security zone name that is defined in the TCP/IP Network Access Control file. If one or more of these conditions is not true, RACF uses the literal string 'TCPIP'.

  If this is a request to change a password, the password is changed.

  For more information about the RACF SERVAUTH class, see *z/OS V1.5 Security Server RACF Security Administrator's Guide*. For more information about TCP/IP Network Access Control, see*z/OS Communications Server: IP Configuration Guide* and "Allowing access from remote requesters" on page 272.

4. The remote request is now treated like a local connection request (using the DIST environment for the DSNR resource class). DB2 calls RACF to check the ID's authorization against the *ssnm*.DIST resource.
5. DB2 invokes the connection exit routine. The parameter list that is passed to the routine describes where the remote request originated.
6. The remote request has a primary authorization ID, possibly one or more secondary IDs, and an SQL ID. (The SQL ID cannot be translated.) The plan or package owner ID also accompanies the request. Privileges and authorities that are granted to those IDs at the DB2 server govern the actions that the request can take.

Planning to send remote requests

If you are planning to send requests to another DB2 subsystem, consider that the security administrator of that subsystem might have chosen any of the options described in "Controlling requests from remote applications" on page 238. You need to know what those choices are and make entries in your CDB to correspond to them. You can also choose some things independently of what the other subsystem requires.

If you are planning to send remote requests to a DBMS that is not DB2 UDB for z/OS, you need to satisfy the requirements of that system. You probably need documentation for the particular type of system; some of the choices that are described in this section might not apply.

Network protocols and authentication tokens: DB2 chooses how to send authentication tokens based on the network protocols that are used (SNA or TCP/IP). If the request is sent using SNA, the authentication tokens are sent in the SNA attachment request (FMH5), unless you are using Kerberos. If you use Kerberos, authentication tokens are sent with DRDA security commands.

If the request uses TCP/IP, the authentication tokens are always sent using DRDA security commands.

The communications database for the requester

The information in this section, up to “What IDs you send” on page 257, is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page 1301.

The communications database (CDB) is a set of DB2 catalog tables that let you control aspects of remote requests. This section concentrates on the columns of the communications database that pertain to security issues related to the requesting system. (SYSIBM.IPLIST is not related to security and therefore not described in this topic.)

Columns used in SYSIBM.LUNAMES

This table is used only for requests that use SNA protocols.

LUNAME CHAR(8)
The LUNAME of the remote system. A blank value identifies a default row that serves requests from any system that is not specifically listed elsewhere in the column.

SECURITY_OUT (CHAR 1)
Indicates the security option that is used when local DB2 SQL applications connect to any remote server that is associated with the corresponding LUNAME.

A The letter A signifies the security option of already verified, and it is the default. With A, outbound connection requests contain an authorization ID and no authentication token. The value that is used for an outbound request is either the DB2 user's authorization ID or a translated ID, depending on the value in the USERNAMES column.

R The letter R signifies the RACF PassTicket security option. Outbound connection requests contain a user ID and a RACF PassTicket. The LUNAME column is used as the RACF PassTicket application name.

The value that is used for an outbound request is either the DB2 user's authorization ID or a translated ID, depending on the value in the USERNAMES column. The translated ID is used to build the RACF PassTicket. Do not specify R for CONNECT statements with a USER parameter.

P The letter P signifies the password security option. Outbound connection requests contain an authorization ID and a password. The password is obtained from RACF if ENCRYPTPSWDS=Y, or from SYSIBM.USERNAMES if ENCRYPTPSWDS=N. If you get the password from SYSIBM.USERNAMES, the USERNAMES column of SYSIBM.LUNAMES must contain B or O. The value that is used for an outbound request is the translated ID.

ENCRYPTPSWDS CHAR(1)
Indicates whether passwords received from and sent to the corresponding LUNAME are encrypted. This column only applies to DB2 UDB for z/OS and DB2 UDB for z/OS partners when passwords are used as authentication tokens.

Y Yes, passwords are encrypted. For outbound requests, the encrypted password is extracted from RACF and sent to the server. For inbound requests, the password is treated as encrypted.

N No, passwords are not encrypted. This is the default; any character but Y is treated as N.

Recommendation: When you connect to a DB2 UDB for z/OS partner that is at Version 5 or a subsequent release, use RACF PassTickets (SECURITY_OUT='R') instead of encrypting passwords.

USERNAMES CHAR(1)
Indicates whether an ID accompanying a remote attachment request, which is received from or sent to the corresponding LUNAME, is subject to translation and "come from" checking. When you specify I, O, or B, use the SYSIBM.USERNAMES table to perform the translation.

I An inbound ID is subject to translation.

O An outbound ID, sent to the corresponding LUNAME, is subject to translation.

B Both inbound and outbound IDs are subject to translation.

blank No IDs are translated.

Columns used in SYSIBM.IPNAMES

This table is used only for requests that use TCP/IP protocols.

LINKNAME CHAR(8)
The name used in the LINKNAME column of SYSIBM.LOCATIONS to identify the remote system.

SECURITY_OUT
Indicates the DRDA security option that is used when local DB2 SQL applications connect to any remote server that is associated with this TCP/IP host.

A The letter A signifies the security option of already verified, and it is the default. Outbound connection requests contain an authorization ID and no password. The value that is used for an outbound request is either the DB2 user's authorization ID or a translated ID, depending on the value in the USERNAMES column.

The authorization ID is not encrypted when it is sent to the partner. For encryption, see option D.

R The letter R signifies the RACF PassTicket security option. Outbound connection requests contain a user ID and a RACF PassTicket. The LINKNAME column must contain the server's LU name, which is used as the RACF PassTicket application name to generate the PassTicket.

The value that is used for an outbound request is either the DB2 user's authorization ID or a translated ID, depending on the value in the USERNAMES column. The translated ID is used to build the RACF PassTicket. Do not specify R for CONNECT statements with a USER parameter.

The authorization ID is not encrypted when it is sent to the partner.

D The letter D signifies the security option of user ID and security-sensitive data encryption. Outbound connection requests contain an authorization ID and no password. The authorization

ID that is used for an outbound request is either the DB2 user's authorization ID or a translated ID, depending on the USERNAMES column.

This option indicates that the user ID and the security-sensitive data are to be encrypted. If you do not require encryption, see option A.

E The letter E signifies the security option of user ID, password, and security-sensitive data encryption. Outbound connection requests contain an authorization ID and a password. The password is obtained from the SYSIBM.USERNAMES table. The USERNAMES column must specify "O".

This option indicates that the user ID, password, and security-sensitive data are to be encrypted. If you do not require security-sensitive data encryption, see option P.

P The letter P signifies the password security option. Outbound connection requests contain an authorization ID and a password. The password is obtained from the SYSIBM.USERNAMES table. If you specify P, the USERNAMES column must specify "O".

If you specify P and the server supports encryption, the user ID and the password are encrypted. If the server does not support encryption, the user ID and the password are sent to the partner in clear text. If you also need to encrypt security-sensitive data, see option E.

USERNAMES CHAR(1)
This column indicates whether an outbound request translates the authorization ID. When you specify O, use the SYSIBM.USERNAMES table to perform the translation.

O An outbound ID, sent to the corresponding LUNAME, is subject to translation.

blank No translation is done.

Columns used in SYSIBM.USERNAMES

This table is used by both SNA and TCP/IP connections.

TYPE CHAR(1)
Indicates whether the row is used for inbound or outbound translation:

I The row applies to inbound IDs.

O The row applies to outbound IDs.

The field should contain only I or O. Any other character, including blank, causes the row to be ignored.

AUTHID VARCHAR(128)
An authorization ID that is permitted and perhaps translated. If blank, any authorization ID is permitted with the corresponding LINKNAME, and all authorization IDs are translated in the same way.

LINKNAME CHAR(8)
Identifies the VTAM or TCP/IP network locations that are associated with this row. A blank value in this column indicates that this name translation rule applies to any TCP/IP or SNA partner.

If you specify a nonblank value for this column, one or both of the following situations must be true:

- A row exists in table SYSIBM.LUNAMES that has an LUNAME value that matches the LINKNAME value that appears in this column.
- A row exists in table SYSIBM.IPNAMES that has a LINKNAME value that matches the LINKNAME value that appears in this column.

NEWAUTHID VARCHAR(128)
The translated authorization ID. If blank, no translation occurs.

PASSWORD CHAR(8)
A password that is sent with outbound requests. This password is not provided by RACF and cannot be encrypted.

Columns used in SYSIBM.LOCATIONS

This table contains a row for every accessible remote server. Each row associates a LOCATION name with the TCP/IP or SNA network attributes for the remote server. Requesters are not defined in this table.

LOCATION CHAR(16)
Indicates the unique location name by which the the remote server is known to local DB2 SQL applications.

LINKNAME CHAR(8)
Identifies the VTAM or TCP/IP network locations that are associated with this row. A blank value in this column indicates that this name translation rule applies to any TCP/IP or SNA partner.

If you specify a nonblank value for this column, one or both of the following situations must be true:

- A row exists in table SYSIBM.LUNAMES that has an LUNAME value that matches the LINKNAME value that appears in this column.
- A row exists in table SYSIBM.IPNAMES that has a LINKNAME value that matches the LINKNAME value that appears in this column.

PORT CHAR(32)
TCP/IP is used for outbound DRDA connections when the following statement is true:

- A row exists in SYSIBM.IPNAMES, where the LINKNAME column matches the value specified in the SYSIBM.LOCATIONS LINKNAME column.

If the previously mentioned row is found, the value of the PORT column is interpreted as follows:

- If PORT is blank, the default DRDA port (446)is used.
- If PORT is nonblank, the value specified for PORT can take one of two forms:
  - If the value in PORT is left justified with one to five numeric characters, the value is assumed to be the TCP/IP port number of the remote database server.
  - Any other value is assumed to be a TCP/IP service name, which can be converted to a TCP/IP port number using the TCP/IP getservbyname socket call. TCP/IP service names are not case-sensitive.

TPN VARCHAR(64)
Used only when the local DB2 begins an SNA conversation with another server. When used, TPN indicates the SNA LU 6.2 transaction program

name (TPN) that will allocate the conversation. A length of zero for the column indicates the default TPN. For DRDA conversations, this is the DRDA default, which is X'07F6C4C2'.

For DB2 private protocol conversations, this column is not used. For an SQL/DS™ server, TPN should contain the resource ID of the SQL/DS machine.

DBALIAS(128)
This name is used to access a remote database server. If DBALIAS is blank, the location name is used to access the remote database server. This column does not change the name of any database objects sent to the remote site that contains the location qualifier.

What IDs you send

At least one authorization ID is always sent to the server to be used for authentication. That ID is one of the following values:

- The primary authorization ID of the process.
- If you connect to the server using a CONNECT statement with the USER keyword, the ID that you specify as the USER ID. The CONNECT statement allows non-RACF user IDs on the USER keyword. If connecting to a remote location, the user ID is not authenticated by RACF.

However, other IDs can accompany some requests. You need to understand what other IDs are sent because they are subject to translation. You must include these other IDs in table SYSIBM.USERNAMES to avoid an error when you use outbound translation. Table 76 shows the IDs you need to send in different situations.

Table 76. IDs that accompany the primary ID on a remote request

In this situation:	You send this ID also:
An SQL query, using DB2 private-protocol or DRDA-protocol access	The plan owner
A remote BIND, COPY, or REBIND PACKAGE command	The package owner

For DRDA, if you use the SYSIBM.USERNAMES table that contains the plan owner ID, the plan owner ID is sent to the z/OS server as part of its accounting data. If the connection is to a remote non-DB2 for z/OS server using DRDA protocol and if the outbound translation is specified, a row for the plan owner in the USERNAMES table is optional.

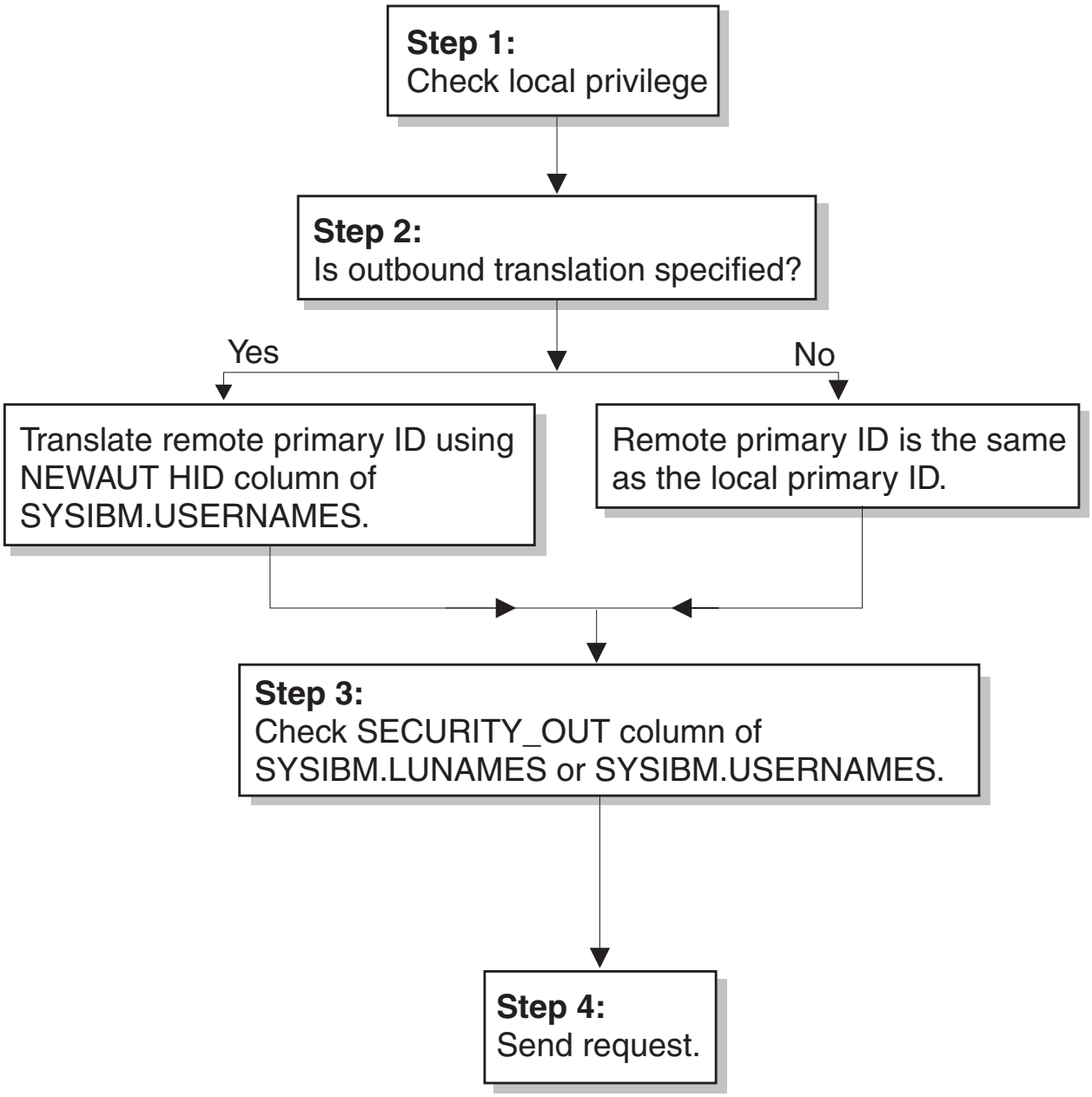


Figure 22. Steps in sending a request from a DB2 subsystem

Details of steps in sending a request from DB2: These notes explain the steps in Figure 22.

1. The DB2 subsystem that sends the request checks whether the primary authorization ID has the privilege to execute the plan or package.

   DB2 determines which value in the LINKNAME column of the SYSIBM.LOCATIONS table matches either the LUNAME column in the SYSIBM.LUNAMES table or the LINKNAME column in the SYSIBM.IPNAMES table. This check determines whether SNA or TCP/IP protocols are used to carry the DRDA request. (Statements that use DB2 private protocol, not DRDA, always use SNA.)
2. When a plan is executed, the authorization ID of the plan owner is sent with the primary authorization ID. When a package is bound, the authorization ID of the package owner is sent with the primary authorization ID. If the

USERNAMES column of the SYSIBM.LUNAMES table contains O or B, or if the USERNAMES column of the SYSIBM.IPNAMES table contains O, both IDs are subject to translation under control of the SYSIBM.USERNAMES table. Ensure that these IDs are included in SYSIBM.USERNAMES, or SQLCODE -904 is issued. DB2 translates the ID as follows:

- If a nonblank value of NEWAUTHID is in the row, that value becomes the new ID.
- If NEWAUTHID is blank, the ID is not changed.

If the SYSIBM.USERNAMES table does not contain a new authorization ID to which the primary authorization ID is translated, the request is rejected with SQLCODE -904.

If the USERNAMES column does not contain O or B, the IDs are not translated.

3. SECURITY_OUT is checked for outbound security options, as follows. Figure 23 on page 260 illustrates this step.

   A Already verified. No password is sent with the authorization ID. This option is valid only if the server accepts already verified requests.
   - For SNA, the server must have specified A in the SECURITY_IN column of SYSIBM.LUNAMES.
   - For TCP/IP, the server must have specified YES in the TCP/IP ALREADY VERIFIED field of installation panel DSNTIP5.

   R RACF PassTicket. If the primary authorization ID was translated, that translated ID is sent with the PassTicket. See "Sending RACF PassTickets" on page 263 for information about setting up PassTickets.

   P Password. The outbound request must be accompanied by a password:
   - If the requester is DB2 UDB for z/OS and uses SNA protocols, passwords can be encrypted if you specify Y in the ENCRYPTPSWDS column of SYSIBM.LUNAMES. If passwords are encrypted, the password is obtained from RACF. If passwords are not encrypted, the password is obtained from the PASSWORD column of SYSIBM.USERNAMES.
   - If the requester uses TCP/IP protocols, the password is obtained from the PASSWORD column of SYSIBM.USERNAMES. If the Integrated Cryptographic Service Facility is enabled and properly configured and the server supports encryption, the password is encrypted.

     Recommendation: Use RACF PassTickets to avoid sending unencrypted passwords over the network.

   D User ID and security-sensitive data encryption. No password is sent with the authorization ID. If the Integrated Cryptographic Service Facility (ICSF) is enabled and properly configured and the server supports encryption, the authorization ID is encrypted before it is sent. If the ICSF is not enabled or properly configured, SQL return code –904 is returned. If the server does not support encryption, SQL return code –30082 is returned.

   E User ID, password, and security-sensitive data encryption. If the ICSF is enabled and properly configured and the server supports encryption, the password is encrypted before it is sent. If the ICSF is not enabled or properly configured, SQL return code –904 is returned. If the server does not support encryption, SQL return code –30082 is returned.

4. Send the request. See Table 76 on page 257 to determine which IDs accompany the primary authorization ID.

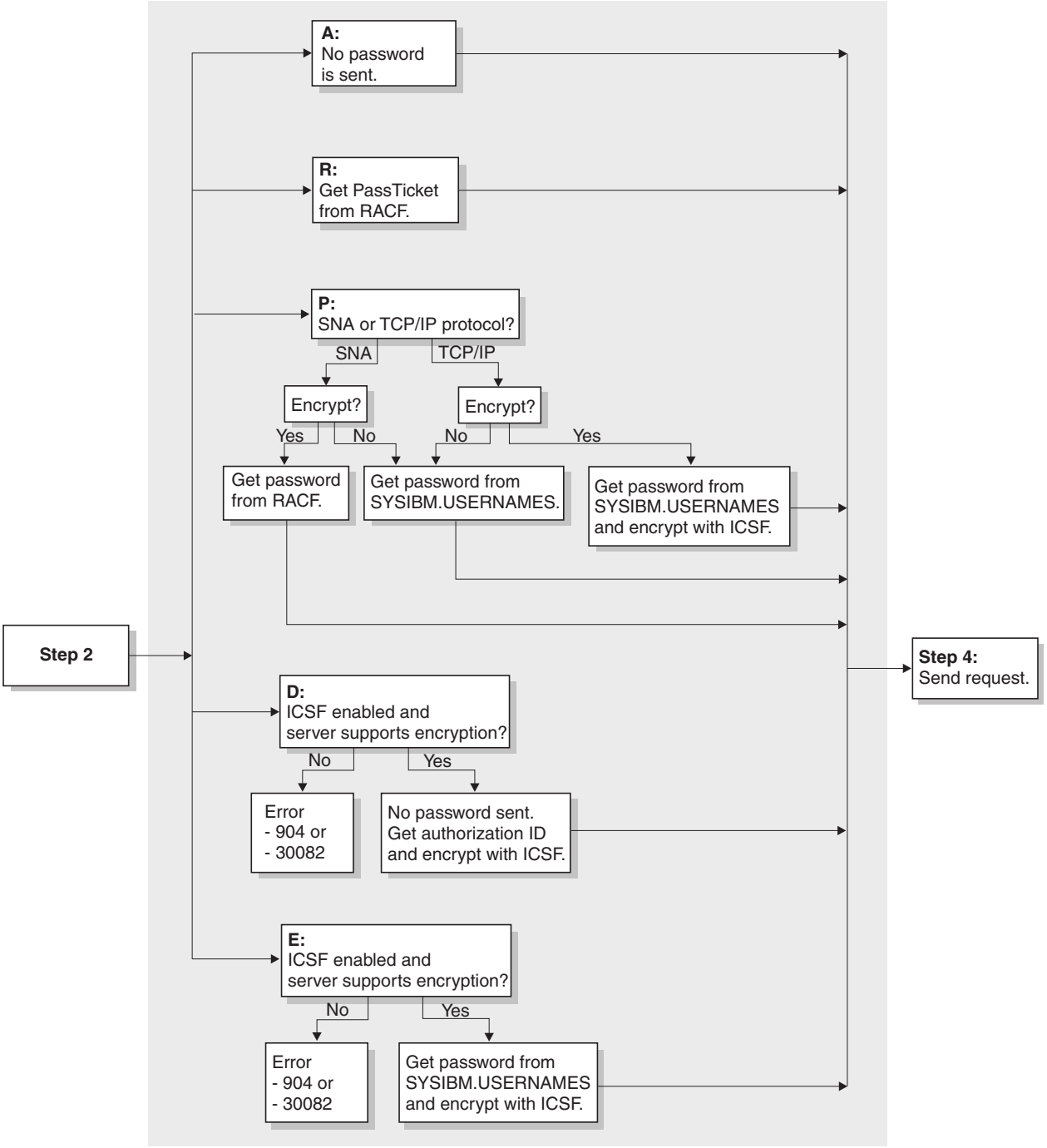


Figure 23. Details of step 3

Translating outbound IDs

If an ID on your system is duplicated on a remote system, you can translate outbound IDs to avoid confusion. You can also translate IDs to ensure that they are accepted by the remote system.

To indicate that you want to translate outbound user IDs, perform the following steps:

1. Specify an O in the USERNAMES column of table SYSIBM.IPNAMES or SYSIBM.LUNAMES.
2. Use the NEWAUTHID column of SYSIBM.USERNAMES to specify the ID to which the outbound ID is translated.

Example 1: Suppose that the remote system accepts from you only the IDs XXGALE, GROUP1, and HOMER.

1. Specify that outbound translation is in effect for the remote system LUXXX by specifying in SYSIBM.LUNAMES the values that are shown in Table 77.

Table 77. SYSIBM.LUNAMES to specify that outbound translation is in effect for the remote system LUXXX

LUNAME	USERNAMES
LUXX	O

If your row for LUXXX already has I for the USERNAMES column (because you translate inbound IDs that come from LUXXX), change I to B for both inbound and outbound translation.

2. Translate the ID GALE to XXGALE on all outbound requests to LUXXX by specifying in SYSIBM.USERNAMES the values that are shown in Table 78.

Table 78. Values in SYSIBM. USERNAMES to translate GALE to XXGALE on outbound requests to LUXXX

TYPE	AUTHID	LINKNAME	NEWAUTHID	PASSWORD
O	GALE	LUXX	XXGALE	GALEPASS

3. Translate EVAN and FRED to GROUP1 on all outbound requests to LUXXX by specifying in SYSIBM.USERNAMES the values that are shown in Table 79.

Table 79. Values in SYSIBM. USERNAMES to translate EVAN and FRED to GROUP1 on outbound requests to LUXXX

TYPE	AUTHID	LINKNAME	NEWAUTHID	PASSWORD
O	EVAN	LUXXX	GROUP1	GRP1PASS
O	FRED	LUXXX	GROUP1	GRP1PASS

4. Do not translate the ID HOMER on outbound requests to LUXXX. (HOMER is assumed to be an ID on your DB2, and on LUXXX.) Specify in SYSIBM.USERNAMES the values that are shown in Table 80.

Table 80. Values in SYSIBM. USERNAMES to not translate HOMER on outbound requests to LUXXX

TYPE	AUTHID	LINKNAME	NEWAUTHID	PASSWORD
O	HOMER	LUXXX	(blank)	HOMERSPW

5. Reject any requests from BASIL to LUXXX before they are sent. To do that, leave SYSIBM.USERNAMES empty. If no row indicates what to do with the ID BASIL on an outbound request to LUXXX, the request is rejected.

Example 2: If you send requests to another LU, such as LUYYY, you generally need another set of rows to indicate how your IDs are to be translated on outbound requests to LUYYY.

However, you can use a single row to specify a translation that is to be in effect on requests to all other LUs. For example, if HOMER is to be sent untranslated everywhere, and DOROTHY is to be translated to GROUP1 everywhere, specify in SYSIBM.USERNAMES the values that are shown in Table 81.

Table 81. Values in SYSIBM. USERNAMES to not translate HOMER and to translate DOROTHY to GROUP1

TYPE	AUTHID	LINKNAME	NEWAUTHID	PASSWORD
O	HOMER	(blank)	(blank)	HOMERSPW
O	DOROTHY	(blank)	GROUP1	GRP1PASS

You can also use a single row to specify that all IDs that accompany requests to a single remote system must be translated. For example, if every one of your IDs is to be translated to THEIRS on requests to LUYYY, specify in SYSIBM.USERNAMES the values that are shown in Table 82.

Table 82. Values in SYSIBM. USERNAMES to translate every ID to THEIRS

TYPE	AUTHID	LINKNAME	NEWAUTHID	PASSWORD
O	(blank)	LUYYY	THEIR	THEPASS

If the ICSF is installed and properly configured, you can use the DSNLEUSR stored procedure to encrypt the translated outbound IDs that are specified in the NEWAUTHID column of SYSIBM.USERNAMES. DB2 decrypts the translated outbound IDs during connection processing. For more information about the DSNLEUSR stored procedure, see Appendix J, “DB2-supplied stored procedures,” on page 1255.

Sending passwords

Recommendation: For the tightest security, do not send passwords through the network. Instead, use one of the following security mechanisms:

- RACF encrypted passwords, described in “Sending RACF encrypted passwords” on page 263
- RACF PassTickets, described in “Sending RACF PassTickets” on page 263
- Kerberos tickets, described in “Establishing Kerberos authentication through RACF” on page 278
- DRDA encrypted passwords or DRDA encrypted user IDs with encrypted passwords, described in “Sending encrypted passwords from a workstation” on page 264

If send passwords through the network, you can put the password for an ID in the PASSWORD column of SYSIBM.USERNAMES.

Recommendation: Use the DSNLEUSR stored procedure to encrypt passwords in SYSIBM.USERNAMES. If the ICSF is installed and properly configured, you can use the DSNLEUSR stored procedure to encrypt passwords in the SYSIBM.USERNAMES table. DB2 decrypts the password during connection processing. For more information about the DSNLEUSR stored procedure, see Appendix J, “DB2-supplied stored procedures,” on page 1255.

DB2 UDB for z/OS allows the use of RACF encrypted passwords or RACF PassTickets. However, workstations, such as Windows workstations, do not support these security mechanisms. RACF encrypted passwords are not a secure mechanism because they can be replayed.

Recommendation: Do not use RACF encrypted passwords unless you are connecting to a previous release of DB2 UDB for z/OS.

Sending RACF encrypted passwords

For DB2 subsystems that communicate with each other by using SNA protocols, you can specify password encryption in SYSIBM.LUNAMES, as shown in Table 83.

Table 83. Specifying password encryption in SYSIBM.LUNAMES

LUNAME	USERNAMES	ENCRYPTPSWDS
LUXXX	O	Y

The partner DB2 must also specify password encryption in its SYSIBM.LUNAMES table. Both partners must register each ID and its password with RACF. Then, for every request to LUXXX, your DB2 calls RACF to supply an encrypted password to accompany the ID. With password encryption, you do not use the PASSWORD column of SYSIBM.USERNAMES, so the security of that table becomes less critical.

Sending RACF PassTickets

To send RACF PassTickets with your remote requests to a particular remote system, enter R in the SECURITY_OUT column of the SYSIBM.IPNAMES or SYSIBM.LUNAMES table for that system.

To set up RACF to generate PassTickets, define and activate a RACF PassTicket data class (PTKTDATA). This class must contain a RACF profile for each remote DB2 subsystem to which you send requests.

1. Activate the RACF PTKTDATA class by issuing the following RACF commands:

   ```
   SETROPTS CLASSACT(PTKTDATA)
   SETROPTS RACLIST(PTKTDATA)
   ```

2. Define profiles for the remote systems by entering the name of each remote system as it appears in the LINKNAME column of table SYSIBM.LOCATIONS. For example, the following command defines a profile for a remote system, DB2A, in the RACF PTKTDATA class:

   ```
   RDEFINE PTKTDATA DB2A SSIGNON(KEYMASKED(E001193519561977))
   ```

3. Refresh the RACF PTKTDATA definition with the new profile by issuing the following command:

   ```
   SETROPTS RACLIST(PTKTDATA) REFRESH
   ```

See *z/OS Security Server Security Administrator's Guide* for more information about RACF PassTickets.

Sending encrypted passwords from a workstation

Depending on the DRDA level, clients can encrypt passwords; user IDs and passwords; or user IDs, passwords, and security-sensitive data when they send them to a DB2 UDB for z/OS server. This support uses the Diffie-Hellman key distribution algorithm.[5]

To enable DB2 Connect to flow encrypted passwords, database connection services (DCS) authentication must be set to DCS_ENCRYPT in the DCS directory entry. When the workstation application issues an SQL CONNECT, the workstation negotiates this support with the database server. If supported, a shared private key is generated by the client and server using the Diffie-Hellman public key technology and the password is encrypted using 56-bit DES with the shared private key. The encrypted password is non-replayable, and the shared private key is generated on every connection. If the server does not support password encryption, the application receives SQLCODE -30073 (DRDA security manager level 6 is not supported).

Establishing RACF protection for DB2

For purposes of illustration, suppose that the system of RACF IDs that is shown in Figure 24 is used to control DB2 usage.

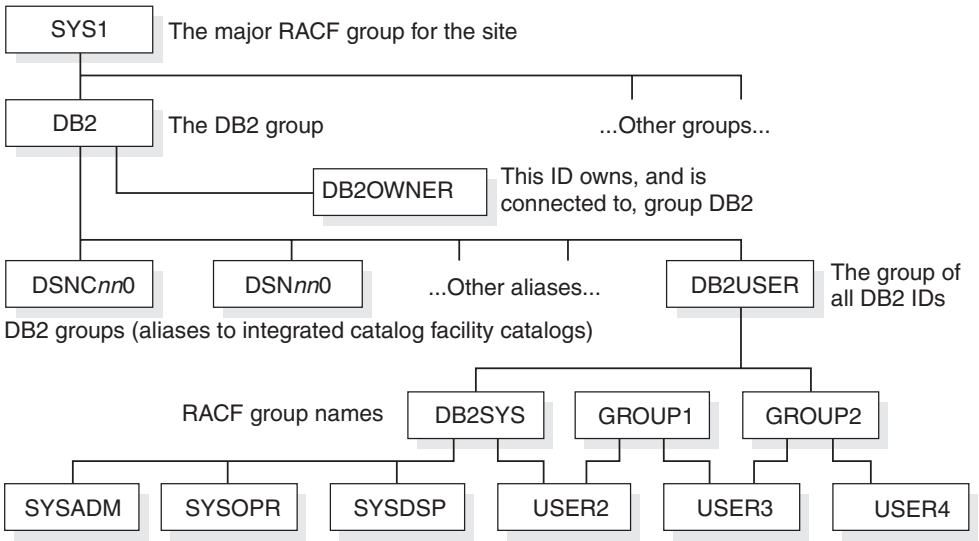


Figure 24. Sample DB2–RACF environment

Figure 24 shows some of the relationships among the names that are shown in Table 84.

Table 84. RACF relationships

RACF ID	Use
SYS1	Major RACF group ID
DB2	DB2 group
DB2OWNER	Owner of the DB2 group

5. Diffie-Hellman is one of the first standard public key algorithms. It results in exchanging a connection key which is used by client and server to generate a shared private key. The 56-bit Data Encryption Standards (DES) algorithm is used for encrypting and decrypting of the password using the shared private key.

Table 84. RACF relationships (continued)

RACF ID	Use
DSNC810	Group to control databases and recovery logs
DSN810	Group to control installation data sets
DB2USER	Group of all DB2 users
SYSADM	ID with DB2 installation SYSADM authority
SYSOPR	ID with DB2 installation SYSOPR authority
DB2SYS, GROUP1, GROUP2	RACF group names
SYSDSP	RACF user ID for DB2 started tasks
USER1, USER2, USER3	RACF user IDs.

Note: These RACF group names and user IDs do not appear in the figure; they are listed in Table 85 on page 268.

To establish RACF protection for DB2, perform the steps that are described in the following sections:

- "Defining DB2 resources to RACF" includes steps that tell RACF what to protect.
- "Permitting RACF access" on page 267 includes steps that make the protected resources available to processes.

Some steps are required and some steps are optional, depending on your circumstances. All steps presume that RACF is already installed. The steps do not need to be taken strictly in the order in which they are shown here.

For a more thorough description of RACF facilities, see *z/OS Security Server Security Administrator's Guide*.

For information about using RACF for multilevel security with row-level granularity, see "Multilevel security" on page 192.

Defining DB2 resources to RACF

To define DB2 resources to the RACF system, perform the following required steps in any order:

- "Define the names of protected access profiles" on page 266
- "Enable RACF checking for the DSNR and SERVER classes" on page 266

No one can access the DB2 subsystem until you instruct RACF to permit access.

Other tasks you might want to perform include:

- Controlling whether two DBMSs using VTAM LU 6.2 can establish sessions with each other, as described in "Enable partner-LU verification" on page 267.
- Ensure that IDs that are associated with stored procedures address spaces are authorized to run the appropriate attachment facility, as described in "Step 1: Control access by using the attachment facilities (required)" on page 274.
- If you are using TCP/IP, ensure that the ID that is associated with the DDF address space is authorized to use z/OS UNIX System Services, as described in "Establishing RACF protection for TCP/IP" on page 278.

Define the names of protected access profiles

The RACF resource class for DB2 is DSNR, and that class is contained in the RACF class descriptor table. Profiles in that class protect access to a DB2 subsystem from one of these environments—IMS, CICS, the distributed data facility (DDF), TSO, CAF, or batch.

Each profile has a name of the form *subsystem.environment*, where:

- *subsystem* is the name of a DB2 subsystem, of one to four characters; for example, DSN or DB2T.
- *environment* denotes the environment, by one of the following terms:
  - MASS for IMS (including MPP, BMP, Fast Path, and DL/I batch).
  - SASS for CICS.
  - DIST for DDF.
  - RRSAF for Resource Recovery Services attachment facility. Stored procedures use RRSAF in WLM-established address spaces.
  - BATCH for all others, including TSO, CAF, batch, all utility jobs, DB2-established stored procedures address space, and requests that come through the call attachment facility.

To control access, you need to define a profile, as a member of class DSNR, for every combination of subsystem and environment you want to use. For example, suppose that you want to access:

- Subsystem DSN from TSO and DDF
- Subsystem DB2P from TSO, DDF, IMS, and RRSAF
- Subsystem DB2T from TSO, DDF, CICS, and RRSAF

Then define the profiles with the following names:

```
DSN.BATCH   DSN.DIST
DB2P.BATCH  DB2P.DIST  DB2P.MASS  DB2P.RRSAF
DB2T.BATCH  DB2T.DIST  DB2T.SASS  DB2T.RRSAF
```

You can do that with a single RACF command, which also names an owner for the resources:

```
RDEFINE DSNR (DSN.BATCH  DSN.DIST  DB2P.BATCH  DB2P.DIST  DB2P.MASS DB2P.RRSAF
              DB2T.BATCH DB2T.DIST DB2T.SASS DB2T.RRSAF)  OWNER(DB2OWNER)
```

In order to access a subsystem in a particular environment, a user must be on the access list of the corresponding profile. You add users to the access list with the RACF PERMIT command. If you do not want to limit access to particular users or groups, you can give universal access to a profile with a command like this:

```
RDEFINE DSNR (DSN.BATCH) OWNER(DB2OWNER) UACC(READ)
```

Enable RACF checking for the DSNR and SERVER classes

To enable RACF access checking for resources in the DSNR resource class, issue this RACF command:

```
SETROPTS CLASSACT(DSNR)
```

Only users with the SPECIAL attribute can issue the command.

If you are using stored procedures in a WLM-established address space, you might also need to enable RACF checking for the SERVER class. See “Step 2: Control access to WLM (optional)” on page 275.

Enable partner-LU verification

With RACF and VTAM, you can control whether two LUs that use LU 6.2 can connect to each other.

Each member of a connecting pair must establish a profile for the other member. For example, if LUAAA and LUBBB are to connect and know each other by those LUNAMES, issue RACF commands similar to these:

At LUAAA: `RDEFINE APPCLU netid.LUAAA.LUBBB UACC(NONE) ...`
At LUBBB: `RDEFINE APPCLU netid.LUBBB.LUAAA UACC(NONE) ...`

Here, *netid* is the network ID, given by the VTAM start option NETID.

When you create those profiles with RACF, use the SESSION operand to supply:

- The VTAM password as a session key (SESSKEY suboperand)
- The maximum number of days between changes of the session key (INTERVAL suboperand)
- An indication of whether the LU pair is locked (LOCK suboperand)

For details, see *z/OS Security Server RACF Security Administrator's Guide*.

Finally, to enable RACF checking for the new APPCLU resources, issue this RACF command at both LUAAA and LUBBB:

```
SETROPTS CLASSACT(APPCLU)
```

Permitting RACF access

To let processes use the protected resources, take the steps described in the following sections:

1. "Define RACF user IDs for DB2 started tasks"
2. "Add RACF groups" on page 270
3. "Permit access for users and groups" on page 271

Define RACF user IDs for DB2 started tasks

A DB2 subsystem has the following started-task address spaces:

- *ssnm*DBM1 for database services
- *ssnm*MSTR for system services
- *ssnm*DIST for the distributed data facility
- *ssnm*SPAS for the DB2-established stored procedures address space
- Names for your WLM-established address spaces for stored procedures

You must associate each of these address spaces with a RACF user ID. Each of them can also be assigned a RACF group name. The DB2 address spaces **cannot** be started with batch jobs.

If you have IMS or CICS applications issuing DB2 SQL requests, you must associate RACF user IDs, and can associate group names, with:

- The IMS control region
- The CICS address space
- The four DB2 address spaces

If the IMS and CICS address spaces are started as batch jobs, provide their RACF IDs and group names with the USER and GROUP parameters on the JOB statement. If they are started as started tasks, assign the IDs and group names as you do for the DB2 address spaces, by changing the RACF STARTED class or the RACF started procedures table.

Stored procedures: Entries for stored procedures address spaces are required in the RACF started procedures table. The associated RACF user ID and group name

do not need to match those that are used for the DB2 address spaces, but they must be authorized to run the call attachment facility (for the DB2-established stored procedures address space) or Resource Recovery Services attachment facility (for WLM-established stored procedures address spaces). Note: WLM-established stored procedures started tasks IDs require an OMVS segment.

Changing the RACF started-procedures table: To change the RACF started-procedures table (ICHRIN03), change, reassemble, and link edit the resulting object code to z/OS. Figure 25 on page 269 shows the sample entries for three DB2 subsystems and optional entries for CICS and IMS. (Refer to *z/OS Security Server RACF System Programmer's Guide* for a description of how to code a RACF started-procedures table.) The example provides the DB2 started tasks for each of three DB2 subsystems, named DSN, DB2T, and DB2P, and for CICS and an IMS control region.

The IDs and group names associated with the address spaces are shown in Table 85.

Table 85. DB2 address space IDs and associated RACF user IDs and group names

Address Space	RACF User ID	RACF Group Name
DSNMSTR	SYSDSP	DB2SYS
DSNDBM1	SYSDSP	DB2SYS
DSNDIST	SYSDSP	DB2SYS
DSNSPAS	SYSDSP	DB2SYS
DSNWLM	SYSDSP	DB2SYS
DB2TMSTR	SYSDSPT	DB2TEST
DB2TDBM1	SYSDSPT	DB2TEST
DB2TDIST	SYSDSPT	DB2TEST
DB2TSPAS	SYSDSPT	DB2TEST
DB2PMSTR	SYSDSPD	DB2PROD
DB2PDBM1	SYSDSPD	DB2PROD
DB2PDIST	SYSDSPD	DB2PROD
DB2PSPAS	SYSDSPD	DB2PROD
CICSSYS	CICS	CICSGRP
IMSCNTL	IMS	IMSGRP

Figure 25 on page 269 shows a sample job that reassembles and link edits the RACF started-procedures table (ICHRIN03):

```
//*
//*  REASSEMBLE AND LINKEDIT THE RACF STARTED-PROCEDURES
//*  TABLE ICHRIN03 TO INCLUDE USERIDS AND GROUP NAMES
//*  FOR EACH DB2 CATALOGED PROCEDURE. OPTIONALLY, ENTRIES
//*  FOR AN IMS OR CICS SYSTEM MIGHT BE INCLUDED.
//*
//*  AN IPL WITH A CLPA (OR AN MLPA SPECIFYING THE LOAD
//*  MODULE) IS REQUIRED FOR THESE CHANGES TO TAKE EFFECT.
//*

ENTCOUNT DC    AL2(((ENDTABLE-BEGTABLE)/ENTLNGTH)+32768)
*              NUMBER OF ENTRIES AND INDICATE RACF FORMAT
*
*  PROVIDE FOUR ENTRIES FOR EACH DB2 SUBSYSTEM NAME.
*
BEGTABLE DS    0H
*        ENTRIES FOR SUBSYSTEM NAME "DSN"
         DC    CL8'DSNMSTR'        SYSTEM SERVICES PROCEDURE
         DC    CL8'SYSDSP'         USERID
         DC    CL8'DB2SYS'         GROUP NAME
         DC    X'00'               NO PRIVILEGED ATTRIBUTE
         DC    XL7'00'             RESERVED BYTES
ENTLNGTH EQU   *-BEGTABLE     CALCULATE LENGTH OF EACH ENTRY
         DC    CL8'DSNDBM1'   DATABASE SERVICES PROCEDURE
         DC    CL8'SYSDSP'         USERID
         DC    CL8'DB2SYS'         GROUP NAME
         DC    X'00'               NO PRIVILEGED ATTRIBUTE
         DC    XL7'00'             RESERVED BYTES
         DC    CL8'DSNDIST'        DDF PROCEDURE
         DC    CL8'SYSDSP'         USERID
         DC    CL8'DB2SYS'         GROUP NAME
         DC    X'00'               NO PRIVILEGED ATTRIBUTE
         DC    XL7'00'             RESERVED BYTES
         DC    CL8'DSNSPAS'        STORED PROCEDURES PROCEDURE
         DC    CL8'SYSDSP'         USERID
         DC    CL8'DB2SYS'         GROUP NAME
         DC    X'00'               NO PRIVILEGED ATTRIBUTE
         DC    XL7'00'             RESERVED BYTES
         DC    CL8'DSNWLM'         WLM-ESTABLISHED S.P. ADDRESS SPACE
         DC    CL8'SYSDSP'         USERID
         DC    CL8'DB2SYS'         GROUP NAME
         DC    X'00'               NO PRIVILEGED ATTRIBUTE
         DC    XL7'00'             RESERVED BYTES
*        ENTRIES FOR SUBSYSTEM NAME "DB2T"
         DC    CL8'DB2TMSTR'       SYSTEM SERVICES PROCEDURE
         DC    CL8'SYSDSPT'        USERID
         DC    CL8'DB2TEST'        GROUP NAME
         DC    X'00'               NO PRIVILEGED ATTRIBUTE
         DC    XL7'00'             RESERVED BYTES
         DC    CL8'DB2TDBM1'       DATABASE SERVICES PROCEDURE
         DC    CL8'SYSDSPT'        USERID
         DC    CL8'DB2TEST'        GROUP NAME
         DC    X'00'               NO PRIVILEGED ATTRIBUTE
         DC    XL7'00'             RESERVED BYTES
         DC    CL8'DB2TDIST'       DDF PROCEDURE
         DC    CL8'SYSDSPT'        USERID
         DC    CL8'DB2TEST'        GROUP NAME
         DC    X'00'               NO PRIVILEGED ATTRIBUTE
         DC    XL7'00'             RESERVED BYTES
         DC    CL8'DB2TSPAS'       STORED PROCEDURES PROCEDURE
         DC    CL8'SYSDSPT'        USERID
         DC    CL8'DB2TEST'        GROUP NAME
         DC    X'00'               NO PRIVILEGED ATTRIBUTE
         DC    XL7'00'             RESERVED BYTES
```

Figure 25. Sample job to reassemble the RACF started-procedures table (Part 1 of 2)

```
*        ENTRIES FOR SUBSYSTEM NAME "DB2P"
         DC    CL8'DB2PMSTR'       SYSTEM SERVICES PROCEDURE
         DC    CL8'SYSDSPD'        USERID
         DC    CL8'DB2PROD'        GROUP NAME
         DC    X'00'               NO PRIVILEGED ATTRIBUTE
         DC    XL7'00'             RESERVED BYTES
         DC    CL8'DB2PDBM1'       DATABASE SERVICES PROCEDURE
         DC    CL8'SYSDSPD'        USERID
         DC    CL8'DB2PROD'        GROUP NAME
         DC    X'00'               NO PRIVILEGED ATTRIBUTE
         DC    XL7'00'             RESERVED BYTES
         DC    CL8'DB2PDIST'       DDF PROCEDURE
         DC    CL8'SYSDSPD'        USERID
         DC    CL8'DB2PROD'        GROUP NAME
         DC    X'00'               NO PRIVILEGED ATTRIBUTE
         DC    XL7'00'             RESERVED BYTES
         DC    CL8'DB2PSPAS'       STORED PROCEDURES PROCEDURE
         DC    CL8'SYSDSPD'        USERID
         DC    CL8'DB2PROD'        GROUP NAME
         DC    X'00'               NO PRIVILEGED ATTRIBUTE
         DC    XL7'00'             RESERVED BYTES
*        OPTIONAL ENTRIES FOR CICS AND IMS CONTROL REGION
         DC    CL8'CICSSYS'        CICS PROCEDURE NAME
         DC    CL8'CICS'           USERID
         DC    CL8'CICSGRP'        GROUP NAME
         DC    X'00'               NO PRIVILEGED ATTRIBUTE
         DC    XL7'00'             RESERVED BYTES
         DC    CL8'IMSCNTL'   IMS CONTROL REGION PROCEDURE
         DC    CL8'IMS'            USERID
         DC    CL8'IMSGRP'         GROUP NAME
         DC    X'00'               NO PRIVILEGED ATTRIBUTE
         DC    XL7'00'             RESERVED BYTES
ENDTABLE DS    0D
         END
```

Figure 25. Sample job to reassemble the RACF started-procedures table (Part 2 of 2)

Add RACF groups

The details of this step depend on the groups that you define. To add the user DB2OWNER, issue the following RACF command:

```
ADDUSER  DB2OWNER  CLAUTH(DSNR USER) UACC(NONE)
```

That gives class authorization to DB2OWNER for DSNR and USER. DB2OWNER can add users to RACF and issue the RDEFINE command to define resources in class DSNR. DB2OWNER has control over and responsibility for the entire DB2 security plan in RACF.

The RACF group SYS1 already exists. To add group DB2 and make DB2OWNER its owner, issue the following RACF command:

```
ADDGROUP DB2 SUPGROUP(SYS1) OWNER(DB2OWNER)
```

To connect DB2OWNER to group DB2 with the authority to create new subgroups, add users, and manipulate profiles, issue the following RACF command:

```
CONNECT DB2OWNER GROUP(DB2) AUTHORITY(JOIN) UACC(NONE)
```

To make DB2 the default group for commands issued by DB2OWNER, issue the following RACF command:

```
ALTUSER DB2OWNER DFLTGRP(DB2)
```

To create the group DB2USER and add five users to it, issue the following RACF commands:

```
ADDGROUP DB2USER SUPGROUP(DB2)
ADDUSER (USER1 USER2 USER3 USER4 USER5) DFLTGRP(DB2USER)
```

To define a user to RACF, use the RACF ADDUSER command. That invalidates the current password. You can then log on as a TSO user to change the password.

DB2 considerations when using RACF groups:

- When a user is newly connected to, or disconnected from, a RACF group, the change is not effective until the next logon. Therefore, before using a new group name as a secondary authorization ID, a TSO user must log off and log on, or a CICS or IMS user must sign on again.
- A user with the SPECIAL, JOIN, or GROUP-SPECIAL RACF attribute can define new groups with any name that RACF accepts and can connect any user to them. Because the group name can become a secondary authorization ID, you should control the use of those RACF attributes.
- Existing RACF group names can duplicate existing DB2 authorization IDs. That duplication is unlikely for the following reasons:
  - A group name cannot be the same as a user name.
  - Authorization IDs that are known to DB2 are usually known to RACF.

  However, you can create a table with an owner name that is the same as a RACF group name and use the IBM-supplied sample connection exit routine. Then any TSO user with the group name as a secondary ID has ownership privileges on the table. You can prevent that situation by designing the connection exit routine to stop unwanted group names from being passed to DB2.

Permit access for users and groups

In this scenario, DB2OWNER is authorized for class DSNR, owns the profiles, and has the right to change them. The following commands let users that are members of the group DB2USER, the system administrators, and operators to be TSO users. These users can run batch jobs and DB2 utilities on the three systems: DSN, DB2P, and DB2T. The ACCESS(READ) operand allows use of DB2 without the ability to manipulate profiles.

```
PERMIT  DSN.BATCH  CLASS(DSNR) ID(DB2USER)  ACCESS(READ)
PERMIT  DB2P.BATCH CLASS(DSNR) ID(DB2USER)  ACCESS(READ)
PERMIT  DB2T.BATCH CLASS(DSNR) ID(DB2USER)  ACCESS(READ)
```

Defining profiles for IMS and CICS: You want the IDs for attaching systems to use the appropriate access profile. For example, to let the IMS user ID use the access profile for IMS on system DB2P, issue the following RACF command:

```
PERMIT  DB2P.MASS  CLASS(DSNR) ID(IMS)  ACCESS(READ)
```

To let the CICS group ID use the access profile for CICS on system DB2T, issue the following RACF command:

```
PERMIT  DB2T.SASS   CLASS(DSNR) ID(CICSGRP)  ACCESS(READ)
```

Providing installation authorities to default IDs: When DB2 is installed, IDs are named to have special authorities—one or two IDs for SYSADM and one or two IDs for SYSOPR. Those IDs can be connected to the group DB2USER; if they are not, you need to give them access. The next command permits the default IDs for the SYSADM and SYSOPR authorities to use subsystem DSN through TSO:

```
PERMIT  DSN.BATCH  CLASS(DSNR) ID(SYSADM,SYSOPR) ACCESS(READ)
```

IDs also can be group names.

Using secondary IDs: You can use secondary authorization IDs to define a RACF group. After you define the RACF group, you can assign privileges to it that are shared by multiple primary IDs. For example, suppose that DB2OWNER wants to create a group GROUP1 and to give the ID USER1 administrative authority over the group. USER1 should be able to connect other existing users to the group. To create the group, DB2OWNER issues this RACF command:

```
ADDGROUP  GROUP1  OWNER(USER1) DATA('GROUP FOR DEPT. G1')
```

To let the group connect to the DSN system through TSO, DB2OWNER issues this RACF command:

```
PERMIT DSN.BATCH CLASS(DSNR) ID(GROUP1) ACCESS(READ)
```

USER1 can now connect other existing IDs to the group GROUP1 by using the RACF CONNECT command:

```
CONNECT (USER2 EPSILON1 EPSILON2) GROUP(GROUP1)
```

If you add or update secondary IDs for CICS transactions, you must start and stop the CICS attachment facility to ensure that all threads sign on and get the correct security information.

Allowing users to create data sets: Chapter 13, "Auditing," on page 285 recommends using RACF to protect the data sets that store DB2 data. If you use that method, when you create a new group of DB2 users, you might want to connect it to a group that can create data sets. To allow USER1 to create and control data sets, DB2OWNER creates a generic profile and permits complete control to USER1 and to the four administrators. The SYSDSP parameter also gives control to DB2. See See "Creating generic profiles for data sets" on page 281.

```
ADDSD  'DSNC810.DSNDBC.ST*' UACC(NONE)

PERMIT 'DSNC810.DSNDBC.ST*'
       ID(USER1 SYSDSP SYSAD1 SYSAD2 SYSOP1 SYSOP2) ACCESS(ALTER)
```

Allowing access from remote requesters: The recommended way of controlling access from remote requesters is to use the DSNR RACF class with a PERMIT command to access the distributed data address space (such as DSN.DIST).

The following RACF commands let the users in the group DB2USER access DDF on the DSN subsystem. These DDF requests can originate from any partner in the network.

Example: To permit READ access on profile DSN.DIST in the DSNR class to DB2USER, issue the following RACF command:

```
PERMIT  DSN.DIST   CLASS(DSNR) ID(DB2USER)  ACCESS(READ)
```

If you want to ensure that a specific user can access only when the request originates from a specific LU name, you can use WHEN(APPCPORT) on the PERMIT command.

Example: To permit access to DB2 distributed processing on subsystem DSN when the request comes from USER5 at LUNAME equal to NEWYORK, issue the following RACF command:

```
PERMIT DSN.DIST CLASS(DSNR) ID(USER5) ACCESS(READ) +
       WHEN(APPCPORT(NEWYORK))
```

For connections that come through TCP/IP, use the RACF APPCPORT class, the RACF SERVAUTH class, or both classes, with TCP/IP Network Access Control to protect unauthorized acces to DB2.

Example: To use the RACF APPCPORT class, perform the following steps:

1. Activate the ACCPORT class by issuing the following RACF command:
   ```
   SETROPTS CLASSACT(APPCPORT) REFRESH
   ```
2. Define the general resource profile and name it TCPIP. Specify NONE for universal access and APPCPORT for class. Issue the following RACF command:
   ```
   RDEFINE APPCPORT (TCPIP) UACC(NONE)
   ```
3. Permit READ access on profile TCPIP in the APPCPORT class. To permit READ access to USER5, issue the following RACF command:
   ```
   PERMIT TCPIP ACCESS(READ) CLASS(APPPORT) ID(USER5)
   ```
4. Permit READ access on profile DSN.DIST in the DSNR class. To permit READ access to USER5, issue the following RACF command:
   ```
   PERMIT DSN.DIST CLASS(DSNR) ID(USER5) ACCESS(READ) +
          WHEN(APPCPORT(TCPIP))
   ```
5. Refresh the APPCPORT class by issuing the following RACF command:
   ```
   SETROPTS CLASSACT(APPCPORT) REFRESH RACLIST(APPCPORT)
   ```

If the RACF APPCPORT class is active on your system, and a resource profile for the requesting LU name already exists, you must permit READ access to the APPCPORT resource profile for the user IDs that DB2 uses. You must permit READ access even when you are using the DSNR resource class. Similarly, if you are using the RACF APPL class and that class restricts access to the local DB2 LU name or generic LU name, you must permit READ access to the APPL resource for the user IDs that DB2 uses.

Requirement: To use the RACF SERVAUTH class and TCP/IP Network Access Control, you must have z/OS V1.5 (or later) installed.

Example: To use the RACF SERVAUTH class and TCP/IP Network Access Control, perform the following steps:

1. Set up and configure TCP/IP Network Access Control by using the NETACCESS statement that is in your TCP/IP profile.

   For example, suppose that you need to allow z/OS system access only to IP addresses from 9.0.0.0 to 9.255.255.255. You want to define these IP addresses as a security zone, and you want to name the security zone IBM. Suppose also that you need to deny access to all IP addressed outside of the IBM security zone, and that you want to define these IP addresses as a separate security zone. You want to name this second security zone WORLD. To establish these security zones, use the following NETACCESS clause:
   ```
   NETACCESS INBOUND OUTBOUND
   ; NETWORK/MASK       SAF
     9.0.0.0/8          IBM
     DEFAULT            WORLD
   ENDNETACCESS
   ```
   Now, suppose that USER5 has an IP address of 9.1.2.3. TCP/IP Network Access Control would determine that USER5 has an IP address that belongs to the IBM security zone. USER5 would be granted access to the system. Alternatively, suppose that USER6 has an IP address of 1.1.1.1. TCP/IP Network Access Control would determine that USER6 has an IP address that belongs to the WORLD security zone. USER6 would not be granted access to the system.
2. Activate the SERVAUTH class by issuing the following TSO command:

```
SETROPTS CLASSACT(SERVAUTH)
```

3. Activate RACLIST processing for the SERVAUTH class by issuing the following TSO command:

```
SETROPTS RACLIST(SERVAUTH)
```

4. Define the IBM and WORLD general resource profiles in RACF to protect the IBM and WORLD security zones by issuing the following commands:

```
RDEFINE SERVAUTH (EZB.NETACCESS.ZOSV1R5.TCPIP.IBM) UACC(NONE)
RDEFINE SERVAUTH (EZB.NETACCESS.ZOSV1R5.TCPIP.WORLD) UACC(NONE)
```

5. Permit USER5 and SYSDSP read access to the IBM profile by using the following commands.

```
PERMIT EZB.NETACCESS.ZOSV1R5.TCPIP.IBM ACCESS READ CLASS(SERVAUTH) ID(USER5)
PERMIT EZB.NETACCESS.ZOSV1R5.TCPIP.IBM ACCESS READ CLASS(SERVAUTH) ID(SYSDSP)
```

6. Permit SYSDSP read access to the WORLD profile by using the following command:

```
PERMIT EZB.NETACCESS.ZOSV1R5.TCPIP.WORLD ACCESS READ CLASS(SERVAUTH) ID(USER5)
```

7. For these permissions to take effect, refresh the RACF database by using the following command:

```
SETROPTS CLASSACT(SERVAUTH) REFRESH RACLIST(SERVAUTH)
```

For more information about the NETACCESS statement, see *z/OS V1.5 Communications Server: IP Configuration Reference*.

Issuing DB2 commands

IDs must be authorized to issue DB2 commands. If you authorize IDs by issuing DB2 GRANT statements, the GRANTs must be made to a primary authorization ID, to a secondary authorization ID, or to PUBLIC.

When RACF is used for access control, an ID must have appropriate RACF authorization on DB2 commands or must be granted authorization for DB2 commands to issue commands from a logged-on MVS console or from TSO SDSF.

You can ensure that an ID can issue DB2 commands from logged-on MVS consoles or TSO SDSF by using one of the following methods:

- Grant authorization for DB2 commands to the primary or secondary authorization ID.
- Define RACF classes and permits for DB2 commands.
- Grant SYSOPR authority to appropriate IDs.

Establishing RACF protection for stored procedures

This section summarizes the procedures that you can follow for establishing RACF protection for stored procedures that run on your DB2 subsystem.

This section contains the following procedures:

- "Step 1: Control access by using the attachment facilities (required)"
- "Step 2: Control access to WLM (optional)" on page 275
- "Step 3: Control access to non-DB2 resources (optional)" on page 277

Step 1: Control access by using the attachment facilities (required)

The user ID that is associated with the DB2-established address space must be authorized to run the DB2 call attachment facility. It must be associated with the *ssnm*.BATCH profile, as described in "Define the names of protected access profiles" on page 266.

The user ID that is associated with the WLM-established stored procedures address space must be authorized to run Resource Recovery Services attachment facility (RRSAF). The user id is also associated with the *ssnm*.RRSAF profile that you create.

Control access to the DB2 subsystem through RRSAF by performing the following steps:

1. If you have not already established a profile for controlling access from the RRS attachment facility as described in "Define the names of protected access profiles" on page 266, define *ssnm*.RRSAF in the DSNR resource class with a universal access authority of NONE, as shown in the following command:

   ```
   RDEFINE DSNR (DB2P.RRSAF  DB2T.RRSAF) UACC(NONE)
   ```

2. Activate the resource class; use the following command:

   ```
   SETROPTS RACLIST(DSNR) REFRESH
   ```

3. Add user IDs that are associated with the stored procedures address spaces to the RACF Started Procedures Table, as shown in this example:

   ```
   .
   .
   .
   DC    CL8'DSNWLM'         WLM-ESTABLISHED S.P. ADDRESS SPACE
   DC    CL8'SYSDSP'         USERID
   DC    CL8'DB2SYS'         GROUP NAME
   DC    X'00'               NO PRIVILEGED ATTRIBUTE
   DC    XL7'00'             RESERVED BYTES
   .
   .
   .
   ```

4. Give read access to *ssnm*.RRSAF to the user ID that is associated with the stored procedures address space:

   ```
   PERMIT  DB2P.RRSAF  CLASS(DSNR) ID(SYSDSP)  ACCESS(READ)
   ```

Step 2: Control access to WLM (optional)

You can control which address spaces can be WLM-established server address spaces that run stored procedures. To do this, use the server resource class, which WLM uses to identify valid address spaces to which work can be sent. If the server class is not defined or activated, any address space is allowed to connect to WLM as a server address space and to identify itself as a server address space that runs stored procedures.

To use the server resource class, perform the following steps:

1. Run a version of RACF in which the resource class SERVER is included as part of the predefined resource classes (RACF Version 2 Release 2 and subsequent releases).
2. Define a RACF profile for the server resource class named SERVER, as follows:

   ```
   RDEFINE SERVER (DB2.ssnm.applenv)
   ```

   applenv is the name of the application environment that is associated with the stored procedure. See "Assigning procedures and functions to WLM application environments" on page 1020 for more information about application environments.

   Assume that you want to define the following profile names:

   - DB2.DB2T.TESTPROC
   - DB2.DB2P.PAYROLL
   - DB2.DB2P.QUERY

   Use the following RACF command:

```
RDEFINE SERVER (DB2.DB2T.TESTPROC DB2.DB2P.PAYROLL DB2.DB2P.QUERY)
```

3. Activate the resource class SERVER as follows:

```
SETROPTS RACLIST(SERVER) REFRESH
```

4. Permit read access to the server resource name to the user IDs that are associated with the stored procedures address space as follows.

```
PERMIT  DB2.DB2T.TESTPROC CLASS(SERVER) ID(SYSDSP) ACCESS(READ)
PERMIT  DB2.DB2P.PAYROLL  CLASS(SERVER) ID(SYSDSP) ACCESS(READ)
PERMIT  DB2.DB2P.QUERY    CLASS(SERVER) ID(SYSDSP) ACCESS(READ)
```

Control of stored procedures in a WLM environment: Programs can be grouped together and isolated in different WLM environments based on application security requirements. For example, payroll applications might be isolated in one WLM environment because they contain sensitive data, such as employee salaries.

To prevent users from creating a stored procedure in a sensitive WLM environment, DB2 invokes RACF to determine if the user is allowed to create stored procedures in the specified WLM environment. The WLM ENVIRONMENT keyword on the CREATE PROCEDURE statement identifies the WLM environment to use for running a given stored procedure. Attempts to create a stored procedure fail if the user is not properly authorized.

DB2 performs a resource authorization check using the DSNR RACF class as follows:

- In a DB2 data sharing environment, DB2 uses the following RACF resource name:

  ```
  db2_groupname.WLMENV.wlm_environment
  ```

- In a non-data sharing environment, DB2 checks the following RACF resource name:

  ```
  db2_subsytem_id.WLMENV.wlm_environment
  ```

You can use the RACF RDEFINE command to create RACF profiles that prevent users from creating stored procedures and user-defined functions in sensitive WLM environments. For example, you can prevent all users on DB2 subsystem DB2A (non-data sharing) from creating a stored procedure or user-defined function in the WLM environment named PAYROLL; to do this, use the following command:

```
RDEFINE DSNR (DB2A.WLMENV.PAYROLL) UACC(NONE)
```

The RACF PERMIT command authorizes a user to create a stored procedure or user-defined function in a WLM environment. For example, you can authorize a DB2 user (DB2USER1) to create stored procedures on DB2 subsystem DB2A (non-data sharing) in the WLM environment named PAYROLL:

```
PERMIT  DB2A.WLMENV.PAYROLL  CLASS(DSNR) ID(DB2USER1)  ACCESS(READ)
```

Control of stored procedures in a DB2-established stored procedures address space: DB2 invokes RACF to determine if a user is allowed to create a stored procedures in a DB2-established stored procedure address space. The NO WLM ENVIRONMENT keyword on the CREATE PROCEDURE statement indicates that a given stored procedure will run in a DB2-established stored procedures address space. Attempts to create a procedure fail if the user is not authorized, or if there is no DB2-established stored procedures address space exists.

The RACF PERMIT command authorizes a user to create a stored procedure in a DB2-established stored procedures address space. For example, you can authorize a DB2 user (DB2USER1) to create stored procedures on DB2 subsystem DB2A in the stored procedures address space named DB2ASPAS:

```
PERMIT  DB2A.WLMENV.DB2ASPAS  CLASS(DSNR) ID(DB2USER1)  ACCESS(READ)
```

Step 3: Control access to non-DB2 resources (optional)

With stored procedures that run in a DB2-established address space, the user ID of the stored procedures address space (from the started procedures table of RACF) is used to access non-DB2 resources such as IMS or CICS transactions, APPC conversations, or VSAM files.

With WLM-established address spaces, you can specify that access to non-DB2 resources is controlled by the authorization ID of the caller rather than that of the stored procedure address space. To do this, specify U in the EXTERNAL_SECURITY column of table SYSIBM.SYSROUTINES for the stored procedure.

When you specify U for EXTERNAL_SECURITY, a separate RACF environment is established for that stored procedure. Use EXTERNAL_SECURITY=U only when the caller must access resources outside of DB2. Figure 26 shows the user ID that is associated with each part of a stored procedure.

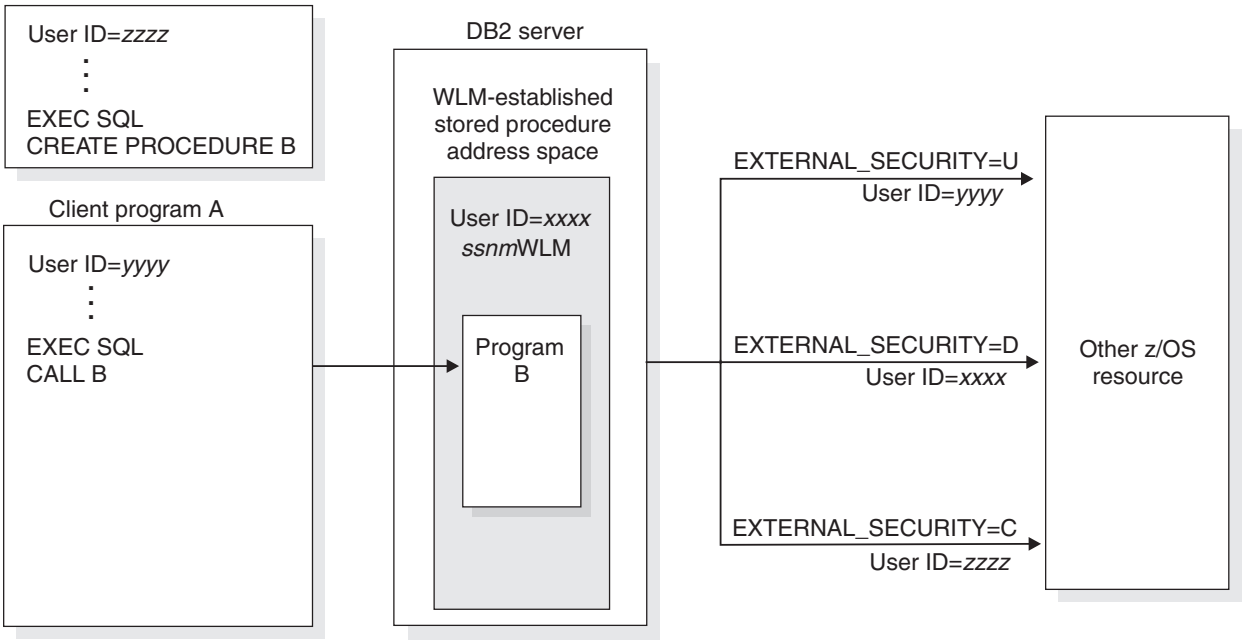


Figure 26. Accessing non-DB2 resources from a stored procedure

For WLM-established stored procedures address spaces, enable the RACF check for the caller's ID when you want to access non-DB2 resources by performing the following steps:

1. Use the ALTER PROCEDURE statement with the EXTERNAL_SECURITY USER clause.
2. Ensure that the ID of the stored procedure's caller has RACF authority to the resources.
3. For the best performance, cache the RACF profiles in the virtual look-aside facility (VLF) of z/OS. Do this by specifying the following keywords in the COFVLF*xx* member of library SYS1.PARMLIB.

   ```
   CLASS NAME(IRRACEE)
   EMAJ(ACEE)
   ```

Establishing RACF protection for TCP/IP

The ID that is associated with the DB2 distributed address space must be authorized to use UNIX system services if your DB2 subsystem is going to send or accept any requests over TCP/IP. To do this, you must create an OMVS segment in the RACF user profile, as follows:

```
ALTUSER (SYSDSP) OMVS(UID(0))
```

To give root authority to the DDF address space, you must specify a UID of 0.

Establishing Kerberos authentication through RACF

Kerberos security is a network security technology that was developed at the Massachusetts Institute of Technology. DB2 can use Kerberos security services to authenticate remote users. With Kerberos security services, remote end users access DB2 when they issue their Kerberos name and password. This same name and password is used for access throughout the network, so a separate password to access DB2 is not necessary.

The Kerberos security technology does not require passwords to flow in readable text, so it provides security even for client/server environments. This flexibility is possible because Kerberos uses an authentication technology that uses encrypted tickets that contain authentication information for the end user.

For information about Kerberos security, z/OS, and DB2 see the following books:

- *z/OS SecureWay Security Server Network Authentication Server Administration*
- *z/OS SecureWay Security Server Network Authentication Server Programming*
- *z/OS Security Server Security Administrator's Guide*
- *z/OS Security Server Command Language Reference*

For information about Kerberos security, OS/390, and DB2, see the following books:

- *OS/3090 SecureWay Security Server Network Authentication and Privacy Administration Guide*
- *OS/390 SecureWay Security Server (RACF) Security Administrator's Guide*
- *OS/390 Security Server (RACF) Command Language Reference*

Each remote user who is authenticated to DB2 by means of Kerberos authentication must be registered in RACF profiles. Each organization wishing to run a Kerberos server establishes its own *realm*. The name of the realm in which a client is registered is part of the client's name and can be used by the application server to decide whether to honor a request. Follow these steps to authenticate and register a remote user:

1. Define the Kerberos realm to RACF. The name of the local realm must be supplied in the definition. You must also supply a Kerberos password for RACF to grant Kerberos ticket-granting tickets. Define a Kerberos realm with the following command:

   ```
   RDEFINE REALM KERBDFLT KERB(KERBNAME(localrealm) PASSWORD(mykerpw)
   ```

2. Define local principals to RACF. Change RACF passwords before the principals become active Kerberos users. Define a Kerberos principal with the following commands:

   ```
   AU RONTOMS KERB(KERBNAME(rontoms))
   ALU RONTOMS PASSWORD(new1pw) NOEXPIRE
   ```

3. Map foreign Kerberos principals by defining KERBLINK profiles to RACF with a command similar to the following command:

   ```
   RDEFINE KERBLINK /.../KERB390.ENDICOTT.IBM.COM/RWH APPLDATA('RONTOMS')
   ```

   You must also define a principal name for the user ID used in the *ssnmDIST* started task address space. This step is required because the ssnmDIST address space must have the RACF authority to use its SAF ticket parsing service.

   ```
   ALU SYSDSP PASSWORD(pw) NOEXPIRE KERB(KERBNAME(SYSDSP))
   ```

   In this example, the user ID that is used for the *ssnmDIST* started task address space is SYSDSP. See "Define RACF user IDs for DB2 started tasks" on page 267 for more information, including how to determine the user ID for the ssnmDIST started task.
4. Define foreign Kerberos authentication servers to the local Kerberos authentication server by using REALM profiles. You must supply a password for the key to be generated. REALM profiles define the trust relationship between the local realm and the foreign Kerberos authentication servers. PASSWORD is a required keyword, so all REALM profiles have a KERB segment. The command is similar to the following command:

   ```
   RDEFINE REALM /.../KERB390.ENDICOTT.IBM.COM/KRBTGT/KER2000.ENDICOTT.IBM.COM +
   KERB(PASSWORD(realm0pw))
   ```

The z/OS SecureWay Kerberos Security Server rejects ticket requests from users with revoked or expired passwords; therefore, plan password resets that use a method that avoids a password change at a subsequent logon. For example, use the TSO logon panel the PASSWORD command without a specified ID operand, or the ALTUSER command with NOEXPIRE specified.

Data sharing environment: Data sharing Sysplex environments that use Kerberos security must have a Kerberos Security Server instance running on each system in the Sysplex. The instances must either be in the same realm and share the same RACF database, or have different RACF databases and be in different realms.

Other methods of controlling access

You can also help control access to DB2 from within IMS or CICS.

- IMS security

  IMS terminal security lets you limit the entry of a transaction code to a particular logical terminal (LTERM) or group of LTERMs in the system. To protect a particular program, you can authorize a transaction code that is to be entered only from any terminal on a list of LTERMs. Alternatively, you can associate each LTERM with a list of the transaction codes that a user can enter from that LTERM. IMS then passes the validated LTERM name to DB2 as the initial primary authorization ID.
- CICS security

  CICS transaction code security works with RACF to control the transactions and programs that can access DB2. Within DB2, you can use the ENABLE and DISABLE options of the bind operation to limit access to specific CICS subsystems.

Chapter 12. Protecting data sets through RACF

To fully protect the data in DB2, you must take steps to ensure that no other process has access to the data sets in which DB2 data resides.

Use RACF, or a similar external security system, to control access to the data sets just as RACF controls access to the DB2 subsystem. This section explains how to create RACF profiles for data sets and allow their use through DB2.

Assume that the RACF groups DB2 and DB2USER, and the RACF user ID DB2OWNER, have been set up for DB2 IDs, as described under "Defining DB2 resources to RACF" on page 265 (and shown in Figure 24 on page 264). Given that setting, the examples that follow show you how to:

- Add RACF groups to control data sets that use the default DB2 qualifiers
- Create generic profiles for different types of DB2 data sets and permit their use by DB2 started tasks
- Permit use of the profiles by specific IDs
- Allow certain IDs to create data sets.

The following topics provide additional information:

- "Adding groups to control DB2 data sets"
- "Creating generic profiles for data sets"
- "Permitting DB2 authorization IDs to use the profiles" on page 283
- "Allowing DB2 authorization IDs to create data sets" on page 283

Adding groups to control DB2 data sets

The default high-level qualifier for data sets that contain DB2 databases and recovery logs is DSNC810; for distribution, target, SMP, and other installation data sets, the default high-level qualifier is DSN810. The DB2OWNER user ID can create groups that control those data sets by issuing the following commands:

```
ADDGROUP DSNC810 SUPGROUP(DB2) OWNER(DB2OWNER)
ADDGROUP DSN810 SUPGROUP(DB2) OWNER(DB2OWNER)
```

Creating generic profiles for data sets

DB2 uses specific names to identify data sets for special purposes. In "Define RACF user IDs for DB2 started tasks" on page 267, SYSDSP is the RACF user ID for DB2 started tasks. DB2OWNER can issue the following commands to create generic profiles for the data sets and give complete control over the data sets to DB2 started tasks:

- For active logs, issue the following commands:

```
ADDSD  'DSNC810.LOGCOPY*' UACC(NONE)
PERMIT 'DSNC810.LOGCOPY*' ID(SYSDSP) ACCESS(ALTER)
```

- For archive logs, issue the following commands:

```
ADDSD  'DSNC810.ARCHLOG*' UACC(NONE)
PERMIT 'DSNC810.ARCHLOG*' ID(SYSDSP) ACCESS(ALTER)
```

- For bootstrap data sets, issue the following commands:

```
ADDSD  'DSNC810.BSDS*'    UACC(NONE)
PERMIT 'DSNC810.BSDS*'    ID(SYSDSP) ACCESS(ALTER)
```

© Copyright IBM Corp. 1982, 2008

- For table spaces and index spaces, issue the following commands:

```
ADDSD  'DSNC810.DSNDBC.*' UACC(NONE)
PERMIT 'DSNC810.DSNDBC.*' ID(SYSDSP) ACCESS(ALTER)
```

- For installation libraries, issue the following command:

```
ADDSD 'DSN810.*'          UACC(READ)
```

  Started tasks do not need control.

- For other general data sets, issue the following commands:

```
ADDSD  'DSNC810.*'        UACC(NONE)
PERMIT 'DSNC810.*'        ID(SYSDSP) ACCESS(ALTER)
```

Although all of those commands are not absolutely necessary, the sample shows how you can create generic profiles for different types of data sets. Some parameters, such as universal access, could vary among the types. In the example, installation data sets (DSN810.*) are universally available for read access.

If you use generic profiles, specify NO on installation panel DSNTIPP for ARCHIVE LOG RACF, or you might get a z/OS error when DB2 tries to create the archive log data set. If you specify YES, DB2 asks RACF to create a separate profile for each archive log that is created, which means that you cannot use generic profiles for these data sets.

To protect VSAM data sets, use the cluster name. You do not need to protect the data component names, because the cluster name is used for RACF checking.

Access by stand-alone DB2 utilities: The following DB2 utilities access objects that are outside of DB2 control:

- DSN1COPY and DSN1PRNT: table space and index space data sets
- DSN1LOGP: active logs, archive logs, and bootstrap data sets
- DSN1CHKR: DB2 directory and catalog table spaces
- Change Log Inventory (DSNJU003) and Print Log Map (DSNJU004): bootstrap data sets

The Change Log Inventory and Print Log Map utilities run as batch jobs that are protected by the USER and PASSWORD options on the JOB statement. To provide a value for the USER option, for example SVCAID, issue the following commands:

- For DSN1COPY:

```
PERMIT 'DSNC810.*' ID(SVCAID) ACCESS(CONTROL)
```

- For DSN1PRNT:

```
PERMIT 'DSNC810.*' ID(SVCAID) ACCESS(READ)
```

- For DSN1LOGP:

```
PERMIT 'DSNC810.LOGCOPY*' ID(SVCAID) ACCESS(READ)
PERMIT 'DSNC810.ARCHLOG*' ID(SVCAID) ACCESS(READ)
PERMIT 'DSNC810.BSDS*'    ID(SVCAID) ACCESS(READ)
```

- For DSN1CHKR:

```
PERMIT 'DSNC810.DSNDBDC.*' ID(SVCAID) ACCESS(READ)
```

- For Change Log Inventory:

```
PERMIT 'DSNC810.BSDS*' ID(SVCAID) ACCESS(CONTROL)
```

- For Print Log Map:

```
PERMIT 'DSNC810.BSDS*' ID(SVCAID) ACCESS(READ)
```

The level of access depends on the intended use, not on the type of data set (VSAM KSDS, VSAM linear, or sequential). For update operations, ACCESS(CONTROL) is required; for read-only operations, ACCESS(READ) is sufficient.

You can use RACF to permit programs, rather than user IDs, to access objects. When you use RACF in this manner, IDs that are not authorized to access the log data sets might be able to do so by running the DSN1LOGP utility. Permit access to database data sets through DSN1PRNT or DSN1COPY.

Permitting DB2 authorization IDs to use the profiles

Authorization IDs with installation SYSADM or installation SYSOPR authority need access to most DB2 data sets. (For a list of the privileges that go with those authorities, see “Explicit privileges and authorities” on page 133.) The following command adds the two default IDs that have the SYSADM and SYSOPR authorities if no other IDs are named when DB2 is installed:

```
ADDUSER (SYSADM SYSOPR)
```

The next two commands connect those IDs to the groups that control data sets, with the authority to create new RACF database profiles. The ID that has installation SYSOPR authority (SYSOPR) does not need that authority for the installation data sets.

```
CONNECT (SYSADM SYSOPR)  GROUP(DSNC810) AUTHORITY(CREATE) UACC(NONE)
CONNECT (SYSADM)         GROUP(DSN810)  AUTHORITY(CREATE) UACC(NONE)
```

The following set of commands gives the IDs complete control over DSNC810 data sets. The system administrator IDs also have complete control over the installation libraries. Additionally, you can give the system programmer IDs the same control.

```
PERMIT 'DSNC810.LOGCOPY*' ID(SYSADM SYSOPR) ACCESS(ALTER)
PERMIT 'DSNC810.ARCHLOG*' ID(SYSADM SYSOPR) ACCESS(ALTER)
PERMIT 'DSNC810.BSDS*'    ID(SYSADM SYSOPR) ACCESS(ALTER)
PERMIT 'DSNC810.DSNDBC.*' ID(SYSADM SYSOPR) ACCESS(ALTER)
PERMIT 'DSNC810.*'        ID(SYSADM SYSOPR) ACCESS(ALTER)
PERMIT 'DSN810.*'         ID(SYSADM)        ACCESS(ALTER)
```

Allowing DB2 authorization IDs to create data sets

The following command connects several IDs, which are already connected to the DB2USER group, to group DSNC810 with CREATE authority:

```
CONNECT (USER1 USER2 USER3 USER4 USER5)
         GROUP(DSNC810) AUTHORITY(CREATE) UACC(NONE)
```

Those IDs can now explicitly create data sets whose names have DSNC810 as the high-level qualifier. Any such data sets that are created by DB2 or by these RACF user IDs are protected by RACF. Other RACF user IDs are prevented by RACF from creating such data sets.

If no option is supplied for PASSWORD on the ADDUSER command that adds those IDs, the first password for the new IDs is the name of the default group, DB2USER. The first time that the IDs sign on, they all use that password, but they must change the password during their first session.

Chapter 13. Auditing

This chapter provides answers to some fundamental auditing questions. Foremost among them are the following questions:
1. Who is privileged to access what objects?
2. Who has actually accessed the data?

Who is privileged? You can find answers to the first question in the DB2 catalog, which is the primary audit trail for the DB2 subsystem. Most of the catalog tables describe the DB2 objects, such as tables, views, table spaces, packages, and plans. Several other tables (every table with the character string "AUTH" in its name) hold records of every granted privilege or authority. Every catalog record of a grant contains the following information:
- Name of the object
- Type of privilege
- IDs that receive the privilege
- ID that grants the privilege
- Time of the grant
- Other information

You can retrieve data from catalog tables by writing SQL queries. For examples, see "Finding catalog information about privileges" on page 187.

Who accessed data? You can find answers to the second question by using the audit trace, another important audit trail for DB2. The audit trace can record:
- Changes in authorization IDs
- Changes to the structure of data (such as dropping a table)
- Changes to data values (such as updating or inserting records)
- Access attempts by unauthorized IDs
- Results of GRANT statements and REVOKE statements
- Mapping of Kerberos security tickets to IDs
- Other activities that are of interest to auditors

This chapter presents the following topics:

- "Using the audit trace"
- "The role of authorization IDs in auditing" on page 289
- "Using audit records" on page 290.
- "Using other sources of audit information" on page 292
- "Determining which security measures are enabled" on page 292
- "Ensuring data accuracy and consistency" on page 293
- "Determining whether data is consistent" on page 296

Using the audit trace

The information in this chapter is General-use Programming Interface and Associated Guidance Information, as defined in "Notices" on page 1301.

The DB2 audit trace can indicate who has accessed data. When started, the audit trace records certain types of actions and sends the report to a named destination. As with other DB2 traces, you can choose the following options for the audit trace:
- Categories of events to trace
- Particular authorization IDs or plan IDs to audit
- Ways to start and stop the trace

© Copyright IBM Corp. 1982, 2008

- Destinations for audit records

You can choose whether to audit the activity on a table by specifying an option of the CREATE and ALTER statements.

Starting the audit trace

You can automatically start an audit trace whenever DB2 is started by setting the AUDIT TRACE field on the DSNTIPN installation panel to one of the following options:

- * (an asterisk) to provide a complete audit trace.
- NO, the default, if you do not want an audit trace to start automatically.
- YES to start a trace automatically for the default class (class 1: access denials) and the default destination (the SMF data set).
- A list of audit trace classes (for example, 1,3,5) to start a trace automatically for those classes. This option uses the default destination.

The START TRACE command: As with other DB2 traces, you can start an audit trace at any time with the START TRACE command. You can choose the audit classes to trace and the destination for trace records. You can also include an identifying comment.

Example: The following command starts an audit trace for classes 4 and 6 with distributed activity:

```
-START TRACE (AUDIT) CLASS (4,6) DEST (GTF) LOCATION (*)
  COMMENT ('Trace data changes; include text of dynamic DML statements.')
```

Stopping the audit trace

You can have several different traces that run at the same time, including more than one audit trace. One way to stop a particular trace is to issue the STOP TRACE command with the same options that were used for START TRACE. You must include enough options to uniquely identify a particular trace.

Example: The following command stops the trace that the example in “Starting the audit trace” started:

```
-STOP TRACE (AUDIT) CLASS (4,6) DEST (GTF)
```

If you did not saved the START command, you can determine the trace number and stop the trace by its number. Use DISPLAY TRACE to find the number.

Example: DISPLAY TRACE (AUDIT) might return a message like the following output:

```
TNO  TYPE     CLASS         DEST     QUAL
01   AUDIT    01            SMF      NO
02   AUDIT    04,06         GTF      YES
```

The message indicates that two audit traces are active. Trace 1 traces events in class 1 and sends records to the SMF data set. Trace 1 can be a trace that starts automatically whenever DB2 starts. Trace 2 traces events in classes 4 and 6 and sends records to GTF.

You can stop either trace by using its identifying number (TNO).

Example: To stop trace 1, use the following command:

```
-STOP TRACE AUDIT TNO(1)
```

Audit class descriptions

When you start the trace, you choose which events to audit by specifying one or more audit classes. The trace records are limited to 5000 bytes, so descriptions that contain long SQL statements might be truncated. Table 86 describes the available classes and the events that they include.

Table 86. Audit classes and the events that they trace

Audit class	Events that are traced
1	Access attempts that DB2 denies because of inadequate authorization. This class is the default.
2	Explicit GRANT and REVOKE statements and their results. This class does not trace implicit grants and revokes.
3	CREATE, ALTER, and DROP statements that affect audited tables, and the results of these statements. This class traces the dropping of a table that is caused by DROP TABLESPACE or DROP DATABASE and the creation of a table with AUDIT CHANGES or AUDIT ALL. ALTER TABLE statements are audited only when they change the AUDIT option for the table.
4	Changes to audited tables. Only the first attempt to change a table, within a unit of recovery, is recorded. (If the agent or the transaction issues more than one COMMIT statement, the number of audit records increases accordingly.) The changed data is not recorded, only the attempt to make a change is recorded. If the change is not successful and is rolled back, the audit record remains; it is not deleted. This class includes access by the LOAD utility. Accesses to a dependent table that are caused by attempted deletions from a parent table are also audited. The audit record is written even if the delete rule is RESTRICT, which prevents the deletion from the parent table. The audit record is also written when the rule is CASCADE or SET NULL, which can result in deletions that cascade to the dependent table.
5	All read accesses to tables that are identified with the AUDIT ALL clause. As in class 4, only the first access within a DB2 unit of recovery is recorded. References to a parent table are also audited.
6	The bind of static and dynamic SQL statements of the following types: • INSERT, UPDATE, DELETE, CREATE VIEW, and LOCK TABLE statements for audited tables. Except for the values of host variables, the audit record contains the entire SQL statement. • SELECT statements on tables that are identified with the AUDIT ALL clause. Except for the values of host variables, the audit record contains the entire SQL statement.
7	Assignment or change of an authorization ID because of the following reasons: • Changes through an exit routine (default or user-written) • Changes through a SET CURRENT SQLID statement • An outbound or inbound authorization ID translation • An ID that is being mapped to a RACF ID from a Kerberos security ticket
8	The start of a utility job, and the end of each phase of the utility.
9	Various types of records that are written to IFCID 0146 by the IFI WRITE function.

Limitations of the audit trace

The audit trace does not record everything, as the following list of limitations indicates:

- The auditing that is described in this chapter takes place only when the audit trace is on.
- The trace does not record old data after it is changed because the log records old data.
- If an agent or transaction accesses a table more than once in a single unit of recovery, the audit trace records only the first access.
- The audit trace does not record accesses if you do not start the audit trace for the appropriate class of events.
- The audit trace does not audit some utilities. The trace audits the first access of a table with the LOAD utility, but it does not audit access by the COPY, RECOVER, and REPAIR utilities. The audit trace does not audit access by stand-alone utilities, such as DSN1CHKR and DSN1PRNT.
- The trace audits only the tables that you specifically choose to audit.
- You cannot audit access to auxiliary tables.
- You cannot audit the catalog tables because you cannot create or alter catalog tables.

This auditing coverage is consistent with the goal of providing a moderate volume of audit data with a low impact on performance. However, when you choose classes of events to audit, consider that you might ask for more data than you are willing to process.

Auditing in a distributed data environment

The DB2 audit trace records any access to your data, whether the request is from a remote location or from your local DB2. The trace record for a remote request reports the authorization ID as the final result of one of the following conditions:

- An outbound translation
- An inbound translation
- Activity of an authorization exit routine

Essentially, the ID on a trace record for a remote request is the same as the ID to which you grant access privileges for your data.

Requests from your location to a remote DB2 are audited only if an audit trace is active at the remote location. The output from the trace appears only in the records at that location.

Auditing a specific table

To audit a table, use the AUDIT clause in the CREATE TABLE statement or the ALTER TABLE statement.

Example: DB2 audits the department table whenever the audit trace is on if you create the table with the following statement:

```
CREATE TABLE DSN8810.DEPT
      (DEPTNO    CHAR(3)           NOT NULL,
       DEPTNAME  VARCHAR(36)       NOT NULL,
       MGRNO     CHAR(6)                   ,
       ADMRDEPT  CHAR(3)           NOT NULL,
       LOCATION  CHAR(16)                  ,
       PRIMARY KEY (DEPTNO)                )
  IN DSN8D81A.DSN8S81D
  AUDIT CHANGES;
```

Because this statement includes the AUDIT CHANGES option, DB2 audits the table for each access that inserts, updates, or deletes data (trace class 4).

Example: To also audit the table for read accesses (class 5), issue the following statement:

```
ALTER TABLE DSN8810.DEPT
  AUDIT ALL;
```

The statement is effective regardless of whether the table was previously chosen for auditing.

Example: To prevent all auditing of the table, issue the following statement:

```
ALTER TABLE DSN8810.DEPT
  AUDIT NONE;
```

For the CREATE TABLE statement, the default audit option is NONE. For the ALTER TABLE statement, no default option exists. If you do not use the AUDIT clause in an ALTER TABLE statement, the audit option for the table is unchanged.

When CREATE TABLE statements or ALTER TABLE statements affect the audit of a table, you can audit those statements. However, the results of those audits are in audit class 3, not in class 4 or class 5. Use audit class 3 to determine whether auditing was turned off for a table for an interval of time.

If an ALTER TABLE statement turns auditing on or off for a specific table, any plans and packages that use the table are invalidated and must be rebound. If you change the auditing status, the change does not affect plans, packages, or dynamic SQL statements that are currently running. The change is effective only for plans, packages, or dynamic SQL statements that begin running after the ALTER TABLE statement has completed.

The role of authorization IDs in auditing

In general, audit traces identify a process by its primary authorization ID. The audit trace records the primary ID before and after invocation of an authorization exit routine. Therefore, you can identify the primary ID that is associated with a change.

Exception: If a primary ID has been translated many times, you might not be able to identify the primary ID that is associated with a change.

Example: Suppose that the server does not recognize the translated ID from the requesting site. In this case, you cannot use the primary ID to gather all audit records for a user that accesses remote data.

The AUTHCHG record shows the values of all secondary authorization IDs that are established by an exit routine. See “Audit class descriptions” on page 287, Audit Class 7, for a description of the AUTHCHG record.

With the audit trace, you can also determine which primary ID is responsible for the action of a secondary ID or a current SQL ID.

Example: Suppose that the user with primary ID SMITHJ sets the current SQL ID to TESTGRP to grant privileges over the table TESTGRP.TABLE01 to another user. The DB2 catalog records the grantor of the privileges as TESTGRP. However, the audit trace shows that SMITHJ issued the grant statement.

Recommendation: Consider carefully the consequences of altering that ID by using an exit routine because the trace identifies a process by its primary ID. If the

primary ID identifies a unique user, individual accountability is possible. However, if several users share the same primary ID, you cannot tell which user issues a particular GRANT statement or runs a particular application plan.

Auditing specific IDs

As with other DB2 traces, you can start an audit trace for a particular plan name, a particular primary authorization ID, or a combination of the two. For examples of auditing specific IDs, see "DB2 trace" on page 1189. You might consider having audit traces on at all times for IDs with SYSADM authority because they have complete access to every table. If you have a network of DB2 subsystems, you might need to trace multiple authorization IDs if primary authorization ID are translated several times. For embedded SQL, the audited ID is the primary authorization ID of the person who bound the plan or package. For dynamic SQL, the audited ID is the primary authorization ID.

Determining which IDs hold privileges and authorities

Auditors must be aware of the privileges and authorities that are associated with IDs in the DB2 subsystem. To determine which IDs hold privileges and authorities, use the following methods:

- Query the DB2 catalog to determine which IDs hold particular privileges. See "Finding catalog information about privileges" on page 187 for information about querying the catalog.
- Check on individual IDs that are associated with group IDs. Some authorization IDs that you encounter are probably group IDs, to which many individual IDs can be connected. To see which IDs are connected to a group, obtain a report from RACF or from whatever external security system you are using. These reports can tell you which IDs have the required privileges to use DB2 data sets and other resources. For instructions on how to obtain such reports, see *z/OS Security Server RACF System Programmer's Guide* or the documentation for your external security system.

See Chapter 9, "Controlling access to DB2 objects," on page 133 for more information about privileges and authorities.

Using audit records

You can prepare the System Management Facility (SMF) or Generalized Trace Facility (GTF) to accept audit trace records in the same way that you prepare performance or accounting trace records. See "Recording SMF trace data" on page 1193 and "Recording GTF trace data" on page 1195 for information. The records are of SMF type 102, as are performance trace records.

IFCIDs identify all DB2 trace records. For instructions on interpreting trace output and mapping records for the IFCIDs, see Appendix D, "Interpreting DB2 trace output," on page 1135. Chapter 2 of *DB2 Command Reference* lists the IFCIDs for each trace class and the description of the START TRACE command.

If you send trace records to SMF (the default), data might be lost in the following circumstances:

- SMF fails while DB2 continues to run.
- An unexpected abend (such as a TSO interrupt) occurs while DB2 is transferring records to SMF.

In those circumstances, SMF records the number of records that are lost. z/OS provides an option to stop the system rather than to lose SMF data.

The contents of audit records

Among other things, the audit trace records can indicate the following information:

- The ID that initiated the activity
- The LOCATION of the ID that initiated the activity (if the access was initiated from a remote location)
- The type of activity and the time that the activity occurred
- The DB2 objects that were affected
- Whether access was denied
- The owner of a particular plan and package
- The database alias (DBALIAS) that was used to access a remote location or a location alias that was accepted from a remote application

Formatting audit records

To extract, format, and print the records, you can use any of the following methods:

- Use the DB2 Performance Expert. See "OMEGAMON" on page 1196 for more information.
- Write your own application program to access the SMF data.
- Use the instrumentation facility interface (IFI) as an online resource to retrieve audit records. For more information about using the IFI, see Appendix E, "Programming for the Instrumentation Facility Interface (IFI)," on page 1151.

Suggested audit trace reports

If you regularly start the audit trace for all classes, you accumulate data from which to make reports. Consider producing audit trace reports that focus on the following important security events:

Use of sensitive data
You should define tables that contain sensitive data, such as employee salary records, with the AUDIT ALL option. You can report use by table and by authorization ID to look for access by unusual IDs, at unusual times, or of unexpected types. You should also record any ALTER or DROP operations that affect the data. Use audit classes 3, 4, and 5.

Grants of critical privileges
Carefully monitor IDs with special authorities, such as SYSADM and DBADM. Also carefully monitor IDs with privileges over sensitive data, such as an update privilege on records of accounts payable. You can query the DB2 catalog to determine which IDs hold privileges and authorities at a particular time. To determine which IDs received privileges and then had them revoked, use audit class 2 and consult the audit records.

Unsuccessful access attempts
Investigate all unsuccessful access attempts. Although some access failures are only user errors, others can be attempts to violate security. If you have sensitive data, always use trace audit class 1. You can report by table or by authorization ID.

Using other sources of audit information

In addition to the audit trace, DB2 offers the following other sources of audit information:

Additional DB2 traces
DB2 accounting, statistics, and performance traces are also available. You can read about the accounting trace, the statistics trace, and performance traces in "DB2 trace" on page 1189. You can also use DB2 Performance Expert to print reports of these traces. See "OMEGAMON" on page 1196. for information about DB2 Performance Expert.

Recovery log
Although the recovery log is not an all-purpose log, it can be useful for auditing. You can print information from the log by using the DSN1LOGP utility. For example, the summary report can show which table spaces have been updated within the range of the log that you scan. The REPORT utility can indicate what log information is available and where it is located. For information about running DSN1LOGP and REPORT, see *DB2 Utility Guide and Reference*.

Image copies of table spaces
Typical recovery procedures generate image copies of table spaces. You can inspect these copies, or use them with the RECOVER utility to recover a table space to a particular point in time. If you recover to a point in time, you narrow the time period during which a particular change could have been made. For guidance in using COPY and RECOVER, see Chapter 21, "Backing up and recovering databases," on page 471.

z/OS console log
The z/OS console log contains messages about exceptional conditions that are encountered during DB2 operation. Inspect this log for symptoms of problems.

Audit trail
Whenever you install a new version, release, or maintenance of DB2, an automatic record provides an audit trail. The audit trail helps you determine whether the changes are appropriate and whether authorized personnel made them. The audit trail can also aid in investigation of application-related problems.

Determining which security measures are enabled

Auditors must know which security measures are enabled on the DB2 subsystem. You can determine whether DB2 authorization checking, the audit trace, and data definition control are enabled in the following ways:

Audit trace
To see whether the trace is running, display the status of the trace by the command DISPLAY TRACE(AUDIT).

DB2 Authorization checking
Follow the instructions for changing DB2 installation parameters in "The Update Process" in Part 2 of *DB2 Installation Guide*. Without changing anything, look at panel DSNTIPP. If the value of the USE PROTECTION field is YES, DB2 checks privileges and authorities before permitting any activity.

Data definition control
Data definition control is a security measure that provides additional

constraints to existing authorization checks. With it, you control how specific plans or collections of packages can use data definition statements. Read Chapter 10, "Controlling access through a closed application," on page 215 for a description of this function. To determine whether data definition control is active, look at option 1 on the DSNTIPZ installation panel.

Ensuring data accuracy and consistency

DB2 provides many controls that you can apply to data entry and update. Some of the controls are automatic; some are optional. All of the controls prohibit certain operations and provide error or warning messages if those operations are attempted. This section describes typical auditing concerns for ensuring data accuracy and consistency.

The set of techniques in this section is not exhaustive. Other combinations of techniques are possible. For example, you can use table check constraints or a view with the check option to ensure that data values are members of a certain set. Or you can set up a master table and define referential constraints. You can also enforce the controls through application programs, and restrict the INSERT and UPDATE privileges only to those programs.

Ensuring that the required data is present

To ensure that the required data is present, define columns with the NOT NULL clause.

You can also control the type of data by assigning column data types and lengths.

Example: Alphabetic data cannot be entered into a column with one of the numeric data types. Data that is inserted into a DATE column or a TIME column must have an acceptable format, and so on.

For suggestions about assigning column data types and the NOT NULL attribute, see *DB2 SQL Reference*.

Ensuring that data is unique

In some cases, you must ensure that the data in a column or a set of columns is unique. The preferred method for ensuring that data values are unique is to create a unique index on a column or set of columns. See *The Official Introduction to DB2 UDB for z/OS* for suggestions about indexes.

Ensuring that data fits a pattern or value range

Triggers and table check constraints enhance the ability to control data integrity.

Triggers are very powerful for defining and enforcing rules that involve different states of DB2 data. For example, a rule can prevent a salary column from more than a ten percent increase. A trigger can enforce this rule and provide the value of the salary before and after the increase for comparison. See Chapter 5 of *DB2 SQL Reference* for information using the CREATE TRIGGER statement to create a trigger.

A check constraint designates the values that specific columns of a base table can contain. A check constraint can express simple constraints, such as a required pattern or a specific range, and rules that refer to other columns of the same table.

As an auditor, you might check that the table definition expresses required constraints on column values as table check constraints. For a full description of the rules for those constraints, see the CREATE TABLE information in Chapter 5 of *DB2 SQL Reference*.

An alternative technique is to create a view with the check option, and then insert or update values only through that view.
Example: Suppose that, in table T, data in column C1 must be a number between 10 and 20. Suppose also that data in column C2 is an alphanumeric code that must begin with A or B. Create view V1 with the following statement:

```
CREATE VIEW V1 AS
  SELECT * FROM T
    WHERE C1 BETWEEN 10 AND 20
    AND  (C2 LIKE 'A%' OR C2 LIKE 'B%')
  WITH CHECK OPTION;
```

Because of the CHECK OPTION, view V1 allows only data that satisfies the WHERE clause.

You cannot use the LOAD utility with a view, but that restriction does not apply to user-written exit routines. Several types of user-written routines are pertinent here:

Validation routines
You can use validation routines to validate data values. Validation routines access an entire row of data, check the current plan name, and return a nonzero code to DB2 to indicate an invalid row.

Edit routines
Edit routines have the same access as validation routines, and can also change the row that is to be inserted. Auditors typically use edit routines to encrypt data and to substitute codes for lengthy fields. However, edit routines can also validate data and return nonzero codes.

Field procedures
Field procedures access data that is intended for a single column; they apply only to short-string columns. However, they accept input parameters, so generalized procedures are possible. A column that is defined with a field procedure can be compared only to another column that uses the same procedure.

See Appendix B, “Writing exit routines,” on page 1051 for information about using exit routines.

Ensuring that data is consistent

Referential integrity ensures that data is consistent across tables. When you define primary and foreign keys, DB2 automatically enforces referential integrity. Therefore, every value of a foreign key in a dependent table must be a value of a primary key in the appropriate parent table.

Recommendation: Use referential integrity to ensure that a column allows only specific values. Set up a master table of allowable values, and define its primary key. Define foreign keys in other tables that must have matching values in their columns. In most cases, you should use the SET NULL delete rule.

DB2 does not enforce informational referential constraints across subsystems. For information about the means, implications, and limitations of enforcing referential integrity, see *DB2 Application Programming and SQL Guide*.

Ensuring that changes are tracked

Triggers offer an efficient means of maintaining an audit trail. You can define a trigger to activate in response to certain DELETE, INSERT, or UPDATE statements that change data.

You can qualify a trigger by providing a list of column names when you define the trigger. The qualified trigger is activated only when one of the named columns is changed. A trigger that performs validation for changes that are made in an UPDATE operation must access column values both before and after the update. Transition variables (available only to row triggers) contain the column values of the row change that activated the trigger. The old column values and the column values from after the triggering operation are both available.

See *DB2 SQL Reference* for information about when to use triggers.

Ensuring that concurrent users access consistent data

Locks can ensure that data remains consistent, even when multiple users try to access the same data at the same time. From an auditing standpoint, you can use locks to ensure that only one user is privileged to change data at a given time. You can also ensure that no users are privileged to access uncommitted data.

If you use repeatable read (RR), read stability (RS), or cursor stability (CS) as your isolation level, DB2 automatically controls access to data by using locks. However, if you use uncommitted read (UR) as your isolation level, users can access uncommitted data and introduce inconsistent data. Auditors must know which applications use UR isolation, and they must know whether these applications can introduce inconsistent data or create security risks.

For static SQL, you can determine which plans and packages use UR isolation by querying the catalog.

Example: For static SQL statements, use the following query to determine which plans use UR isolation:

```
SELECT DISTINCT Y.PLNAME
   FROM SYSIBM.SYSPLAN X, SYSIBM.SYSSTMT Y
   WHERE (X.NAME = Y.PLNAME AND X.ISOLATION = 'U')
       OR Y.ISOLATION = 'U'
   ORDER BY Y.PLNAME;
```

Example: For static SQL statements, use the following query to determine which packages use UR isolation:

```
SELECT DISTINCT Y.COLLID, Y.NAME, Y.VERSION
   FROM SYSIBM.SYSPACKAGE X, SYSIBM.SYSPACKSTMT Y
   WHERE (X.LOCATION = Y.LOCATION AND
          X.LOCATION = ' '        AND
          X.COLLID   = Y.COLLID   AND
          X.NAME     = Y.NAME     AND
          X.VERSION  = Y.VERSION  AND
          X.ISOLATION = 'U')
       OR Y.ISOLATION = 'U'
   ORDER BY Y.COLLID, Y.NAME, Y.VERSION;
```

For dynamic SQL statements, turn on performance trace class 3 to determine which plans and packages use UR isolation.

For more information about locks and concurrency, see Chapter 31, “Improving concurrency,” on page 809.

Consistency between systems: When an application program writes data to both DB2 and IMS, or to both DB2 and CICS, the subsystems prevent concurrent use of data until the program declares a point of consistency. For a detailed description of how data remains consistent between systems, see "Multiple system consistency" on page 455.

Checking for lost and incomplete transactions

Database balancing is a technique that can alert you to lost and incomplete transactions. Database balancing determines, for each set of data, whether the opening balance, the control totals, and the processed transactions equal the closing balance and control totals.

DB2 has no automatic mechanism to calculate control totals and column balances and compare them with transaction counts and field totals. Therefore, to use database balancing, you must design these mechanisms into the application program.

Example: Use your application program to maintain a control table. The control table contains information to balance the control totals and field balances for update transactions against a user's view. The control table might contain these columns:

- View name
- Authorization ID
- Number of logical rows in the view (not the same as the number of physical rows in the table)
- Number of insert transactions and update transactions
- Opening balances
- Totals of insert transaction amounts and update transaction amounts
- Relevant audit trail information such as date, time, workstation ID, and job name

The program updates the transaction counts and amounts in the control table each time it completes an insert or update to the view. To maintain coordination during recovery, the program commits the work only after it updates the control table. After the application processes all transactions, the application writes a report that verifies the control total and balancing information.

Determining whether data is consistent

When you control data entry, you perform only part of a complete security and auditing policy. You must also verify the results when data is accessed and changed. The following sections describe methods for determining whether your data is consistent:

- "Automatically checking the consistency of data" on page 297
- "Submitting SQL queries to check data consistency" on page 297
- "Checking data consistency with the CHECK utility" on page 297
- "Checking data consistency with the DISPLAY DATABASE command" on page 298
- "Checking data consistency with the REPORT utility" on page 298
- "Checking data consistency with the operation log" on page 298
- "Using internal integrity reports to check data consistency" on page 298

Automatically checking the consistency of data

Whenever an operation changes the contents of a data page or an index page, DB2 verifies that the modifications do not produce inconsistent data. Additionally, you can run the DSN1CHKR utility to verify the integrity of the DB2 catalog and the directory table spaces. You can also run this utility to scan the specified table space for broken links, damaged hash chains, or orphan entries. For more information, see Part 3 of *DB2 Utility Guide and Reference*.

Submitting SQL queries to check data consistency

General-use Programming Interface

If you suspect that a table contains inconsistent data, you can submit an SQL query to search for a specific type of error.

Example: Consider the view that is created in "Ensuring that data fits a pattern or value range" on page 293. The view allows an insert or update to table T1 only if the value in column C1 is between 10 and 20 and if the value in C2 begins with A or B. To check that the control has not been bypassed, issue the following statement:

```
SELECT * FROM T1
  WHERE  NOT (C1 BETWEEN 10 AND 20
  AND  (C2 LIKE 'A%' OR C2 LIKE 'B%'));
```

If the control has not been bypassed, DB2 returns no rows and thereby confirms that the contents of the view are valid.

You can also use SQL statements to get information from the DB2 catalog about referential constraints that exist. For several examples, see *DB2 SQL Reference*.

End of General-use Programming Interface

Checking data consistency with the CHECK utility

The CHECK utility helps to ensure data consistency in the following ways:

CHECK INDEX
The CHECK INDEX utility checks the consistency of indexes with the data to which the indexes point. It determines whether each index pointer points to a data row with the same value as the index key. If an index key points to a LOB, the CHECK INDEX utility determines whether the index key points to the correct LOB.

CHECK DATA
The CHECK DATA utility checks referential constraints (but not informational referential constraints). It determines whether each foreign key value in each row is a value of the primary key in the appropriate parent table.

The CHECK DATA utility also checks table check constraints and checks the consistency between a base table space and any associated LOB table spaces. It determines whether each value in a row is within the range that was specified for that column when the table was created.

CHECK LOB
The CHECK LOB utility checks the consistency of a LOB table space. It determines whether any LOBs in the LOB table space are invalid.

See *DB2 Utility Guide and Reference* for more information about the CHECK utility.

Checking data consistency with the DISPLAY DATABASE command

If you allow a table to be loaded without enforcing referential constraints on its foreign key columns, the table might contain data that violates the constraints. DB2 places the table space that contains the table in the CHECK-pending status. You can determine which table spaces are in CHECK-pending status by using the DISPLAY DATABASE command with the RESTRICT option. You can also use the DISPLAY DATABASE command to display table spaces with invalid LOBs. See Chapter 2 of *DB2 Command Reference* for information about using this command.

Checking data consistency with the REPORT utility

You can use the REPORT utility to determine:

- Which table spaces contain a set of tables that are interconnected by referential constraints
- Which LOB table spaces are associated with which base tables

See *DB2 Utility Guide and Reference* for information about using the REPORT utility.

Checking data consistency with the operation log

You can use the operation log to verify that DB2 is operated reliably, and to reveal unauthorized operations and overrides. The operation log consists of an automated log of DB2 operator commands (such as those that start or stop the subsystem or its databases) and any abend of DB2. The operation log records the following information:

- Command or condition type
- Date and time when the command was issued
- Authorization ID that issued the command
- Database connection code

You can obtain this information from the system log (SYSLOG), the SMF data set, or the automated job scheduling system. To obtain the information, use SMF reporting, job-scheduler reporting, or a user-developed program. You should review the log report daily and keep a history file for comparison. Because abnormal DB2 termination can indicate integrity problems, you should implement an immediate notification procedure to alert the appropriate personnel (DBA, systems supervisor, and so on) of abnormal DB2 terminations.

Using internal integrity reports to check data consistency

You can generate internal integrity reports for application programs and for utilities.

For application programs: You should record any DB2 return codes that indicate possible data integrity problems, such as inconsistency between index and table information, physical errors on database disk, and so on. All programs must check the SQLCODE or the SQLSTATE for the return code that is issued after an SQL statement is run. DB2 records, on SMF, the occurrence (but not the cause) of physical disk errors and application program abends. The program can retrieve and report this information; the system log (SYSLOG) and the DB2 job output also have this information. However, in some cases, only the program can provide enough detail to identify the exact nature of problem.

You can incorporate these integrity reports into application programs, or you can use them separately as part of an interface. The integrity report records the incident in a history file and writes a message to the operator's console, a database administrator's TSO terminal, or a dedicated printer for certain codes. The recorded information includes:

- Date
- Time
- Authorization ID
- Terminal ID or job name
- Application
- Affected view or affected table
- Error code
- Error description

Review reports frequently by time and by authorization ID.

For utilities: When a DB2 utility reorganizes or reconstructs data in the database, it produces statistics to verify record counts and to report errors. The LOAD and REORG utilities produce data record counts and index counts to verify that no records were lost. In addition to that, keep a history log of any DB2 utility that updates data, particularly REPAIR. Regularly produce and review these reports, which you can obtain through SMF customized reporting or a user-developed program.

Chapter 14. A sample security plan for employee data

This chapter shows one approach to enforcing a security plan by using authorization IDs, implicit privileges, granted privileges and authorities, and the audit trace. Do not view this sample security plan as an exact model for your security plan. Instead, use this sample to understand various possibilities and problem areas that you might encounter when you implement your security plan.

Suppose that the Spiffy Computer Company management team determines the following security objectives:

- Managers can see, but not update, all of the employee data for members of their own departments. Managers of managers can see all of the data for employees of departments that report to them.
- The employee table resides at a central location. Managers at remote locations can query the data in that table.
- The payroll operations department makes changes to the employee table. Members of the payroll operations department can update any column of the employee table except for the salary, bonus, and commission columns. Furthermore, members of payroll operations can update any row except for rows that are for members of their own department. Because changes to the table are made only from a central location, distributed access does not affect payroll operations.
- Changes to the salary, bonus, and commission columns are made through a process that involves the payroll update table. When an employee's compensation changes, a member of the payroll operations department can insert rows in the payroll update table. For example, a member of the payroll operations department might insert a row in the compensation table that lists an employee ID and an updated salary. Next, the payroll management group can verify inserted rows and transfer the changes to the employee table.
- No one else can see the employee data. The security plan cannot fully achieve this objective because some ID must occasionally exercise SYSADM authority. While exercising SYSADM authority, an ID can retrieve any data in the system. The security plan uses the trace facility to monitor the use of that power.

The following sections discuss how to implement a security plan that accomplishes the objectives:

- "Securing manager access"
- "Securing payroll operations access" on page 305
- "Securing administrator, owner, and other access" on page 308

Securing manager access

The Spiffy Computer Company imposes the following security restrictions on managers:

- Managers can retrieve, but not change, all information in the employee table for members of their own departments.
- Managers of managers have the same privileges for their own departments and for the departments that directly report to them.

Spiffy Computer Company can most easily implement these restrictions by using views.

© Copyright IBM Corp. 1982, 2008

Example: To create a view of employee data for every employee that reports to a manager, the Spiffy security planners perform the following steps:

1. Add a column that contains manager IDs to DSN8810.DEPT, as shown in the following statement:

```
ALTER TABLE DSN8810.DEPT
   ADD MGRID CHAR(8) FOR SBCS DATA NOT NULL WITH DEFAULT;
```

2. Create a view that selects employee information about employees that work for a given manager, as shown in the following statement:

```
CREATE VIEW DEPTMGR AS
   SELECT * FROM DSN8810.EMP, DSN8810.DEPT
      WHERE WORKDEPT = DEPTNO
      AND MGRID = USER;
```

3. Ensure that every manager has the SELECT privilege on the view. See "Granting the SELECT privilege to managers" for information about ensuring that every manager has the appropriate SELECT privilege.

Granting the SELECT privilege to managers

The security planners for Spiffy Computer Company can take one of two approaches when they grant the SELECT privilege on a view to managers:

Individual approach
: Grant privileges to individual IDs, and revoke them if the user of the ID leaves the company or transfers to another position.

Functional approach
: Create RACF groups, and grant privileges to the group IDs, with the intention of never revoking them. When an individual ID needs those privileges, connect that ID to the group; disconnect the ID when its user leaves or transfers.

The Spiffy security planners know that the functional approach is usually more convenient in the following situations:

- Each function, such as the manager function, requires many different privileges. When functional privileges are revoked from one user, they must be granted to another user.
- Several users need the same set of privileges.
- The privileges are given with the grant option, or the privileges let users create objects that must persist after their original owners leave or transfer. In both cases, revoking the privileges might not be appropriate. The revokes cascade to other users. To change ownership, you might need to drop objects and re-create them.

Some of the Spiffy requirements for securing manager access suggest the functional approach. However, in this case, the function needs only one privilege. The privilege does not carry the grant option, and the privilege does not allow new objects to be created.

Therefore, the Spiffy security planners choose the individual approach, and plan to re-examine their decision later. Spiffy security planners grant all managers the SELECT privilege on the views for their departments.

Example: To grant the SELECT privilege on the DEPTMGR view to the manager with ID EMP0060, they use the following GRANT statement:

```
GRANT SELECT ON DEPTMGR TO EMP0060;
```

Securing distributed access

Some managers must use views to query data in the central employee table from remote locations. Furthermore, the security plan must ensure that this type of distributed access is secure. To secure distributed access to employee data, the Spiffy security planners must address the following questions:

- Which IDs should hold privileges on which views?
- How do the central location and the remote locations divide security responsibilities for IDs?

The Spiffy security planners answer those questions with the following decisions:

- IDs that are managed at the central location hold privileges on views for departments that are at remote locations. For example, the ID MGRD11 has the SELECT privilege on the view DEPTD11.
- If the manager of Department D11 uses a remote system, the ID at that system must be translated to MGRD11. Then a request is sent to the central system. All other IDs are translated to CLERK before they are sent to the central system.
- The communications database (CDB) manages the translated IDs, like MGRD11.
- An ID from a remote system must be authenticated on any request to the central system.

The following sections describe how the Spiffy security planners plan to implement their distributed access plan:

- "Actions at the central server location"
- "Actions at remote locations" on page 304.

Actions at the central server location

The following actions must occur at the central server location to enable distributed access that the Spiffy security plan requires:

- The central DB2 subsystem must authenticate every incoming ID with RACF.
- For SNA connections, the Spiffy security planners must include an entry in table SYSIBM.LUNAMES in the CDB; the entry in the LUNAME column identifies the LU name of every remote location. The entry must specify that connections must be verified.

  Example: Table 87 shows an entry in SYSIBM.LUNAMES for LUREMOTE.

  Table 87. The SYSIBM.LUNAMES table at the central location

LUNAME	USERNAMES	SECURITY_IN	ENCRYPTPSWDS
LUREMOTE	blank	V	N

  The value of V for SECURITY_IN indicates that incoming remote connections must include verification. The value of N for ENCRYPTPSWDS indicates that passwords are not in internal RACF encrypted format.

  The security plan treats all remote locations alike, so it does not require encrypted passwords. The option to require encrypted passwords is available only between two DB2 subsystems that use SNA connections.
- For TCP/IP connections, the Spiffy security planners must set the TCP/IP ALREADY VERIFIED field of installation panel DSNTIP5 to NO. This setting ensures that the incoming requests that use TCP/IP are not accepted without authentication.
- The Spiffy security planners must grant all privileges and authorities that are required by the manager of Department D11 to the ID, MGRD11. The security planners must grant similar privileges to IDs that correspond to the remaining managers.

Actions at remote locations

The following actions must occur at the remote locations to enable distributed access for the Spiffy security plan:

- For SNA connections, the Spiffy security planners must include an entry in table SYSIBM.LUNAMES for the LU name of the central location. The entry must specify an outbound ID translation for attachment requests to that location.

  Example: Table 88 shows an entry in SYSIBM.LUNAMES for LUCENTRAL.

  Table 88. The SYSIBM.LUNAMES table at the remote location

LUNAME	USERNAMES	SECURITY_OUT
LUCENTRAL	O	R

  The value of O for USERNAMES indicates that translation checking is performed on outbound IDs, but not on inbound IDs. The value of R for SECURITY_OUT indicates that outbound connection requests contain a user ID and a RACF PassTicket.

- For TCP/IP connections, the Spiffy security planners must include an entry in table SYSIBM.IPNAMES for the LU name that is used by the central location. The content of the LUNAME column is used to generate RACF PassTickets. The entry must specify outbound ID translation for requests to that location.

  Example: Table 89 shows an entry in SYSIBM.IPNAMES for LUCENTRAL.

  Table 89. The SYSIBM.IPNAMES table at the remote location

LINKNAME	USERNAMES	SECURITY_OUT	IPADDR
LUCENTRAL	O	R	central.vnet.ibm.com

- The Spiffy security planners must include entries in table SYSIBM.USERNAMES to translate outbound IDs.

  Example: Table 90 shows two entries in SYSIBM.USERNAMES.

  Table 90. The SYSIBM.USERNAMES table at the remote location

TYPE	AUTHID	LINKNAME	NEWAUTHID
O	MEL1234	LUCENTRAL	MGRD11
O	blank	LUCENTRAL	CLERK

  MEL1234 is translated to MGRD11 before it is sent to the LU that is specified in the LINKNAME column. All other IDs are translated to CLERK before they are sent to that LU.

Exception: For a product other than DB2 UDB for z/OS, the actions at the remote location might be different. If you use a different product, check the documentation for that product. The remote product must satisfy the requirements that are imposed by the central subsystem.

Auditing manager use

Because Spiffy payroll data is extremely sensitive, the security plan requires that the audit trace is automatically started for all classes whenever DB2 is started. To ensure that an audit record exists for every access to the employee table, the Spiffy security planners create the employee table with AUDIT ALL. Every week, the security planners scan the records and determine the number of accesses by each manager.

The report highlights any number of accesses outside an expected range. The Spiffy system operator makes a summary of the reports every two months, and

scans it for unusual patterns of access. A large number of accesses or an unusual pattern might reveal use of a manager's logon ID by an unauthorized employee.

Securing payroll operations access

The Spiffy Computer Company imposes the following restrictions on members of the payroll operations department:

- Members of the payroll operations department can update any column of the employee table except for SALARY, BONUS, and COMM.
- Members of payroll operations can update any row except for rows that are for members of their own department.

Because changes to the table are made only from the central location, distributed access does not affect payroll operations.

The Spiffy security planners can most easily implement these security objectives for members of the payroll operations department by using views. The PAYDEPT view shows all the columns of the employee table except for job, salary, bonus, and commission. The view does not show the rows for members of the payroll operations department.

Example: The WORKDEPT value for the payroll operations department is P013. The owner of the employee table uses the following statement to create the PAYDEPT view:

```
CREATE VIEW PAYDEPT AS
  SELECT EMPNO, FIRSTNME, MIDINT, LASTNAME, WORKDEPT,
         PHONENO, HIREDATE, JOB, EDLEVEL, SEX, BIRTHDATE
    FROM DSN8810.EMP
    WHERE WORKDEPT<>'P013'
  WITH CHECK OPTION;
```

The CHECK OPTION ensures that every row that is inserted or updated through the view conforms to the definition of the view.

A second view, the PAYMGR view, gives Spiffy payroll managers access to any record, including records for the members of the payroll operations department.

Example: The owner of the employee table uses the following statement to create the PAYMGR view:

```
CREATE VIEW PAYMGR AS
  SELECT EMPNO, FIRSTNME, MIDINT, LASTNAME, WORKDEPT,
         PHONENO, HIREDATE, JOB, EDLEVEL, SEX, BIRTHDATE
    FROM DSN8810.EMP
  WITH CHECK OPTION;
```

Neither PAYDEPT nor PAYMGR provides access to compensation amounts. When a row is inserted for a new employee, the compensation amounts remain null. An update process can change these values at a later time. The owner of the employee table creates, owns, and grants privileges on both views. For information about granting privileges on these views, see “Granting privileges to payroll operations and payroll management” on page 307.

Securing compensation updates

The Spiffy security plan does not allow members of payroll operations to update compensation amounts directly. Instead, a separate payroll update table contains the employee ID, job, salary, bonus, and commission. Members of payroll operations make all job, salary, and bonus changes to the payroll update table,

except those for their own department. After they verify the prospective changes, the managers of payroll operations run an application program. The program reads the payroll update table and makes the corresponding changes to the employee table. Only the payroll update program has the privilege of updating job, salary, and bonus in the employee table.

Spiffy Computer Company calculates commission amounts separately by using a complicated formula. The formula considers the employee's job, department, years of service with the company, and responsibilities for various projects. The formula is embedded in the commission program, which is run regularly to insert new commission amounts in the payroll update table. The plan owner must have the SELECT privilege on the employee table and other tables to run the commission program.

Additional controls for compensation updates

By separating potential salary changes into the payroll update table, the Spiffy security planners allow payroll management to verify changes before they go into effect. At Spiffy Computer Company, managers check the changes against a written change request that is signed by a required level of management. The Spiffy security planners consider that check to be the most important control on salary updates, but the plan also includes the following controls:

- The employee ID in the payroll update table is a foreign key column that refers to the employee ID in the employee table. Enforcing the referential constraint prevents an employee ID from being changed to an invalid value.
- The employee ID in the payroll update table is also a primary key for that table. Therefore, the values in the employee ID column must be unique. Because of enforced uniqueness, every change that is made for any one employee during a given operating period must appear in the same row of the table. No two rows can carry conflicting changes.

The Spiffy security plan documents an allowable range of salaries, bonuses, and commissions for each job level. To keep the values within the allowable ranges, the Spiffy security planners use table check constraints for the salaries, bonuses, and commissions. The planners use this approach because it is both simple and easy to control. See Part 1 of *DB2 Application Programming and SQL Guide* for information about using table check constraints.

In a similar situation, you might also consider the following ways to ensure that updates and inserts stay within certain ranges:

- Keep the ranges in a separate DB2 table. To verify changes, query the payroll update table and the table of ranges. Retrieve any rows for which the planned update is outside the allowed range.
- Build the ranges into a validation routine. Apply the validation routine to the payroll update table to automatically reject any insert or update that is outside the allowed range.
- Embody the ranges in a view of the payroll table, using WITH CHECK OPTION, and make all updates to the view. The ID that owns the employee table also owns the view.
- Create a trigger to prevent salaries, bonuses, and commissions from increasing by more than the percent that is allowed for each job level. See *DB2 SQL Reference* for more information about using triggers.

Granting privileges to payroll operations and payroll management

The Spiffy security plan for the payroll operations department strongly suggests the functional approach, for the following reasons:

- Payroll operations members require several privileges, including the SELECT, INSERT, UPDATE, and DELETE privileges on the PAYDEPT view.
- Several members of the department require the same set of privileges.
- If members of the department leave, others are hired or transferred to replace the departing members.

Therefore, the security plan calls for a RACF group for the payroll operations department. DB2USER can define the group, as described in "Add RACF groups" on page 270. DB2USER can retain ownership of the group, or it can assign the ownership to an ID that is used by payroll management.

The owner of the employee table can grant the privileges that the group requires. The owner grants all required privileges to the group ID, with the intent not to revoke them. The primary IDs of new members of the department are connected to the group ID, which becomes a secondary ID for each of them. The primary IDs of members who leave the department are disconnected from the group ID.

Example: The following statement grants the SELECT, INSERT, UPDATE, and DELETE privileges on the PAYDEPT view to the payroll operations group ID PAYOPS:

```
GRANT SELECT, INSERT, UPDATE, DELETE ON PAYDEPT TO PAYOPS;
```

This statement grants the privileges without the GRANT OPTION to keep members of payroll operations from granting privileges to other users.

The payroll managers require different privileges and a different RACF group ID. The Spiffy security planners add a RACF group for payroll managers and name it PAYMGRS. The security planners associate the payroll managers' primary IDs with the PAYMGRS secondary ID. Next, privileges on the PAYMGR view, the compensation application, and the payroll update application are granted to PAYMGRS. The payroll update application must have the appropriate privileges on the update table.

Example: The following statement grants the SELECT, INSERT, UPDATE, and DELETE privileges on the PAYMGR view to the payroll managers' group ID PAYMGRS:

```
GRANT SELECT, INSERT, UPDATE, DELETE ON PAYMGR TO PAYMGRS;
```

Example: The following statement grants the EXECUTE privilege on the compensation application:

```
GRANT EXECUTE ON PLAN COMPENS TO PAYMGRS;
```

Auditing payroll operations and payroll management

Like the employee table, the payroll update table is created with AUDIT ALL. For both tables, the audit trace reports the number of accesses by the payroll operations and payroll management groups. The Spiffy security planners scan the reports of payroll access for large numbers or unusual patterns of access.

Securing administrator, owner, and other access

In addition to the privileges of managers, the payroll operations department, and payroll management, the security plan considers the privileges of the following roles:

- "Securing access by IDs with database administrator authority"
- "Securing access by IDs with system administrator authority"
- "Securing access by owners with implicit privileges on objects" on page 309
- "Securing access by other users" on page 310

Securing access by IDs with database administrator authority

An ID with DBADM authority over a database has many privileges on that database and its tables. These privileges include the SELECT, INSERT, DELETE, UPDATE, and ALTER statements on any table in the database, and the CREATE and DROP statements on indexes for those tables. For security reasons, the Spiffy security planners prefer not to grant all of the privileges that come with DBADM authority on DSN8D81A. DSN8D81A is the database that holds the employee table and the payroll update table.

The Spiffy security planners prefer to grant DBCTRL authority on the database because granting DBCTRL authority does not expose as many security risks as granting DBADM authority. DBCTRL authority allows an ID to support the database without allowing the ID to retrieve or change the data in the tables. However, database DSN8D81A contains several additional tables (see Appendix A, "DB2 sample tables," on page 1031). These additional tables require some of the privileges that are included in DBADM authority but not included in DBCTRL authority.

The Spiffy security planners decide to compromise between the greater security of granting DBCTRL authority and the greater flexibility of granting DBADM authority. To balance the benefits of each authority, the Spiffy security planners create an administrative ID with some, but not all of the DBADM privileges. The security plan calls for a RACF group ID with the following authorities and privileges:

- DBCTRL authority over DSN8D81A
- The INDEX privilege on all tables in the database except the employee table and the payroll update table
- The SELECT, INSERT, UPDATE, and DELETE privileges on certain tables, excluding the employee table and the payroll update table

An ID with SYSADM authority grants the privileges to the group ID.

In a similar situation, you also might consider putting the employee table and the payroll update table in a separate database. Then you can grant DBADM authority on DSN8D81A, and grant DBCTRL authority on the database that contains the employee table and the payroll update table.

Securing access by IDs with system administrator authority

An ID with SYSADM authority can access data from any table in the entire DB2 subsystem, including the employee table and the payroll update table. The Spiffy security planners want to minimize the security risk that is associated with granting SYSADM authority by granting the authority to as few users as possible. They know that the subsystem might require SYSADM authority only for certain

tasks and only for relatively short periods. They also know that the privileges that are associated with the SYSADM authority give an ID control over all of the data in a subsystem.

To limit the number of users with SYSADM authority, the Spiffy security plan grants the authority to DB2OWNER, the ID that is responsible for DB2 security. That does not mean that only IDs that are connected to DB2OWNER can exercise privileges that are associated with SYSADM authority. Instead, DB2OWNER can grant privileges to a group, connect other IDs to the group as needed, and later disconnect them.

The Spiffy security planners prefer to have multiple IDs with SYSCTRL authority instead of multiple IDs with SYSADM authority. IDs with SYSCTRL authority can exercise most of the SYSADM privileges and can assume much of the day-to-day work. IDs with SYSCTRL authority cannot access data directly or run plans unless the privileges for those actions are explicitly granted to them. However, they can run utilities, examine the output data sets, and grant privileges that allow other IDs to access data. Therefore, IDs with SYSCTRL authority can access some sensitive data, but they cannot easily access the data. As part of the Spiffy security plan, DB2OWNER grants SYSCTRL authority to selected IDs.

The Spiffy security planners also use RACF group IDs or secondary IDs to relieve the need to have SYSADM authority continuously available. SYSADM grants the necessary privileges to a RACF group or secondary ID. IDs that belong to the group can then bind plans and packages on their own behalf.

Securing access by owners with implicit privileges on objects

The Spiffy security plan must consider the ID that owns and grants privileges on the tables, views, and programs. The ID that owns these objects has many implicit privileges on the objects, including the SELECT and UPDATE privileges on the employee table. The owner of the objects can also grant privileges on the objects to other users.

The Spiffy security planners want to limit the number of IDs that have privileges on the employee table and the payroll update table to the smallest convenient value. To meet that objective, they decide that the owner of the employee table should issue all of the CREATE VIEW and GRANT statements. They also decide to have the owner of the employee table own the plans and packages that are associated with employee data. The employee table owner implicitly has the following privileges, which the plans and packages require:

- The owner of the payroll update program must have the SELECT privilege on the payroll update table and the UPDATE privilege on the employee table.
- The owner of the commission program must have the UPDATE privilege on the payroll update table and the SELECT privilege on the employee table.
- The owners of several other payroll programs must have the proper privileges to do payroll processing, such as printing payroll checks, writing summary reports, and so on.

To bind these plans and packages, an ID must have the BIND or BINDADD privileges.

The list of privileges that are required by the owner of the employee table suggests the functional approach. The Spiffy security planners create a RACF group for the owner of the employee table.

Securing access by other users

Exceptions occur when any user other than the following users accesses the employee table or the payroll table:

- Department managers
- Members of the payroll operations department
- Payroll managers
- The payroll update program

The audit report lists each exception in full. Auditors check each exception to determine whether it was a planned operation by the users with SYSADM or DBADM authority, or the employee table owner.

The audit report also lists denials of access to the tables. Those denials represent attempts by unauthorized IDs to use the tables. Some are possibly accidental; others can be attempts to violate the security system.

After running the periodic reports, the security planners archive the audit records. The archives provide a complete audit trail of access to the employee data through DB2.

Part 4. Operation and recovery

Chapter 15. Basic operation 317
Entering commands 317
DB2 operator commands 318
Where DB2 commands are entered 320
Where command responses go 322
Authorities for DB2 commands 323
Starting and stopping DB2 324
Starting DB2. 324
Messages at start 324
Options at start. 325
Restricting access to data 325
Wait state at start 325
Starting after an abend 326
Stopping DB2 326
Submitting work 327
Using DB2I (DB2 Interactive) 327
Running TSO application programs 327
Running IMS application programs 328
Running CICS application programs. 329
Running batch application programs 330
Running application programs using CAF. . . 330
Running application programs using RRSAF 331
Receiving messages 331
Receiving unsolicited DB2 messages. 332
Determining operational control 332

Chapter 16. Scheduling administrative tasks 335
Interacting with the administrative task scheduler 335
Adding a task 335
Scheduling capabilities of the administrative task scheduler 335
Defining task schedules 336
Choosing an administrative task scheduler in a data sharing environment 339
ADMIN_TASK_ADD 339
UNIX cron format 346
Listing scheduled tasks 347
Listing the last execution status of scheduled tasks 347
Removing a scheduled task. 348
ADMIN_TASK_REMOVE 349
Manually starting the administrative task scheduler. 351
Manually stopping the administrative task scheduler. 351
Synchronization between administrative task schedulers in a data sharing environment . . . 351
Troubleshooting the administrative task scheduler. 352
Enabling tracing for administrative task scheduler problem determination. 352
Recovering the administrative task scheduler task list 352
Problem executing a task 353
Problem in user-defined table functions . . 354
Problem in stored procedures 354
Architecture of the administrative task scheduler 355
The life cycle of the administrative task scheduler. 356
Scheduler task lists 356
Architecture of the administrative task scheduler in a data sharing environment . . . 356
Security guidelines for the administrative task scheduler. 357
User roles in the administrative task scheduler 357
Protection of the interface of the administrative task scheduler 357
Protection of the resources of the administrative task scheduler 358
Secure execution of tasks in the administrative task scheduler 358
Execution of scheduled tasks in the administrative task scheduler 359
Multi-threading in the administrative task scheduler. 359
Scheduled execution of a stored procedure . . 360
How the administrative task scheduler works with Unicode 360
Scheduled execution of a JCL job 361
Execution of scheduled tasks in a data sharing environment. 361

Chapter 17. Monitoring and controlling DB2 and its connections 363
Controlling DB2 databases and buffer pools . . . 363
Starting databases 364
Starting an object with a specific status. . . 364
Starting a table space or index space that has restrictions 365
Monitoring databases. 365
Obtaining information about application programs. 368
Obtaining information about pages in error 369
Stopping databases 371
Altering buffer pools 372
Monitoring buffer pools 373
Controlling user-defined functions 373
Starting user-defined functions 373
Monitoring user-defined functions 374
Stopping user-defined functions 374
Controlling DB2 utilities. 375
Starting online utilities 375
Monitoring online utilities 375
Stand-alone utilities 376
Controlling the IRLM. 376
Starting the IRLM 377
Modifying the IRLM 377
Monitoring the IRLM connection 378
Stopping the IRLM 378
Monitoring threads 379
Controlling TSO connections 380
Connecting to DB2 from TSO 381

© Copyright IBM Corp. 1982, 2008

Monitoring TSO and CAF connections 381
Disconnecting from DB2 while under TSO. . . . 383
Controlling CICS connections 383
Connecting from CICS 384
Restarting CICS 384
Displaying indoubt units of recovery . . . 385
Recovering indoubt units of recovery manually 385
Displaying postponed units of recovery . . 385
Controlling CICS connections 385
Defining CICS threads 386
Monitoring the threads 386
Disconnecting applications 386
Disconnecting from CICS 387
Orderly termination 387
Forced termination 387
Controlling IMS connections 388
Connecting to the IMS control region 388
Thread attachment 389
Thread termination 390
Displaying indoubt units of recovery . . . 390
Recovering indoubt units of recovery . . . 391
Displaying postponed units of recovery . . 391
Duplicate correlation IDs 392
Resolving residual recovery entries 392
Controlling IMS dependent region connections 393
Connecting from dependent regions 393
Monitoring the activity on connections . . . 394
Disconnecting from dependent regions . . . 396
Disconnecting from IMS 396
Controlling RRS connections 397
Connecting to RRS using RRSAF 397
Restarting DB2 and RRS. 398
Displaying indoubt units of recovery . . . 398
Recovering indoubt units of recovery manually 398
Displaying postponed units of recovery . . 399
Monitoring RRSAF connections 399
Displaying RRSAF connections 399
Disconnecting RRSAF applications from DB2 400
Controlling connections to remote systems . . . 400
Starting the DDF 401
Suspending and resuming DDF server activity 401
Monitoring connections to other systems . . . 402
The command DISPLAY DDF 402
The command DISPLAY LOCATION . . . 403
The command DISPLAY THREAD 405
Canceling dynamic SQL from a client application 410
The command CANCEL THREAD 411
Using VTAM commands to cancel threads 412
Monitoring and controlling stored procedures 413
Displaying information about stored procedures and their environment 413
Refreshing the environment for stored procedures or user-defined functions . . . 415
Obtaining diagnostic information about stored procedures 415
Using NetView to monitor errors 416
Stopping the DDF 417
Controlling traces 418
Controlling the DB2 trace 419
Diagnostic traces for attachment facilities . . . 420
Diagnostic trace for the IRLM 420
Controlling the resource limit facility (governor) 421
Changing subsystem parameter values 421

Chapter 18. Managing the log and the bootstrap data set 423
How database changes are made 423
Units of recovery 423
Rolling back work 424
Establishing the logging environment 425
Creation of log records 425
Retrieval of log records 425
Writing the active log. 425
Writing the archive log (offloading) 426
Triggering offload 426
The offloading process 427
Archive log data sets 428
Controlling the log 430
Archiving the log 430
Dynamically changing the checkpoint frequency 433
Monitoring the system checkpoint 433
Setting limits for archive log tape units 433
Displaying log information 434
Resetting the log RBA 434
Log RBA range 434
Resetting the log RBA value in a data sharing
environment. 435
Resetting the log RBA value in a non-data
sharing environment 435
Managing the bootstrap data set (BSDS) 436
BSDS copies with archive log data sets 437
Changing the BSDS log inventory 438
Discarding archive log records. 438
Deleting archive logs automatically 438
Locating archive log data sets 439

Chapter 19. Restarting DB2 after termination 443
Termination 443
Normal termination 443
Abends 444
Normal restart and recovery 444
Phase 1: Log initialization 445
Phase 2: Current status rebuild 446
Phase 3: Forward log recovery. 447
Phase 4: Backward log recovery 448
Restarting automatically 449
Deferring restart processing 450
Restarting with conditions 451
Resolving postponed units of recovery 452
Errors encountered during RECOVER POSTPONED processing 452
Output from RECOVER POSTPONED processing 453
Choosing recovery operations for conditional restart 453
Conditional restart records 454

Chapter 20. Maintaining consistency across multiple systems 455

Multiple system consistency 455
The two-phase commit process 455
Illustration of two-phase commit 456
Maintaining consistency after termination or failure 457
Termination for multiple systems 458
Normal restart and recovery for multiple systems 458
Phase 1: Log initialization 458
Phase 2: Current status rebuild 458
Phase 3: Forward log recovery. 458
Phase 4: Backward log recovery 459
Restarting multiple systems with conditions . . 459
Resolving indoubt units of recovery 459
Resolution of IMS indoubt units of recovery . . 460
Resolution of CICS indoubt units of recovery 461
Resolution of WebSphere Application Server indoubt units of recovery 461
Resolution of remote DBMS indoubt units of recovery 463
Making heuristic decisions 464
Methods for determining the coordinator's commit or abort decision 464
Displaying information about indoubt threads 464
Recovering indoubt threads 465
Resetting the status of an indoubt thread . . 465
Resolution of RRS indoubt units of recovery . . 466
Consistency across more than two systems . . . 467
Commit coordinator and multiple participants 467
Illustration of multi-site update 468

Chapter 21. Backing up and recovering databases 471
Planning for backup and recovery 471
Considerations for recovering distributed data 472
Extended recovery facility (XRF) toleration . . 472
Considerations for recovering indexes 473
Preparing for recovery 473
Events that occur during recovery 474
Complete recovery cycles 475
A recovery cycle example 476
How DFSMShsm affects your recovery environment. 477
Maximizing data availability during backup and recovery 477
How to find recovery information 480
Where recovery information resides 480
Reporting recovery information 480
Preparing to recover to a prior point of consistency 481
Step 1: Resetting exception status. 481
Step 2: Copying the data 481
Step 3: Establishing a point of consistency 482
Preparing to recover the entire DB2 subsystem to a prior point in time 482
Preparing for disaster recovery 483
System-wide points of consistency 484
Essential disaster recovery elements 484
Ensuring more effective recovery from inconsistency 486
Actions to take 486
Actions to avoid 487
Running the RECOVER utility in parallel . . . 488
Using fast log apply during RECOVER. . . . 489
Reading the log without RECOVER 489
Copying page sets and data sets 489
Backing up with DFSMS 491
Backing up with RVA storage control or Enterprise Storage Server 491
Recovering page sets and data sets 491
Recovering the work file database 493
Problems with the user-defined work file data set 493
Problems with DB2-managed work file data sets 493
Recovering error ranges for a work file table space 493
Recovering the catalog and directory 494
Recovering data to a prior point of consistency . . 495
Considerations for recovering to a prior point of consistency 495
Recovering table spaces 495
Recovering tables that contain identity columns 496
Recovering indexes 497
Recovering catalog and directory tables . . 498
Using RECOVER to restore data to a previous point in time 500
Using the TOCOPY, TOLASTCOPY, and TOLASTFULLCOPY options 500
Using the TOLOGPOINT option 501
Planning for point in time recovery 501
Ensuring consistency 502
Restoring data by using DSN1COPY . . . 504
Backing up and restoring data with non-DB2 dump and restore 504
Recovery of dropped objects 505
Avoiding accidentally dropping objects. . . 505
Procedures for recovery 505
Recovery of an accidentally dropped table 505
Recovery of an accidentally dropped table space 507
Discarding SYSCOPY and SYSLGRNX records 510
System-level point in time recovery 511
System-level backup and restore 511
Recovery to an arbitrary point in time . . . 513
Recovery to the point in time of a backup 513
Remote site recovery from a disaster at a local site 514

Chapter 22. Recovery scenarios 517
IRLM failure recovery 517
z/OS or power failure recovery 518
Disk failure recovery 518
Application error recovery 520
IMS-related failure recovery 521
IMS control region (CTL) failure recovery . . . 522
Resolution of indoubt units of recovery. . . . 522
Problem 1 522
Problem 2 523
IMS application failure recovery 524
Problem 1 524

Problem 2 524
CICS-related failure recovery 525
CICS application failure recovery 525
Recovery when CICS is not operational . . . 525
Recovery when CICS cannot connect to DB2 526
Manually recovering CICS indoubt units of recovery 527
CICS attachment facility failure recovery . . . 530
Subsystem termination recovery 530
Resource failure recovery 531
Active log failure recovery 531
Problem 1 - Out of space in active logs . . . 532
Problem 2 - Write I/O error on active log data set 533
Problem 3 - Dual logging is lost 533
Problem 4 - I/O errors while reading the active log. 533
Archive log failure recovery 535
Problem 1 - Allocation problems 535
Problem 2 - Write I/O errors during archive log offload 536
Problem 3 - Read I/O errors on archive data set during recover 536
Problem 4 - Insufficient disk space for offload processing 536
Temporary resource failure recovery 537
BSDS failure recovery 538
Problem 1 - An I/O error occurs 538
Problem 2 - An error occurs while opening 538
Problem 3 - Unequal timestamps exist . . . 539
Recovering the BSDS from a backup copy . . . 539
DB2 database failures. 542
Recovery from down-level page sets. 543
Procedure for recovering invalid LOBs 544
Table space input/output error recovery 545
DB2 catalog or directory input/output errors . . . 546
Integrated catalog facility failure recovery 547
Recovery when the VSAM volume data set (VVDS) is destroyed 547
Out of disk space or extent limit recovery . . . 548
Violation of referential constraint recovery. . . . 552
Distributed data facility failure recovery 553
Conversation failure recovery 553
Communications database failure recovery . . 554
Problem 1 554
Problem 2 554
Database access thread failure recovery. . . . 555
VTAM failure recovery 555
TCP/IP failure recovery 556
Remote logical unit failure recovery 556
Indefinite wait condition recovery 557
Security failure recovery for database access threads 557
Remote site recovery from a disaster at the local site 558
Restoring data from image copies and archive logs 558
Using a tracker site for disaster recovery . . . 568
Characteristics of a tracker site 568
Setting up a tracker site 569
Using RESTORE SYSTEM LOGONLY to establish a recovery cycle 569
Using the RECOVER utility to establish a recovery cycle 571
Media failures during LOGONLY recovery 574
Maintaining the tracker site 574
Making the tracker site the takeover site . . 574
Using data mirroring 576
The rolling disaster 576
Consistency groups 578
Recovering in a data mirroring environment 579
Recovering with Extended Remote Copy . . 581
Resolving indoubt threads 582
Description of the recovery environment . . . 582
Configuration 582
Applications. 582
Threads 582
Communication failure recovery 583
Making a heuristic decision 584
Recovery from an IMS outage that results in an IMS cold start 585
Recovery from a DB2 outage at a requester that results in a DB2 cold start 586
Recovery from a DB2 outage at a server that results in a DB2 cold start 589
Correcting a heuristic decision. 589

Chapter 23. Recovery from BSDS or log failure during restart 591
Log initialization or current status rebuild failure recovery 593
Description of failure during log initialization 594
Description of failure during current status rebuild 595
Restart by truncating the log 595
Step 1: Find the log RBA after the inaccessible part of the log 595
Step 2: Identify lost work and inconsistent data 598
Step 3: Determine what status information has been lost 601
Step 4: Truncate the log at the point of error 601
Step 5: Start DB2 602
Step 6: Resolve data inconsistency problems 602
Failure during forward log recovery. 602
Understanding forward log recovery failure . . 603
Starting DB2 by limiting restart processing . . 603
Step 1: Find the log RBA after the inaccessible part of the log 604
Step 2: Identify incomplete units of recovery and inconsistent page sets 606
Step 3: Restrict restart processing to the part of the log after the damage. 607
Step 4: Start DB2 607
Step 5: Resolve inconsistent data problems 607
Failure during backward log recovery 607
Understanding backward log recovery failure 608
Bypassing backout before restarting 608
Failure during a log RBA read request 609
Unresolvable BSDS or log data set problem during restart 610

Preparing for recovery or restart 611
Performing fall back to a prior shutdown point 611
Failure resulting from total or excessive loss of log data 613
Total loss of the log 613
Excessive loss of data in the active log 614
Resolving inconsistencies resulting from a conditional restart 616
Inconsistencies in a distributed environment . . 616
Procedures for resolving inconsistencies . . . 616
Method 1. Recover to a prior point of consistency 617
Method 2. Re-create the table space 618
Method 3. Use the REPAIR utility on the data 618

Chapter 15. Basic operation

The information under this heading, up to "Running IMS application programs" on page 328, is General-use Programming Interface and Associated Guidance Information, as defined in "Notices" on page 1301.

The simplest elements of operation for DB2 UDB for z/OS are described in this chapter; they include:

"Entering commands"
"Starting and stopping DB2" on page 324
"Submitting work" on page 327
"Receiving messages" on page 331

Normal operation also requires more complex tasks. They are described in:

- Chapter 17, "Monitoring and controlling DB2 and its connections," on page 363, which considers the control of connections to IRLM, to TSO, to IMS, and to CICS, as well as connections to other database management systems.
- Chapter 18, "Managing the log and the bootstrap data set," on page 423, which describes the roles of the log and the bootstrap data set in preparing for restart and recovery.
- Chapter 19, "Restarting DB2 after termination," on page 443, which tells what happens when DB2 terminates normally or abnormally and how to restart it while maintaining data integrity.
- Chapter 20, "Maintaining consistency across multiple systems," on page 455, which explains the two-phase commit process and the resolution of indoubt units of recovery.
- Chapter 21, "Backing up and recovering databases," on page 471, which explains how to prepare for recovery as well as how to recover.

Recovery after various types of failure is described in:

- Chapter 22, "Recovery scenarios," on page 517
- Chapter 23, "Recovery from BSDS or log failure during restart," on page 591

Operating a data sharing group: Although many of the commands and operational procedures described here are the same in a data sharing environment, some special considerations are described in Chapter 5 of *DB2 Data Sharing: Planning and Administration*. Consider the following issues when you are operating a data sharing group

- New commands used for data sharing, and the concept of command scope
- Logging and recovery operations
- Restart after an abnormal termination
- Disaster recovery procedures
- Recovery procedures for coupling facility resources

Entering commands

You can control most of the operational environment by using DB2 commands. You might need to use other types of commands, including:

- IMS commands that control IMS connections
- CICS commands that control CICS connections

© Copyright IBM Corp. 1982, 2008

- IMS and CICS commands that allow you to start and stop connections to DB2 and display activity on the connections
- z/OS commands that allow you to start, stop, and change the internal resource lock manager (IRLM)

Using these commands is described in Chapter 17, "Monitoring and controlling DB2 and its connections," on page 363. For a full description of the commands available, see Chapter 2 of *DB2 Command Reference*.

DB2 operator commands

The DB2 commands, as well as their functions, are:

ALTER BUFFERPOOL
Sets or alters buffer pool size while DB2 is online.

ALTER GROUPBUFFERPOOL
Alters attributes of group buffer pools, which are used in a data sharing environment.

ALTER UTILITY
Alters the parameter values of an active REORG utility job.

ARCHIVE LOG
Archives (offloads) the current active log.

CANCEL THREAD
Cancels processing for specific local or distributed threads. It can be used for parallel task threads.

DISPLAY ARCHIVE
Displays information about the specifications for archive parameters, status of allocated dedicated tape units, volume and data set names associated with all active tape units, and correlation ID of the requester.

DISPLAY BUFFERPOOL
Displays buffer pool information while DB2 is online.

DISPLAY DATABASE
Displays the status of a database.

DISPLAY DDF
Displays information about the status and configuration of the distributed data facility (DDF), and about the connections or threads controlled by DDF.

DISPLAY FUNCTION SPECIFIC
Displays the statistics about external user-defined functions accessed by DB2 applications.

DISPLAY GROUP
Displays information about the data sharing group to which a DB2 subsystem belongs.

DISPLAY GROUPBUFFERPOOL
Displays status and statistical information about DB2 group buffer pools, which are used in a data sharing environment.

DISPLAY LOCATION
Displays statistics about threads and conversations between remote DB2 subsystem and the local subsystem.

DISPLAY LOG
Displays the current checkpoint frequency (CHKFREQ) value, information about the current active log data sets, and the status of the offload task.

DISPLAY PROCEDURE
Displays statistics about stored procedures accessed by DB2 applications.

DISPLAY RLIMIT
Displays the status of the resource limit facility (governor).

DISPLAY THREAD
Displays information about DB2, distributed subsystem connections, and parallel tasks.

DISPLAY TRACE
Displays the status of DB2 traces.

DISPLAY UTILITY
Displays the status of a utility.

MODIFY TRACE
Changes the trace events (IFCIDs) being traced for a specified active trace.

RECOVER BSDS
Reestablishes dual bootstrap data sets.

RECOVER INDOUBT
Recovers threads left indoubt after DB2 is restarted.

RECOVER POSTPONED
Completes backout processing for units of recovery (URs) whose backout was postponed during an earlier restart, or cancels backout processing of the postponed URs if the CANCEL option is used.

RESET INDOUBT
Purges DB2 information about indoubt threads.

SET ARCHIVE
Controls or sets the limits for the allocation and the deallocation time of the tape units for archive log processing.

SET LOG
Modifies the checkpoint frequency (CHKFREQ) value dynamically without changing the value in the subsystem parameter load module.

SET SYSPARM
Loads the subsystem parameter module specified in the command.

START DATABASE
Starts a list of databases or table spaces and index spaces.

START DB2
Initializes the DB2 subsystem.

START DDF
Starts the distributed data facility.

START FUNCTION SPECIFIC
Activates an external function that is stopped.

START PROCEDURE
Starts a stored procedure that is stopped.

START RLIMIT
Starts the resource limit facility (governor).

START TRACE
Starts DB2 traces.

STOP DATABASE
Stops a list of databases or table spaces and index spaces.

STOP DB2
Stops the DB2 subsystem.

STOP DDF
Stops or suspends the distributed data facility.

STOP FUNCTION SPECIFIC
Prevents DB2 from accepting SQL statements with invocations of the specified functions.

STOP PROCEDURE
Prevents DB2 from accepting SQL CALL statements for a stored procedure.

STOP RLIMIT
Stops the resource limit facility (governor).

STOP TRACE
Stops traces.

TERM UTILITY
Terminates execution of a utility.

Where DB2 commands are entered

You can enter commands from the following sources:
- A z/OS console or z/OS application program
- An IMS terminal or program
- A CICS terminal
- A TSO terminal
- An APF-authorized program
- An IFI application program

From a z/OS console or z/OS application program: You can enter all DB2 commands from a z/OS console or a z/OS application program. The START DB2 command must be issued from a z/OS console (or from an APF-authorized program, such as SDSF, that passes the START DB2 to the z/OS console). The command group authorization level must be SYS.

More than one DB2 subsystem can run under z/OS. You prefix a DB2 command with special characters that identify which subsystem to direct the command to. The 1- to 8-character prefix is called the *command prefix*. Specify the command prefix on installation panel DSNTIPM. The default character for the command prefix is -DSN1. Examples in this book use the hyphen (-) for the command prefix.

From an IMS terminal or program: You can enter all DB2 commands except START DB2 from either an IMS terminal or program. The terminal or program must be authorized to enter the /SSR command.

An IMS subsystem can attach to more than one DB2 subsystem, so you must prefix a command that is directed from IMS to DB2 with a special character that tells which subsystem to direct the command to. That character is called the *command recognition character* (CRC); specify it when you define DB2 to IMS, in the subsystem member entry in IMS.PROCLIB. (For details, see Part 2 of *DB2 Installation Guide*.)

Recommendation: Use the same character for the CRC and the command prefix for a single DB2 subsystem. You need to use a command prefix of one character or you cannot match these identifiers.

The examples in this book assume that both the command prefix and the CRC are the hyphen (-) . However, if you can attach to more than one DB2 subsystem, you must issue your commands using the appropriate CRC. In the following example, the CRC is a question mark character:

You enter:

```
/SSR ?DISPLAY THREAD
```

DB2 returns the following messages:

```
DFS058  SSR COMMAND COMPLETED
DSNV401I ? DISPLAY THREAD REPORT FOLLOWS -
DSNV402I ? ACTIVE THREADS -
⋮
```

From a CICS terminal: You can enter all DB2 commands except START DB2 from a CICS terminal authorized to enter the DSNC transaction code.

For example, you enter:

```
DSNC -DISPLAY THREAD
```

DB2 returns the following messages:

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV402I - ACTIVE THREADS -
⋮
```

CICS can attach to only one DB2 subsystem at a time; therefore CICS does not use the DB2 command prefix. Instead, each command entered through the CICS attachment facility must be preceded by a hyphen (-), as in previous the example. The CICS attachment facility routes the commands to the connected DB2 subsystem and obtains the command responses.

From a TSO terminal: You can enter all DB2 commands except START DB2 from a DSN session.

Example: The system displays:

```
READY
```

You enter:

```
DSN SYSTEM (subsystem-name)
```

The system displays:

```
DSN
```

You enter:

```
-DISPLAY THREAD
```

DB2 returns the following messages:

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV402I - ACTIVE THREADS -
⋮
```

A TSO session can attach to only one DB2 subsystem at a time; therefore TSO does not use the DB2 command prefix. Instead, each command that is entered through the TSO attachment facility must be preceded by a hyphen (-), as the preceding example demonstrates. The TSO attachment facility routes the command to DB2 and obtains the command response.

All DB2 commands except START DB2 can also be entered from a DB2I panel using option 7, *DB2 Commands*. For more information about using DB2I, see "Using DB2I (DB2 Interactive)" on page 327.

From an APF-authorized program: As with IMS, DB2 commands (including START
DB2) can be passed from an APF-authorized program to multiple DB2 subsystems
by the MGCRE (SVC 34) z/OS service. Thus, the value of the command prefix
identifies the particular subsystem to which the command is directed. The
subsystem command prefix is specified, as in IMS, when DB2 is installed (in the
SYS1.PARMLIB member IEFSSNxx). DB2 supports the z/OS WTO Command And
Response Token (CART) to route individual DB2 command response messages
back to the invoking application program. Use of the CART token is necessary if
multiple DB2 commands are issued from a single application program.

For example, to issue DISPLAY THREAD to the default DB2 subsystem from an APF-authorized program run as a batch job, code:

```
MODESUPV DS    0H
         MODESET MODE=SUP,KEY=ZERO
SVC34    SR    0,0
         MGCRE  CMDPARM
         EJECT
CMDPARM  DS    0F
CMDFLG1  DC    X'00'
CMDLENG  DC    AL1(CMDEND-CMDPARM)
CMDFLG2  DC    X'0000'
CMDDATA  DC    C'-DISPLAY THREAD'
CMDEND   DS    0C
```

and DB2 returns the following messages:

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV402I - ACTIVE THREADS -
⋮
DSN9022I - DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
```

From an IFI application program: An application program can issue DB2 commands using the instrumentation facility interface (IFI). The IFI application program protocols are available through the IMS, CICS, TSO, and call attachment facility (CAF) attaches, and the Resource Recovery Services attachment facility. For an example in which the DB2 START TRACE command for monitor class 1 is issued, see "COMMAND: Syntax and usage with IFI" on page 1154.

Where command responses go

In most cases, DB2 command responses are returned to the entering terminal or, for batch jobs, appear in the printed listing.

In CICS, you can direct command responses to another terminal. Name the other terminal as the destination (*dest*) in this command:

```
DSNC dest -START DATABASE
```

If a DB2 command is entered from an IMS or CICS terminal, the response messages can be directed to different terminals. If the response includes more than one message, the following cases are possible:

- If the messages are issued in a set, the entire set of messages is sent to the IMS or CICS terminal that entered the command. For example, DISPLAY THREAD issues a set of messages.
- If the messages are issued one after another, and not in a set, only the first message is sent to the terminal that entered the command. Later messages are routed to one or more z/OS consoles via the WTO function. For example, START DATABASE issues several messages one after another.

  You can choose alternate consoles to receive the subsequent messages by assigning them the routing codes placed in the DSNZP*xxx* module when DB2 is installed. If you want to have all of the messages available to the person who sent the command, route the output to a console near the IMS or CICS master terminal.

For APF-authorized programs that run in batch jobs, command responses are returned to the master console and to the system log if hard copy logging is available. Hard copy logging is controlled by the z/OS system command VARY. See *z/OS MVS System Commands* for more information.

Authorities for DB2 commands

The ability to issue DB2 commands, such as STOP DB2, and to use most other DB2 functions, requires the appropriate privilege or authority. Privileges and authorities can be granted to authorization IDs in many combinations and can also be revoked.

The individual authorities are listed in Figure 9 on page 139. Each administrative authority has the individual authorities shown in its box, and the individual authorities for all the levels beneath it. For example, DBADM has ALTER, DELETE, INDEX, INSERT, SELECT, and UPDATE authorities as well as those listed for DBCTRL and DBMAINT.

Any user with the STOPALL privilege can issue the STOP DB2 command. Besides those who have been granted STOPALL explicitly, the privilege belongs implicitly to anyone with SYSOPR authority or higher. When installing DB2, you can choose:

- One or two authorization IDs with installation SYSADM authority
- Zero, one, or two authorization IDs with installation SYSOPR authority

The IDs with those authorizations are contained in the load module for subsystem parameters (DSNZP*xxx*).

The START DB2 command can be entered only at a z/OS console authorized to enter z/OS system commands. The command group authorization level must be SYS.

DB2 commands entered from a logged-on z/OS console can be authorized using secondary authorization IDs. The authorization ID associated with a z/OS console is SYSOPR, which carries the authority to issue all DB2 commands except:

- RECOVER BSDS
- START DATABASE
- STOP DATABASE
- ARCHIVE LOG

APF-authorized programs that issue commands through MGCRE (SVC 34) have SYSOPR authority unless DB2 can determine the program's RACF user ID. In that case, DB2 will use that user ID for authorization. To avoid errors, the user should obtain SYSOPR authority for those DB2 instances.

The authority to start or stop any particular database must be specifically granted to an ID with SYSOPR authority. Likewise, an ID with SYSOPR authority must be granted specific authority to issue the RECOVER BSDS and ARCHIVE LOG commands.

The SQL GRANT statement can be used to grant SYSOPR authority to other user IDs such as the /SIGN user ID or the LTERM of the IMS master terminal.

For information about other DB2 authorization levels, see "Establishing RACF protection for DB2" on page 264. *DB2 Command Reference* also has authorization level information for specific commands.

Starting and stopping DB2

Starting and stopping DB2 is a simple process, and one that you will probably not have to do often. Before DB2 is stopped, the system takes a shutdown checkpoint. This checkpoint and the recovery log give DB2 the information it needs to restart.

This section describes the START DB2 and STOP DB2 commands, explains how you can limit access to data at startup, and contains a brief overview of startup after an abend.

Starting DB2

When installed, DB2 is defined as a formal z/OS subsystem. Afterward, the following message appears during any IPL of z/OS:

```
DSN3100I - DSN3UR00 - SUBSYSTEM ssnm READY FOR -START COMMAND
```

where *ssnm* is the DB2 subsystem name. At that point, you can start DB2 from a z/OS console that has been authorized to issue system control commands (z/OS command group SYS), by entering the command START DB2. The command must be entered from the authorized console and not submitted through JES or TSO.

It is *not* possible to start DB2 by a JES batch job or a z/OS START command. The attempt is likely to start an address space for DB2 that then abends, probably with reason code X'00E8000F'.

You can also start DB2 from an APF-authorized program by passing a START DB2 command to the MGCRE (SVC 34) z/OS service.

Messages at start

The system responds with some or all of the following messages depending on which parameters you chose:

```
$HASP373 xxxxMSTR STARTED
DSNZ002I - SUBSYS ssnm SYSTEM PARAMETERS
           LOAD MODULE NAME IS dsnzparm-name
DSNY001I - SUBSYSTEM STARTING
DSNJ127I - SYSTEM TIMESTAMP FOR BSDS=87.267 14:24:30.6
DSNJ001I - csect CURRENT COPY n ACTIVE LOG DATA
           SET IS DSNAME=...,
           STARTRBA=...,ENDRBA=...
DSNJ099I - LOG RECORDING TO COMMENCE WITH
           STARTRBA = xxxxxxxxxxxx
```

```
$HASP373 xxxxDBM1 STARTED
DSNR001I - RESTART INITIATED
DSNR003I - RESTART...PRIOR CHECKPOINT RBA=xxxxxxxxxxxx
DSNR004I - RESTART...UR STATUS COUNTS...
           IN COMMIT=nnnn, INDOUBT=nnnn, INFLIGHT=nnnn,
           IN ABORT=nnnn, POSTPONED ABORT=nnnn
DSNR005I - RESTART...COUNTS AFTER FORWARD RECOVERY
           IN COMMIT=nnnn, INDOUBT=nnnn
DSNR006I - RESTART...COUNTS AFTER BACKWARD RECOVERY
           INFLIGHT=nnnn, IN ABORT=nnnn, POSTPONED ABORT=nnnn
DSNR002I - RESTART COMPLETED
DSN9002I -DSNYASCP 'START DB2' NORMAL COMPLETION
DSNV434I - DSNVRP NO POSTPONED ABORT THREADS FOUND
DSN9022I - DSNVRP 'RECOVER POSTPONED' NORMAL COMPLETION
```

If any of the *nnnn* values in message DSNR004I are not zero, message DSNR007I is issued to provide the restart status table.

The START DB2 command starts the system services address space, the database services address space, and, depending upon specifications in the load module for subsystem parameters (DSNZPARM by default), the distributed data facility address space and the DB2-established stored procedures address space. Optionally, another address space, the internal resource lock manager (IRLM), can be started automatically.

Options at start

Starting invokes the load module for subsystem parameters. This load module contains information specified when DB2 was installed. For example, the module contains the name of the IRLM to connect to. In addition, it indicates whether the distributed data facility (DDF) is available and, if it is, whether it should be automatically started when DB2 is started. For information about using a command to start DDF, see "Starting the DDF" on page 401. You can specify PARM (*module-name*) on the START DB2 command to provide a parameter module other than the one specified at installation.

There is a conditional restart operation, but there are no parameters to indicate normal or conditional restart on the START DB2 command. For information about conditional restart, see "Restarting with conditions" on page 451.

Restricting access to data

You can restrict access to data with another option of the START DB2 command. Use:

ACCESS(MAINT)
: To limit access to users who have installation SYSADM or installation SYSOPR authority.

  Users with those authorities can do maintenance operations such as recovering a database or taking image copies. To restore access to all users, stop DB2 and then restart it. Either omit the ACCESS keyword or use:

ACCESS(*)
: To allow all authorized users to connect to DB2.

Wait state at start

If a JCL error, such as device allocation or region size, occurs while trying to start the database services address space, DB2 goes into wait status. To end the wait, cancel the system services address space and the distributed data facility address

space from the console. After DB2 stops, check the start procedures of all three DB2 address spaces for correct JCL syntax. See *Data Sharing: Planning and Administration* for more information.

To accomplish the check, compare the expanded JCL in the SYSOUT output against the correct JCL provided in *z/OS MVS JCL User's Guide* or *z/OS MVS JCL Reference*. Then, take the member name of the erroneous JCL procedure also provided in the SYSOUT to the system programmer who maintains your procedure libraries. After finding out which proclib contains the JCL in question, locate the procedure and correct it.

Starting after an abend

Starting DB2 after it abends is different from starting it after the command STOP DB2 has been issued. After STOP DB2, the system finishes its work in an orderly way and takes a shutdown checkpoint before stopping. When DB2 is restarted, it uses information from the system checkpoint and recovery log to determine the system status at shutdown.

When a power failure occurs, DB2 abends without being able to finish its work or take a shutdown checkpoint. When DB2 is restarted after an abend, it refreshes its knowledge of its status at termination using information on the recovery log and notifies the operator of the status of various units of recovery.

You can indicate that you want DB2 to postpone some of the backout work traditionally performed during system restart. You can delay the backout of long running units of recovery (URs) using installation options LIMIT BACKOUT and BACKOUT DURATION on panel DSNTIPL. For a description of these installation parameters, see Chapter 2 of *DB2 Installation Guide*.

Normally, the restart process resolves all inconsistent states. In some cases, you have to take specific steps to resolve inconsistencies. There are steps you can take to prepare for those actions. For example, you can limit the list of table spaces that are recovered automatically when DB2 is started. For an explanation of the causes of database inconsistencies and how you can prepare to recover from them, see Chapter 19, "Restarting DB2 after termination," on page 443.

Stopping DB2

Before stopping, all DB2-related write to operator with reply (WTOR) messages must receive replies. Then one of the following commands terminates the subsystem:

```
-STOP DB2 MODE(QUIESCE)
-STOP DB2 MODE(FORCE)
```

For the effects of the QUIESCE and FORCE options, see "Normal termination" on page 443. In a data sharing environment, see *Data Sharing: Planning and Administration*.

The following messages are returned:

```
DSNY002I - SUBSYSTEM STOPPING
DSN9022I - DSNYASCP '-STOP DB2' NORMAL COMPLETION
DSN3104I - DSN3EC00 - TERMINATION COMPLETE
```

Before DB2 can be restarted, the following message must also be returned to the z/OS console that is authorized to enter the START DB2 command:

```
DSN3100I - DSN3EC00 - SUBSYSTEM ssnm READY FOR -START COMMAND
```

If the STOP DB2 command is not issued from a z/OS console, messages DSNY002I and DSN9022I are not sent to the IMS or CICS master terminal operator. They are routed only to the z/OS console that issued the START DB2 command.

Submitting work

An application program running under TSO, IMS, or CICS can make use of DB2 resources by executing embedded SQL statements. How to run application programs from those environments is explained under:

"Running TSO application programs" on page 327
"Running IMS application programs" on page 328
"Running CICS application programs" on page 329
"Running batch application programs" on page 330
"Running application programs using CAF" on page 330
"Running application programs using RRSAF" on page 331

In each case, there are some conditions that the application program must meet to embed SQL statements and to authorize the use of DB2 resources and data.

All application programming defaults, including the subsystem name that the programming attachments discussed here use, are in the DSNHDECP load module. Make sure your JCL specifies the proper set of program libraries.

Using DB2I (DB2 Interactive)

Using the interactive program DB2I, you can run application programs and perform many DB2 operations by entering values on panels. DB2I runs under TSO using ISPF (Interactive System Productivity Facility) services. To use it, follow your local procedures for logging on to TSO, and enter ISPF. The DB2I menu is shown in Part 1 of *DB2 Application Programming and SQL Guide*.

You control each operation by entering the parameters that describe it on the panels provided. DB2 also provides help panels to:

- Explain how to use each operation
- Provide the syntax for and examples of DSN subcommands, DB2 operator commands, and DB2 utility control statements

To access the help panels, press the HELP PF key. (The key can be set locally, but typically is PF1.)

Running TSO application programs

To run TSO application programs:

1. Log on.
2. Enter the DSN command.
3. Respond to the prompt by entering the RUN subcommand.

The following example runs application program DSN8BC3. The program is in library *prefix*.RUNLIB.LOAD, the name assigned to the load module library.

```
DSN SYSTEM (subsystem-name)
RUN PROGRAM (DSN8BC3) PLAN(DSN8BH81) LIB ('prefix.RUNLIB.LOAD')
END
```

A TSO application program that you run in a DSN session must be link-edited with the TSO language interface program (DSNELI). The program cannot include IMS DL/I calls because that requires the IMS language interface module (DFSLI000).

The terminal monitor program (TMP) attaches the DB2-supplied DSN command processor, which in turn attaches the application program.

The DSN command starts a DSN session, which in turn provides a variety of subcommands and other functions. The DSN subcommands are:

ABEND
Causes the DSN session to terminate with a DB2 X'04E' abend completion code and with a DB2 abend reason code of X'00C50101'.

BIND PACKAGE
Generates an application package.

BIND PLAN
Generates an application plan.

DCLGEN
Produces SQL and host language declarations.

END Ends the DB2 connection and returns to TSO.

FREE PACKAGE
Deletes a specific version of a package.

FREE PLAN
Deletes an application plan.

REBIND PACKAGE
Regenerates an existing package.

REBIND PLAN
Regenerates an existing plan.

RUN Executes a user application program.

SPUFI Invokes a DB2I facility for executing SQL statements not embedded in an application program.

You can also issue the following DB2 and TSO commands from a DSN session:
- Any TSO command except TIME, TEST, FREE, and RUN.
- Any DB2 command except START DB2. For a list of those commands, see "DB2 operator commands" on page 318.

DB2 uses the following sources to find an authorization for access by the application program. DB2 checks the first source listed; if it is unavailable, it checks the second source, and so on.
1. RACF USER parameter supplied at logon
2. TSO logon user ID
3. Site-chosen default authorization ID
4. IBM-supplied default authorization ID

Either the RACF USER parameter or the TSO user ID can be modified by a locally defined authorization exit routine.

Running IMS application programs

To run IMS application programs, you can either enter transactions from an IMS terminal or you can invoke IMS transaction and commands by using the DB2-supplied stored procedure DSNAIMS. For more information about DSNAIMS, see Appendix J, "DB2-supplied stored procedures," on page 1255.

Application programs that contain SQL statements run in message processing program (MPP), batch message processing (BMP), Fast Path regions, or IMS batch regions.

The program must be link-edited with the IMS language interface module (DFSLI000). It can write to and read from other database management systems using the distributed data facility, in addition to accessing DL/I and Fast Path resources.

DB2 checks whether the authorization ID provided by IMS is valid. For message-driven regions, IMS uses the SIGNON-ID or LTERM as the authorization ID. For non-message-driven regions and batch regions, IMS uses the ASXBUSER field (if RACF or another security package is active). The ASXBUSER field is defined by z/OS as 7 characters. If the ASXBUSER field contains binary zeros or blanks (RACF or another security package is not active), IMS uses the PSB name instead. See Chapter 11, "Controlling access to a DB2 subsystem," on page 231 for more information about DB2 authorization IDs.

An IMS terminal operator probably notices few differences between application programs that access DB2 data and programs that access DL/I data because no messages relating to DB2 are sent to a terminal operator by IMS. However, your program can signal DB2 error conditions with a message of your choice. For example, at the program's first SQL statement, it receives an SQL error code if the resources to run the program are not available or if the operator is not authorized to use the resources. The program can interpret the code and issue an appropriate message to the operator.

Running IMS batch work: You can run batch DL/I jobs to access DB2 resources; DB2-DL/I batch support uses the IMS attach package.

See Part 5 of *DB2 Application Programming and SQL Guide* for more information about application programs and DL/I batch. See *IMS Application Programming: Design Guide* for more information about recovery and DL/I batch.

Running CICS application programs

To run CICS applications, enter transactions from CICS terminals. You can also invoke CICS transactions by using the CICS transaction-invocation stored procedure. For information about this stored procedure, see Appendix J, "DB2-supplied stored procedures," on page 1255

CICS transactions that issue SQL statements must be link-edited with the CICS attachment facility language interface module, DSNCLI, and the CICS command language interface module. CICS application programs can issue SQL, DL/I, or CICS commands. After CICS has connected to DB2, any authorized CICS transaction can issue SQL requests that can write to and read from multiple DB2 instances using the distributed data facility. The application programs run as CICS applications.

DB2 checks an authorization ID related to the transaction against a plan assigned to it. The authorization ID for the transaction can be the operator ID, terminal ID, transaction ID, RACF-authenticated USERID, or another identifier explicitly provided by the resource control table (RCT). See Chapter 11, "Controlling access to a DB2 subsystem," on page 231 for more information about DB2 authorization IDs.

Running batch application programs

Batch DB2 work can run in background under the TSO terminal monitor program (TMP) or in an IMS batch message processing (BMP) region. IMS batch regions can issue SQL statements.

Batch work is run in the TSO background under the TSO terminal monitor program (TMP). The input stream can invoke TSO command processors, particularly the DSN command processor for DB2, and can include DSN subcommands such as RUN. The following is an example of a TMP job:

```
//jobname  JOB  USER=SYSOPR ...
//GO       EXEC PGM=IKJEFT01,DYNAMNBR=20
.
user DD statements
.
//SYSTSPRT DD   SYSOUT=A
//SYSTSIN  DD   *
DSN SYSTEM (ssid)
.
subcommand  (for example, RUN)
.
END
/*
```

In the example,

- IKJEFT01 identifies an entry point for TSO TMP invocation. Alternate entry points defined by TSO are also available to provide additional return code and ABEND termination processing options. These options permit the user to select the actions to be taken by the TMP upon completion of command or program execution.

  Because invocation of the TSO TMP using the IKJEFT01 entry point might not be suitable for all user environments, refer to the TSO publications to determine which TMP entry point provides the termination processing options best suited to your batch execution environment.
- USER=SYSOPR identifies the user ID (SYSOPR in this case) for authorization checks.
- DYNAMNBR=20 indicates the maximum number of data sets (20 in this case) that can be dynamically allocated concurrently.
- z/OS checkpoint and restart facilities do not support the execution of SQL statements in batch programs invoked by RUN. If batch programs stop because of errors, DB2 backs out any changes made since the last commit point. For information about backup and recovery, see Chapter 21, “Backing up and recovering databases,” on page 471. For an explanation of backing out changes to data when a batch program run in the TSO background abends, see Part 5 of *DB2 Application Programming and SQL Guide*.
- (*ssid*) is the subsystem name or group attachment name.

Running application programs using CAF

General-use Programming Interface

The call attachment facility (CAF) allows you to customize and control your execution environments more extensively than the TSO, CICS, or IMS attachment facilities. Programs executing in TSO foreground or TSO background can use either the DSN session or CAF; each has advantages and disadvantages. z/OS batch and started task programs can use only CAF.

IMS batch applications can also access DB2 databases through CAF, though this method does not coordinate the commitment of work between the IMS and DB2 systems. Using the DB2 DL/I batch support for IMS batch applications is highly recommended.

To use CAF, you must first make available a load module known as the call attachment language interface or DSNALI. When the language interface is available, your program can use CAF to connect to DB2 in two ways:

- Implicitly, by including SQL statements or IFI calls in your program just as you would any program
- Explicitly, by writing CALL DSNALI statements

For an explanation of CAF's capabilities and how to use it, see Part 6 of *DB2 Application Programming and SQL Guide*.

End of General-use Programming Interface

Running application programs using RRSAF

General-use Programming Interface

The Resource Recovery Services attachment facility (RRSAF) is a DB2 attachment facility that relies on a z/OS component called Resource Recovery Services (z/OS RRS). z/OS RRS provides system-wide services for coordinating two-phase commit operations across z/OS products.

Before you can run an RRSAF application, z/OS RRS must be started. RRS runs in its own address space and can be started and stopped independently of DB2.

To use RRSAF, you must first make available a load module known as the RRSAF language interface or DSNRLI. When the language interface is available, your program can use RRSAF to connect to DB2 in two ways:

- Implicitly, by including SQL statements or IFI calls in your program just as you would any program.
- Explicitly, by using CALL DSNRLI statements to invoke RRSAF functions. Those functions establish a connection between DB2 and RRS and allocate DB2 resources.

See "Controlling RRS connections" on page 397 for a description of how applications connect to DB2 using RRSAF. For an explanation of RRSAF's capabilities and how to use it, see Part 6 of *DB2 Application Programming and SQL Guide*.

End of General-use Programming Interface

Receiving messages

DB2 message identifiers have the form DSN*cxxxt*, where:

DSN Is the unique DB2 message prefix.

c Is a 1-character code identifying the DB2 subcomponent that issued the message. For example:
- **2** CICS attachment facility that is shipped with CICS
- **M** IMS attachment facility
- **U** Utilities

xxx Is the message number

t Is the message type, with these values and meanings:
- **A** Immediate action
- **D** Immediate decision
- **E** Eventual action
- **I** Information only

See *DB2 Messages* for an expanded description of message types.

A command prefix, identifying the DB2 subsystem, precedes the message identifier, except in messages from the CICS and IMS attachment facilities. (The CICS attachment facility issues messages in the form DSN2*xxxt*, and the IMS attachment facility issues messages in the form DSNM*xxxt*.) CICS and IMS attachment facility messages identify the z/OS subsystem that generated the message.

The IMS attachment facility issues messages that are identified as SSNM*xxxx* and as DFS*xxxx*. The DFS*xxxx* messages are produced by IMS, under which the IMS attachment facility operates.

Receiving unsolicited DB2 messages

Unsolicited subsystem messages are sent to the z/OS console issuing the START DB2 command, or to consoles assigned the routing codes that were listed in the DSNZP*xxx* module when installing DB2. But the following messages from the IMS and the CICS attachment facilities are exceptions to that rule:

- Specific IMS attachment facility messages are sent to the IMS master terminal.
- Unsolicited CICS messages are sent to the transient data entries specified in the RCT (ERRDEST).
- CICS statistics messages that are issued because of shutdown are sent to the transient data entry specified in the RCT (SHDDEST).

Some DB2 messages sent to the z/OS console are defined as critical using the WTO descriptor code (11). This code signifies “critical eventual action requested” by DB2. Preceded by an at sign (@) or an asterisk (*), critical DB2 messages remain on the screen until specifically deleted. This prevents them from being missed by the operator, who is required to take a specific action.

Determining operational control

Table 91 summarizes the operational control that is available at the operator console or terminal.

Table 91. Operational control summary

Type of operation	z/OS console	TSO terminal	IMS master terminal	Authorized CICS terminal
Issue DB2 commands and receive replies	Yes	Yes[1]	Yes[1]	Yes[1]
Receive DB2 unsolicited output	Yes	No	No	No
Issue IMS commands	Yes[2]	No	Yes	No

Table 91. Operational control summary (continued)

Type of operation	z/OS console	TSO terminal	IMS master terminal	Authorized CICS terminal
Receive IMS attachment facility unsolicited output	No[3]	No	Yes	No
Issue CICS commands	Yes[4]	No	No	Yes
Receive CICS attachment facility unsolicited output	No[3]	No	No	Yes[5]

Notes:

1. Except START DB2. Commands issued from IMS must have the prefix /SSR. Commands issued from CICS must have the prefix DSNC.
2. Using outstanding WTOR.
3. "Attachment facility unsolicited output" does not include "DB2 unsolicited output"; for the latter, see "Receiving unsolicited DB2 messages" on page 332.
4. Use the z/OS command MODIFY *jobname*, *CICS command*. The z/OS console must already be defined as a CICS terminal.
5. Specify the output destination for the unsolicited output of the CICS attachment facility in the RCT.

Chapter 16. Scheduling administrative tasks

The administrative task scheduler runs tasks that are defined in a task list according to a requested schedule. Tasks can be stored procedures or JCL jobs.

You manage the scheduler task list through DB2 stored procedures that add and remove tasks. You can monitor the task list and the status of executed tasks through user-defined functions that are provided as part of DB2.

Tasks run according to a defined schedule, which can be based on an interval, a point in time, or an event. Activity can be further restricted by a limit on the number of invocations or by earliest and latest invocation dates.

Interacting with the administrative task scheduler

The administrative task scheduler is based on scheduled tasks. DB2 users can add, remove and list scheduled tasks that are executed at planned points in time by the scheduler.

At each point in time when the scheduler detects that a task should be executed, it drives the task execution according to the work described in the task definition. There is no user interaction. The scheduler delegates the execution of the task to one of its execution threads, which executes the stored procedure or the JCL job described in the work definition of the task. The execution thread waits for the end of the execution and notifies the administrative task scheduler. The scheduler stores the execution status of the task in its redundant task lists, in relation with the task itself.

Adding a task

You can use the stored procedure ADMIN_TASK_ADD to define new scheduled tasks. The parameters that you use when you call the stored procedure define the schedule and the work for each task.

The request and the parameters are transmitted to the administrative task scheduler associated with the DB2 subsystem where the stored procedure has been called. The parameters are checked and if they are valid, the task is added into the scheduler task lists with a unique task name. The task name and the return code are returned to the stored procedure for output.

At the same time, the scheduler analyzes the task to schedule its next execution.

Scheduling capabilities of the administrative task scheduler

The administrative task scheduler can execute a task once or many times, at fixed points in time, or in response to events.

Five parameters define the scheduling behavior of the task, in one of four ways:

- *interval*: elapsed time between regular executions
- *point-in-time*: specific times for execution
- *trigger-task-name* alone: specific task to trigger execution
- *trigger-task-name* with *trigger-task-cond* and *trigger-task-code*: specific task with required result to trigger execution

© Copyright IBM Corp. 1982, 2008

Only one of these definitions can be specified for any single task. The other parameters must be null.

Table 92. Relationship of null and non-null values for scheduling parameters

Parameter specified	Required null parameters
interval	*point-in-time* *trigger-task-name* *trigger-task-cond* *trigger-task-code*
point-in-time	*interval* *trigger-task-name* *trigger-task-cond* *trigger-task-code*
trigger-task-name alone	*interval* *point-in-time* *trigger-task-cond* *trigger-task-code*
trigger-task-name with *trigger-task-cond* and *trigger-task-code*	*interval* *point-in-time*

If *interval*, *point-in-time*, *trigger-task-name*, *trigger-task-cond*, and *trigger-task-code* are all null, *max-invocations* must be set to 1.

You can restrict scheduled executions either by defining a window of time during which execution is permitted or by specifying how many times a task can execute. Three parameters control restrictions:

- *begin-timestamp*: earliest permitted execution time
- *end-timestamp*: latest permitted execution time
- *max-invocations*: maximum number of executions

The *begin-timestamp* and *end-timestamp* parameters are timestamps that define a window of time during which tasks can start. Before and after this window, the task will not start even if the schedule parameters are met. If *begin-timestamp* is null, the window begins at the time when the task is added, and executions can start immediately. If *end-timestamp* is null, the window extends infinitely into the future, so that repetitive or triggered executions are not limited by time. Timestamps must either be null values or future times, and *end-timestamp* cannot be earlier than *begin-timestamp*.

For repetitive or triggered tasks, the number of executions can be limited using the *max-invocations* parameter. In this case, the task executes no more than the number of times indicated by the parameter, even if the schedule and the window of time would require the task to be executed. Executions that are skipped because they overlap with previous executions that are still running are not counted toward *max-invocations*.

The *max-invocations* parameter defines a limit but no requirement. If the task is executed fewer times than indicated during its execution window, the maximum number of executions will never be reached.

Defining task schedules

Use different combinations of parameters to define schedules for task executions.

The following ADMIN_TASK_ADD parameters provide control over when scheduled tasks execute:

- *interval*
- *point-in-time*
- *trigger-task-name*
- *trigger-task-cond*
- *trigger-task-code*
- *begin-timestamp*
- *end-timestamp*
- *max-invocations*

To define a new scheduled task, connect to the DB2 subsystem with sufficient authorization to call the stored procedure ADMIN_TASK_ADD. The following task definitions show some common scheduling options.

To define	Do this
A task that executes one time only:	Set *max-invocations* to 1. Optionally, provide a value for the *begin-timestamp* parameter to control when execution happens. Leave other parameters null. For example, if *max-invocations* is set to 1 and *begin-timestamp* is set to 2008-05-27-06.30.0, the task executes at 6:30 AM on May 27, 2008. With this definition, the task executes one time. If *begin-timestamp* has been provided, execution happens as soon as permitted.
A regular repetitive execution:	Set *interval* to the number of minutes that you want to pass between the start of one execution and the start of the next execution. Optionally, provide values for the *max-invocations*, *begin-timestamp*, and *end-timestamp* parameters to limit execution. Leave other parameters null. For example, if *interval* is set to 5 and *begin-timestamp* is set to 2008-05-27-06.30.0, the task executes at 6:30 AM on May 27, 2008, then again at 6:35, 6:40, and so forth. With this definition, the task executes every *interval* minutes, so long as the previous execution has finished. If the previous execution is still in progress, the new execution is postponed *interval* minutes. Execution continues to be postponed until the running task completes.

To define	Do this
An irregular repetitive execution:	Set *point-in-time* to a valid UNIX cron format string. The string specifies a set of times. Optionally, provide values for the *max-invocations*, *begin-timestamp* and *end-timestamp* parameters to limit execution. Leave other parameters null. For example, if *point-in-time* is set to 0 22 * * 1,5, the task executes at 10:00 PM each Monday and Friday. With this definition, the task executes at each time specified, so long as the previous execution has finished. If the previous execution is still in progress, the new execution is skipped. Subsequent executions continue to be skipped until the running task completes.
An execution that is triggered when another task completes:	Set *trigger-task-name* to the name of the triggering task. Optionally set *trigger-task-cond* and *trigger-task-code* to limit execution based on the result of the triggering task. The *trigger-task-cond* and *trigger-task-code* parameters must either both be null or both be non-null. Optionally, provide values for the *max-invocations*, *begin-timestamp* and *end-timestamp* parameters to limit execution. Leave other parameters null. For example, assume that a scheduled INSERT job has a task name of test_task. If *trigger-task-name* is test _task, *trigger-task-cond* is EQ, and *trigger-task-code* is 0, then this task executes when the INSERT job completes with a return code of 0. With this definition, the task executes at each time specified, so long as the previous execution has finished. If the previous execution is still in progress, the new execution is skipped. Subsequent executions continue to be skipped until the running task completes.

To define	Do this
An execution that is triggered when DB2 starts:	Set *trigger-task-name* to DB2START. Optionally, provide values for the *max-invocations*, *begin-timestamp* and *end-timestamp* parameters to limit execution. Leave other parameters null. For example, if *trigger-task-name* is DB2START, *begin-timestamp* is 2008-01-01-00.00.0, and *end-timestamp* is 2009-01-01-00.00.0, the task executes each time that DB2 starts during 2008. With this definition, the task executes at each DB2 start, so long as the previous execution has finished. If the previous execution is still in progress, the new execution is skipped. Subsequent executions continue to be skipped until the running task completes.
An execution that is triggered when DB2 stops:	Set *trigger-task-name* to DB2STOP. Optionally, provide values for the *max-invocations*, *begin-timestamp* and *end-timestamp* parameters to limit execution. Leave other parameters null. With this definition, the task executes at each DB2 stop, so long as the previous execution has finished. If the previous execution is still in progress, the new execution is skipped. Subsequent executions continue to be skipped until the running task completes.

Choosing an administrative task scheduler in a data sharing environment

In a data sharing group, tasks can be added, removed, or executed in any of the administrative task schedulers with the same result. Tasks are not localized to a scheduler: a task can be added by one scheduler, and then executed by any of the schedulers in the data sharing group.

To force a task to be executed on a particular scheduler, specify the associated DB2 subsystem ID in the *DB2-SSID* parameter when you schedule the task.

ADMIN_TASK_ADD

The SYSPROC.ADMIN_TASK_ADD stored procedure adds a task to the scheduler task list.

GUPI

Environment

ADMIN_TASK_ADD runs in a WLM-established stored procedure address space and uses the Resource Recovery Services attachment facility to connect to DB2.

Authorization

Anyone who can execute this DB2 stored procedure is allowed to add a task.

Syntax

The following syntax diagram shows the SQL CALL statement for invoking this stored procedure:

```
►►─CALL─SYSPROC.ADMIN_TASK_ADD─(─┬─user-ID─,─password─┬─,─┬─begin-timestamp─┬─,──────────────►
                                 └─NULL─,─NULL────────┘   └─NULL────────────┘

►─┬─end-timestamp─┬─,─┬─max-invocations─┬─,──────────────────────────────────────────────────►
  └─NULL──────────┘   └─NULL────────────┘

►─┬─interval─,─NULL─,─NULL─,─NULL─,─NULL─,──────────────────────────────────────────────────┬►
  ├─NULL─,─point-in-time─,─NULL─,─NULL─,─NULL─,──────────────────────────────────────────┤
  └─NULL─,─NULL─,─trigger-task-name─,─┬─trigger-task-cond─,─trigger-task-code─,─┬──────────┤
                                      └─NULL─,─NULL─,──────────────────────────┘

►─┬─DB2-SSID─┬─,──────────────────────────────────────────────────────────────────────────►
  └─NULL─────┘

►─┬─┬─procedure-schema─┬─,─procedure-name─,─┬─procedure-input─┬─,─NULL─,─NULL─,─NULL─,─┬──────►
  │ └─NULL─────────────┘                    └─NULL────────────┘                        │
  └─NULL─,─NULL─,─NULL─,─JCL-library─,─┬─JCL-member─┬─,─job-wait─,──────────────────────┘
                                       └─NULL───────┘

►─┬─task-name─┬─,─┬─description─┬─,─return-code─,─message─)──────────────────────────────►◄
  └─NULL──────┘   └─NULL────────┘
```

Option descriptions

user-ID
Specifies the user ID under which the task execution is performed.

If this parameter is set to NULL, task execution is performed with the default authorization ID associated with the administrative task scheduler instead.

This is an input parameter of type VARCHAR(128).

password
Specifies the password associated with the input parameter *user-ID*.

The value of *password* is passed to the stored procedure as part of payload, and is not encrypted. It is not stored in dynamic cache when parameter markers are used.

Recommendation: Have the application that invokes this stored procedure pass an encrypted single-use password called a *passticket*.

This is an input parameter of type VARCHAR(24). This parameter is NULL only when *user-ID* is set to NULL, and must be NULL when *user-ID* is NULL.

begin-timestamp
Specifies when a task can first begin execution. When task execution begins depends on how this and other parameters are set:

Non-null value for *begin-timestamp*

At *begin-timestamp*
The task execution begins at *begin-timestamp* if *point-in-time* and *trigger-task-name* are NULL.

Next point in time defined at or after *begin-timestamp*
The task execution begins at the next point in time defined at or after *begin-timestamp* if *point-in-time* is non-null.

When *trigger-task-name* **completes at or after** *begin-timestamp*
The task execution begins the next time that *trigger-task-name* completes at or after *begin-timestamp*.

Null value for *begin-timestamp*

Immediately
The task execution begins immediately if *point-in-time* and *trigger-task-name* are NULL.

Next point in time defined
The task execution begins at the next point in time defined if *point-in-time* is non-null.

When *trigger-task-name* **completes**
The task execution begins the next time that *trigger-task-name* completes.

The value of this parameter cannot be in the past, and it cannot be later than *end-timestamp*.

This is an input parameter of type TIMESTAMP.

end-timestamp
Specifies when a task can last begin execution. If this parameter is set to NULL, then the task can continue to execute as scheduled indefinitely.

The value of this parameter cannot be in the past, and it cannot be earlier than *begin-timestamp*.

This is an input parameter of type TIMESTAMP.

max-invocations
Specifies the maximum number of executions allowed for a task. This value applies to all schedules: triggered by events, recurring by time interval, and recurring by points in time. If this parameter is set to NULL, then there is no limit to the number of times this task can execute.

For tasks that execute only one time, *max-invocations* must be set to 1 and *interval*, *point-in-time* and *trigger-task-name* must be NULL.

If both *end-timestamp* and *max-invocations* are specified, the first limit reached takes precedence. That is, if *end-timestamp* is reached, even though the number of task executions so far has not reached *max-invocations*, the task will not be executed again. If *max-invocations* have occurred, the task will not be executed again even if *end-timestamp* is not reached.

This is an input parameter of type INTEGER.

interval
Defines a time duration between two executions of a repetitive regular task. The first execution occurs at *begin-timestamp*. If this parameter is set to NULL, the task is not regularly executed. If this parameter contains a non-null value, the parameters *point-in-time* and *trigger-task-name* must be set to NULL.

This is an input parameter of type INTEGER.

point-in-time
Defines one or more points in time when a task is executed. If this parameter is set to NULL, the task is not scheduled at fixed points in time. If this parameter contains a non-null value, the parameters *interval* and *trigger-task-name* must be set to NULL.

The *point-in-time* string uses the UNIX cron format. The format contains the following pieces of information separated by blanks: given minute or minutes, given hour or hours, given day or days of the month, given month or months of the year, and given day or days of the week. For each part, you can specify one or several values, ranges, and so forth.

This is an input parameter of type VARCHAR(400).

trigger-task-name
Specifies the name of the task which, when its execution is complete, will trigger the execution of this task.

Task names of DB2START and DB2STOP are reserved for DB2 stop and start events respectively. Those events are handled by the scheduler associated with the DB2 subsystem that is starting or stopping.

If this parameter is set to NULL, the execution of this task will not be triggered by another task. If this parameter contains a non-null value, the parameters *interval* and *point-in-time* must be set to NULL.

This is an input parameter of type VARCHAR(128).

trigger-task-cond
Specifies the type of comparison to be made to the return code after the execution of task *trigger-task-name*. Possible values are:

GT Greater than

GE Greater than or equal to

EQ Equal to

LT Less than

LE Less than or equal to

NE Not equal to

If this parameter is set to NULL, the task execution is triggered without considering the return code of task *trigger-task-name*. This parameter must be set to NULL if *trigger-task-name* is set to NULL or is either DB2START or DB2STOP.

This is an input parameter of type CHAR(2).

trigger-task-code
Specifies the return code from executing *trigger-task-name*.

If the execution of this task is triggered by a stored procedure, *trigger-task-code* contains the SQLCODE that must be returned by the triggering stored procedure in order for this task to execute.

If the execution of this task is triggered by a JCL job, *trigger-task-code* contains the MAXRC that must be returned by the triggering job in order for this task to execute.

To find out what the MAXRC or SQLCODE of a task is after execution, invoke the user-defined function DSNADM. ADMIN_TASK_STATUS returns these information in the columns MAXRC and SQLCODE.

The following restrictions apply to the value of *trigger-task-code*:

- If *trigger-task-cond* is null, then *trigger-task-code* must also be null.
- If *trigger-task-cond* is non-null, then *trigger-task-code* must also be non-null.

If *trigger-task-cond* and *trigger-task-code* are not null, they are used to test the return code from executing *trigger-task-name* to determine whether to execute this task or not.

For example, if *trigger-task-cond* is set to "GE" and *trigger-task-code* is set to "8", then this task will execute if and only if the previous execution of *trigger-task-name* returned a MAXRC (for a JCL job) or an SQLCODE (for a stored procedure) greater than or equal to 8.

This is an input parameter of type INTEGER.

DB2-SSID
Specifies the DB2 subsystem ID whose associated scheduler should execute the task.

This parameter is used in a data sharing environment where, for example different DB2 members have different configurations and executing the task relies on a certain environment. However, specifying a value in *DB2-SSID* will prevent schedulers of other members to execute the task, so that the task can only be executed as long as the scheduler of *DB2-SSID* is running.

For a task being triggered by a DB2 start or DB2 stop event in *trigger-task-name*, specifying a value in *DB2-SSID* will let the task be executed only when the named subsystem is starting and stopping. If no value is given, each member that starts or stops will trigger a local execution of the task, provided that the executions are serialized.

If this parameter is set to NULL, any scheduler can execute the task.

This is an input parameter of type VARCHAR(4).

procedure-schema
Specifies the schema of the DB2 stored procedure this task will execute. If this parameter is set to NULL, DB2 uses a default schema. This parameter must be set to NULL if *procedure-name* is set to NULL.

This is an input parameter of type VARCHAR(128).

procedure-name
Specifies the name of the DB2 stored procedure this task will execute. If this parameter is set to NULL, no stored procedure will be called. In this case, a JCL job must be specified.

This is an input parameter of type VARCHAR(128).

procedure-input
Specifies the input parameters of the DB2 stored procedure this task will execute. This parameter must contain a DB2 SELECT statement that returns one row of data. The returned values will be passed as parameter to the stored procedure.

If this parameter is set to NULL, no parameters are passed to the stored procedure. This parameter must be set to NULL when *procedure-name* is set to NULL.

This is an input parameter of type VARCHAR(4096).

JCL-library
Specifies the name of the data set where the JCL job to be executed is saved.

If this parameter is set to NULL, no JCL job will be executed. In this case, a stored procedure must be specified.

This is an input parameter of type VARCHAR(44).

JCL-member
Specifies the name of the library member where JCL job to be executed is saved.

If this parameter is set to NULL, the data set specified in *JCL-library* must be sequential and contain the JCL job to be executed. This parameter must be set to NULL if *JCL-library* is set to NULL.

This is an input parameter of type VARCHAR(8).

job-wait
Specifies whether the job can be executed synchronously or not. This parameter can only be set to NULL if *JCL-library* is set to NULL. Otherwise, it must be one of the following values:

NO Asynchronous execution

YES Synchronous execution

PURGE
Synchronous execution after which the job status in z/OS is purged

This is an input parameter of type VARCHAR(8).

task-name
Specifies a unique name assigned to the task.

A unique task name is returned when the task is created with a NULL *task-name* value. This name is of the format "TASK_ID_*xxxx*" where *xxxx* is 0001 for the first task named, 0002 for the second task, and so forth.

The following task names are reserved and cannot be given as the value of *task-name*:

- Names starting with "TASK_ID_"
- DB2START
- DB2STOP

This is an input-output parameter of type VARCHAR(128).

description
Specifies a description assigned to the task.

This is an input parameter of type VARCHAR(128).

return-code
Provides the return code from the stored procedure. Possible values are:

0 The call completed successfully.

12 The call did not complete successfully. The *message* output parameter contains messages describing the error.

This is an output parameter of type INTEGER.

message
Contains messages describing the error encountered by the stored procedure. The first messages in this area, if any, are generated by the stored procedure. Messages that are generated by DB2 might follow the first messages.

This is an output parameter of type VARCHAR(1331).

Example

The following Java sample shows how to invoke ADMIN_TASK_ADD:

```
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import java.sql.Timestamp;
import java.sql.Types;

Connection con = DriverManager.getConnection
   ("jdbc:db2://myserver:myport/mydatabase", "myuser", "mypassword");
CallableStatement callStmt = con.prepareCall
   ("CALL SYSPROC.ADMIN_TASK_ADD("
   + "?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)");
// provide the authid
callStmt.setString(1, "myexecuser");
// provide the password
callStmt.setString(2, "myexecpwd");
// set the start time to now
callStmt.setNull(3, Types.TIMESTAMP);
// no end time
callStmt.setNull(4, Types.TIMESTAMP);
// set the max invocation
callStmt.setInt(5, 1);
// This is a non recurrent task
callStmt.setNull(6, Types.INTEGER);
callStmt.setNull(7, Types.VARCHAR);
callStmt.setNull(8, Types.VARCHAR);
callStmt.setNull(9, Types.CHAR);
callStmt.setNull(10, Types.INTEGER);
callStmt.setNull(11, Types.VARCHAR);
// provide the stored procedure schema
callStmt.setString(12, "MYSCHEMA");
// provide the name of the stored procedure to be executed
callStmt.setString(13, "MYPROC");
// provide the stored procedure input parameter
callStmt.setString(14, "SELECT 1 FROM SYSIBM.SYSDUMMY1");
// This is not a JCL job
callStmt.setNull(15, Types.VARCHAR);
callStmt.setNull(16, Types.VARCHAR);
callStmt.setNull(17, Types.VARCHAR);
// add a new task with task name mytask
callStmt.setString(18, "mytask");
callStmt.registerOutParameter(18, Types.VARCHAR);
// provide the task description
callStmt.setString(19, "MY DESCRIPTION");
// register output parameters for error management
callStmt.registerOutParameter(20, Types.INTEGER);
callStmt.registerOutParameter(21, Types.VARCHAR);
// execute the statement
callStmt.execute();
// manage the return code
if ( callStmt.getInt(20) == 0 )
{
  System.out.print("\nSuccessfully added task " + callStmt.getString(18));
}
else
{
  System.out.print("\nError code and message are: "
     + callStmt.getInt(20) + "/" + callStmt.getString(21));
}
```

Output

The output of this stored procedure is the task name, *task-name* and the following output parameters, which are described in "Option descriptions" on page 340:

- *return-code*
- *message*

GUPI

UNIX cron format

The UNIX cron format is a way of specifying time for the *point-in-time* parameter of the ADMIN_TASK_ADD stored procedure.

The cron format has five time and date fields separated by at least one blank. There can be no blank within a field value. Scheduled tasks are executed when the minute, hour, and month of year fields match the current time and date, and at least one of the two day fields (day of month, or day of week) match the current date.

The allowed values for the time and date fields are:

Field **Allowed values**

minute
0-59

hour 0-23

day of month
1-31

month

- 1-12, where 1 is January
- Upper-, lower-, or mixed-case three-character strings, based on the English name of the month: jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, or dec.

day of week

- 0-7, where 0 or 7 is Sunday
- Upper-, lower-, or mixed-case three-character strings, based on the English name of the day: mon, tue, wed, thu, fri, sat, or sun.

Ranges and lists

Ranges of numbers are allowed. Ranges are two numbers separated with a hyphen. The specified range is inclusive.

Example: The range 8-11 for an hour entry specifies execution at hours 8, 9, 10 and 11.

Lists are allowed. A list is a set of numbers or ranges separated by commas.

Examples:

```
1,2,5,9
0-4,8-12
```

Unrestricted range

A field can contain an asterisk (*), which represents all possible values in the field.

The day of a command's execution can be specified by two fields: day of month and day of week. If both fields are restricted by the use of a value other than the asterisk, the command will run when either field matches the current time.

Example: The value 30 4 1,15 * 5 causes a command to run at 4:30 AM on the 1st and 15th of each month, plus every Friday.

Step values

Step values can be used in conjunction with ranges. The syntax *range*/*step* defines the range and an execution interval.

If you specify *first*-*last*/*step*, execution takes place at *first*, then at all successive values that are distant from *first* by *step*, until *last*.

Example: To specify command execution every other hour, use 0-23/2. This expression is equivalent to the value 0,2,4,6,8,10,12,14,16,18,20,22.

If you specify */*step*, execution takes place at every interval of *step* through the unrestricted range.

Example: As an alternative to 0-23/2 for execution every other hour, use */2.

Listing scheduled tasks

You can use the function ADMIN_TASK_LIST to list tasks that are scheduled for execution by the administrative task scheduler.

To list scheduled tasks, connect to the DB2 subsystem with sufficient authorization to call the function ADMIN_TASK_LIST. See the information about ADMIN_TASK_LIST in DB2 SQL Reference.

The function contacts the scheduler in order to update the DB2 task list in the table SYSIBM.ADMIN_TASKS if necessary, and then reads the tasks from the DB2 task list directly. The parameters that were used to create the task are column values of the returned table. The table also includes the authorization ID of the task creator, in the CREATOR column, and the time that the task was created, in the LAST_MODIFIED column.

Listing the last execution status of scheduled tasks

You can use the function ADMIN_TASK_STATUS to view the last execution status of scheduled tasks.

Before a task is first scheduled, all columns of its execution status contain null values, as returned by the table function ADMIN_TASK_STATUS. Afterwards, at least the TASK_NAME, USERID, DB2_SSID, STATUS, NUM_INVOCATIONS and START_TIMESTAMP columns contain a non-null value. This information indicates when and under which user ID the task status last changed, as well as the number of times this task was already executed. The rest of the execution status can be interpreted according to the different values of the STATUS column.

The table function ADMIN_TASK_STATUS contacts the administrative task scheduler in order to update the DB2 task list in table SYSIBM.ADMIN_TASKS, if necessary, and then reads the tasks from this task list directly.

To determine the last execution status of a scheduled task:

1. Execute the table function ADMIN_TASK_STATUS to generate the status table. See the information about ADMIN_TASK_LIST in DB2 SQL Reference.
2. Select the rows in the table that correspond to the task name.

   Tip: You can relate the task execution status to the task definition by joining the output tables from ADMIN_TASK_LIST and ADMIN_TASK_STATUS on the TASK_NAME column.

The table created by ADMIN_TASK_STATUS indicates the last execution of scheduled tasks. Each row is indexed by the task name and contains the last execution status of the corresponding task.

If task execution has never been attempted, because the execution criteria have not been met, the STATUS column contains a null value.

If the scheduler was not able to start executing the task, the STATUS column contains NOTRUN. The START_TIMESTAMP and END_TIMESTAMP columns are the same, and the MSG column indicates why the task execution could not be started. All JCL job execution status columns are NULL, but the DB2 execution status columns contain values if the reason for the failure is related to DB2. (For example, a DB2 connection could not be established.)

If the scheduler started executing the task but the task has not yet completed, the STATUS column contains RUNNING. All other execution status columns contain null values.

If the task execution has completed, the STATUS column contains COMPLETED. The START_TIMESTAMP and END_TIMESTAMP columns contain the actual start and end times. The MSG column might contain informational or error messages. The DB2 and JCL columns are filled with values when they apply.

If the scheduler was stopped during the execution of a task, the status remains RUNNING until the scheduler is restarted. When the scheduler starts again, the status is changed to UNKNOWN, because the scheduler cannot determine if the task was completed.

Removing a scheduled task

You can remove a scheduled task from the task list using the stored procedure ADMIN_TASK_REMOVE.

Even if a task has finished all of its executions and will never be executed again, it remains in the task list until it is explicitly removed through a call to the stored procedure ADMIN_TASK_REMOVE.

Restrictions:

- Only the user who scheduled a task or a user with SYSOPR, SYSADM, or SYSCTRL authority can delete a task.
- A task cannot be removed while it is executing.
- A task that is the trigger for another task cannot be removed.

To remove a scheduled task:

1. Optional: Issue the following SQL statement to identify tasks that will never execute again:

```
SELECT T.TASK_NAME
FROM TABLE (DSNADM.ADMIN_TASK_LIST()) T,
     TABLE (DSNADM.ADMIN_TASK_STATUS()) S
WHERE T.TASK_NAME = S.TASK_NAME AND
      (S.NUM_INVOCATIONS = T.MAX_INVOCATIONS OR
       T.END_TIMESTAMP < CURRENT TIMESTAMP) AND
      STATUS <> 'RUNNING'
```

2. Confirm the name of the task that you want to remove.
3. Call the stored procedure ADMIN_TASK_REMOVE. You must provide the task name as a parameter to the stored procedure. The scheduled task is removed from the task list and its last execution status is deleted. Listing the scheduled tasks and execution statuses no longer returns a row for this task. The task name is freed up for future reuse.

ADMIN_TASK_REMOVE

The SYSPROC.ADMIN_TASK_REMOVE stored procedure removes a task from the task list of the administrative scheduler. If the task is currently running, it continues to execute until completion, and the task is not removed from the administrative scheduler task list. If other tasks depend on the execution of the task to be removed, this task is not removed from the administrative scheduler task list.

GUPI

Environment

See the recommended environment in the installation job DSNTIJRA.

Authorization

Users with SYSOPR, SYSCTRL, or SYSADM authority can remove any task. Other users who have EXECUTE authority on this stored procedure can remove tasks that they added. Attempting to remove a task that was added by a different user returns an error in the output.

Syntax

The following syntax diagram shows the SQL CALL statement for invoking this stored procedure:

```
►►──CALL──SYSPROC.ADMIN_TASK_REMOVE──(──task-name,──return-code,──message──)──────────────────────►◄
```

Option descriptions

task-name

Specifies the task name of the task to be removed from the administrative scheduler task list.

This is an input parameter of type VARCHAR(128) and cannot be null.

return-code
Provides the return code from the stored procedure. Possible values are:

0 The call completed successfully.

12 The call did not complete successfully. The *message* output parameter contains messages describing the error.

This is an output parameter of type INTEGER.

message
Contains messages describing the error encountered by the stored procedure. The first messages in this area, if any, are generated by the stored procedure. Messages that are generated by DB2 might follow the first messages.

This is an output parameter of type VARCHAR(1331).

Example

The following Java sample shows how to invoke ADMIN_TASK_REMOVE:

```
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import java.sql.Timestamp;
import java.sql.Types;

Connection con =
  DriverManager.getConnection("jdbc:db2://myserver:myport/mydatabase",
  "myuser", "mypassword");
CallableStatement callStmt =
  con.prepareCall("CALL SYSPROC.ADMIN_TASK_REMOVE(?, ?, ?)");
// provide the id of the task to be removed
callStmt.setString(1, "mytask");
// register output parameters for error management
callStmt.registerOutParameter(2, Types.INTEGER);
callStmt.registerOutParameter(3, Types.VARCHAR);
// execute the statement callStmt.execute();
// manage the return code
if ( callStmt.getInt(2) == 0 )
  {
  System.out.print("\nSuccessfully removed task "
                   + callStmt.getString(1));
  }
else
  {
  System.out.print("\nError code and message are: "
                   + callStmt.getInt(2) + "/"
                   + callStmt.getString(3));
  }
```

Output

The output of this stored procedure includes the following output parameters, which are described in “Option descriptions” on page 349:

- *return-code*
- *message*

GUPI

Manually starting the administrative task scheduler

The administrative task scheduler normally starts when DB2 starts, but you can start it manually if necessary.

Use one of the following commands to start the scheduler:

- To start a scheduler named *admtproc* from the operator's console using the default tracing option, issue the MVS system command:

  ```
  start admtproc
  ```

- To start a scheduler named *admtproc* from the operator's console with tracing enabled, issue the MVS system command:

  ```
  start admtproc,trace=on
  ```

- To start a scheduler named *admtproc* from the operator's console with tracing disabled, issue the MVS system command:

  ```
  start admtproc,trace=off
  ```

When the administrative scheduler starts, message DSNA671I displays on the console.

Manually stopping the administrative task scheduler

You can stop the administrative task scheduler as part of problem determination or in preparation for maintenance.

To stop the scheduler, use one of the following commands.

- **Recommended method:** To stop a scheduler named *admtproc* from the operator's console, issue the MVS system command:

  ```
  modify admtproc,appl=shutdown
  ```

  – The scheduler stops accepting requests and will not start new task executions. It waits until the execution of all currently running tasks completes and then terminates.

  – Message DSNA670I displays on the console.

- **Alternate method:** If the MODIFY command does not shut down the scheduler, issue the MVS system command:

  ```
  stop admtproc
  ```

  – Any task that was invoked by the scheduler and is currently executing is interrupted.

  – Message DSNA670I displays on the console.

  Interrupted tasks keep their status as RUNNING and are not rescheduled until the scheduler is started again. At startup, the status of the interrupted tasks is set to UNKNOWN and message DSNA690I is written into the status. Look for UNKNOWN in the results of the ADMIN_TASK_STATUS user-defined function. If UNKNOWN is present in the STATUS column of the output table, then you should check to see if the task has completed. If an interrupted task has not completed, you should terminate the work.

Synchronization between administrative task schedulers in a data sharing environment

The administrative task schedulers of a data sharing group synchronize themselves through their task lists.

When a task is added, the stored procedure ADMIN_TASK_ADD is called by a DB2 member. The scheduler associated with this DB2 member adds the task to the

common task list. The task list is shared among all schedulers associated with the data sharing group members. The next time a scheduler accesses the list, it will detect the new task.

All schedulers of the data sharing group access this task list once per minute, to check for new tasks. The scheduler that adds a task does not have to check the list, and can execute the task immediately. Any other scheduler can execute a task only after to finding it in the updated task list. Any scheduler can remove a task without waiting.

In order to remove a task, the stored procedure ADMIN_TASK_REMOVE is called by a DB2 member. The scheduler associated with this DB2 member removes the task from the common task list. The next time a scheduler checks the list, within one minute after the task has been removed, it detects that the task has been deleted.

No scheduler can execute a task without first locking it in the task list. This locking prevents deleted tasks from being executed: the task is no longer present in the list, so it cannot be locked. Because it cannot be locked, it cannot be executed. The locking also prevents double executions: a task that one scheduler has in progress is already locked, so no other scheduler can lock the task.

Troubleshooting the administrative task scheduler

Error and informational messages from the administrative task scheduler might display on the console. Error messages typically indicate a problem with the configuration of the scheduler itself or an error accessing its resources. Informational messages identify important steps in the life cycle of the scheduler: starting, stopping, automatically recovering one of the task lists, and so forth.

Enabling tracing for administrative task scheduler problem determination

If you need to perform problem determination on the administrative task scheduler in response to a message, you can use a trace.

Use the MVS system command MODIFY to enable or disable tracing.

- To start a trace for a scheduler named *admtproc*, issue the following MVS system command:

  ```
  modify admtproc,appl=trace=on
  ```

  Substitute the name of your scheduler for *admtproc*.
- To stop a trace for a scheduler named *admtproc*, issue the following MVS system command:

  ```
  modify admtproc,appl=trace=off
  ```
- To configure the system so that tracing starts automatically when the scheduler starts, modify the procedure parameter TRACE in the JCL job that starts the scheduler. This job has the name that was assigned when the scheduler was installed. The job was copied into one of the PROCLIB library during the installation. Specify TRACE=ON.

  To disable tracing, change the parameter to TRACE=OFF.

Recovering the administrative task scheduler task list

Two redundant, active copies of the task list exist in case of media failure. Console message DSNA679I indicates that one or both of these copies is not accessible or is corrupted. If this happens, you can recover the task list.

One copy of the task list is a shared VSAM data set, by default DSNC910.TASKLIST, where DSNC910 is the DB2 catalog prefix. The other copy is stored in the table ADMIN_TASKS in the SYSIBM schema. Include these redundant copies as part of your backup and recovery plan.

Tip: If DB2 is offline, message DSNA679I displays on the console. As soon as DB2 starts, the administrative task scheduler performs an autonomic recovery of the ADMIN_TASKS table using the contents of the VSAM task list. When the recovery is complete, message DSNA695I displays on the console to indicate that both task lists are again available. (By default, message DSNA679I displays on the console once per minute when DB2 is offline. You can change the frequency of this message by modifying the ERRFREQ parameter either as part of the started task or with a console command.)

Use the following procedures to recover the task list if it is lost or damaged:

- To recover if the ADMIN_TASKS task list is corrupted:
  1. Create a new and operable version of the table.
  2. Grant SELECT, UPDATE, INSERT and DELETE privileges on the table to the administrative scheduler started task user.

  As soon as the ADMIN_TASKS table is accessible again, the scheduler performs an autonomic recovery of the table using the content of the VSAM task list.
- To recover if the VSAM file is corrupted, create an empty version of the VSAM task list. As soon as the VSAM task list is accessible again, the scheduler performs an autonomic recovery using the content of the ADMIN_TASKS task list.
- If both task lists (the VSAM data set and the ADMIN_TASKS table) are corrupted and inaccessible, the scheduler is no longer operable. Messages DSNA681I and DSNA683I display on the console and the scheduler terminates. To recover from this situation:
  1. Create an empty version of the VSAM task list.
  2. Recover the table space DSNADMDB.DSNADMTS, where the ADMIN_TASKS table is located.
  3. Restart the administrative scheduler.

  As soon as both task lists are accessible again, the scheduler performs an autonomic recovery of the VSAM task list using the content of the recovered ADMIN_TASKS table.

Problem executing a task

When the administrative task scheduler tries to execute a task without success, or when an error is detected by the scheduler at the end of a task execution, the error description is written into the last execution status.

Symptoms

A task was scheduled successfully, but the action did not complete or did not complete correctly.

Diagnosing the problem

Use the function ADMIN_TASK_STATUS to review the last execution status of a task and identify any messages or return codes that were passed back to the scheduler.

Important: The task status is overwritten as soon as the next execution of the task starts.

Resolving the problem

Correct the underlying problem and review the schedule. The task can now be executed successfully, but execution occurs only according to the schedule. Failed executions are not rescheduled. If the task is no longer scheduled, for example because it had a defined number of executions, you must remove it and add it again, with adjusted criteria. If the task is still scheduled, you do not need to take any further action unless the failed execution is required. You cannot adjust a schedule, so if you do require the failed execution and all future executions, you must remove the scheduled task and re-create it.

Problem in user-defined table functions

An SQL code and a few characters of an SQL error message are returned in response to either ADMIN_TASK_LIST or ADMIN_TASK_STATUS.

Symptoms

An SQL code is returned. When SQLCODE is -443, the error message cannot be read directly, because only a few characters are available.

Diagnosing the problem

Problem diagnosis and resolution depends on the SQLCODE returned.

-443
The SQLCODE of -443 indicates that an error occurred in the function. Use the message number, which is at the beginning of the truncated return string, to diagnose the problem.

Any other value
Any other SQLCODE indicates that the error is not in the function or in the scheduler. Troubleshoot the task itself.

Problem in stored procedures

An SQL code and a few characters of an SQL error message are returned in response to either ADMIN_TASK_ADD or ADMIN_TASK_REMOVE.

Symptoms

An SQL code is returned.

Diagnosing the problem

An SQL code other than 0 indicates that DB2 encountered a problem calling the stored procedure.

An SQL code of 0 is accompanied by a return code and an error message in the output parameters RETURN_CODE and MSG. The return code is 0 if the scheduled task could be added or removed successfully. If the return code is 12, an error occurred adding or removing the task, and the returned error message describes the cause. The first eight characters of the error message contain the error message ID.

Resolving the problem

Errors can originate with the stored procedure itself or with the scheduler, in which case the error information is transmitted back to the stored procedure for output. Most error messages are clearly in one category or the other. For example, `DSNA650I` *`csect-name`* `CANNOT CONNECT TO ADMIN SCHEDULER` *`proc-name`* indicates an error from the stored procedure. `DSNA652I` *`csect-name`* `THE USER` *`user-name`* `IS NOT ALLOWED TO ACCESS TASK` *`task-name`* belongs to the administrative task scheduler, which is checking the parameters and authorization information passed to it.

Understanding the source of the error should be enough to correct the cause of the problem. Most problems are incorrect usage of the stored procedure or an invalid configuration.

Correct the underlying problem and resubmit the call to the stored procedure to add or remove the task.

Architecture of the administrative task scheduler

The administrative task scheduler is a started task that can be seen as an additional DB2 address space, even if in a separate process. It is accessed through an SQL API and stores the scheduled tasks in two redundant task lists.

The scheduler is part of DB2 for z/OS. When properly configured, it is available and operable with the first DB2 start. The scheduler starts as a task on the z/OS system during DB2 startup. The scheduler has its own address space, named after the started task name.

Each DB2 subsystem has its own distinct scheduler connected to it. DB2 is aware of the scheduler whose name is defined in the subsystem parameter ADMTPROC. The scheduler is aware of DB2 by the subsystem name that is defined in the DB2SSID parameter of the started task.

The scheduler has an SQL interface consisting of stored procedures (ADMIN_TASK_ADD and ADMIN_TASK_REMOVE) and user-defined table functions (ADMIN_TASK_LIST and ADMIN_TASK_STATUS) defined in DB2. This SQL interface allows you to remotely add or remove administrative tasks and to list those tasks and their execution status.

The scheduler executes the tasks according to their defined schedules. The status of the last execution is stored in the task lists as well, and you can access it through the SQL interface.

Related reference

"ADMIN_TASK_ADD" on page 339
The SYSPROC.ADMIN_TASK_ADD stored procedure adds a task to the scheduler task list.

"ADMIN_TASK_REMOVE" on page 349
The SYSPROC.ADMIN_TASK_REMOVE stored procedure removes a task from the task list of the administrative scheduler. If the task is currently running, it continues to execute until completion, and the task is not removed from the administrative scheduler task list. If other tasks depend on the execution of the task to be removed, this task is not removed from the administrative scheduler task list.

The life cycle of the administrative task scheduler

The administrative task scheduler starts as a task on the z/OS system during DB2 startup or initialization. It remains active unless it is explicitly stopped, even when DB2 terminates.

Every DB2 subsystem has a coordinated administrative task scheduler address space that it can start with a z/OS started task procedure.

Two instances of the same scheduler cannot run simultaneously. To avoid starting up a duplicate scheduler, the scheduler at startup checks that there is no address space other than itself with the same name. If another address space with the same name is active, then the new scheduler immediately shut downs and displays a console message (DSNA674I). The scheduler can check the address spaces only in the same system, not the entire Sysplex.

When DB2 terminates, the administrative scheduler remains active so that scheduled JCL jobs can run. When DB2 starts again, it connects to the scheduler automatically. It does not need to be restarted.

Scheduler task lists

The administrative task scheduler manages tasks defined by users and stores them in two redundant task lists, so that the scheduler can continue working even if one task list is unavailable.

The two task lists are the DB2 table SYSIBM.ADMIN_TASKS and the VSAM data set that is indicated in the data definition ADMTDD1 of the scheduler started task. The scheduler maintains the consistency between the two task lists.

The DB2 task list, SYSIBM.ADMIN_TASKS, is accessed through a connection to the DB2 subsystem identified in the *DB2SSID* parameter of the scheduler started task.

The scheduler works and updates both task lists redundantly, and remains operable so long as at least one of the task lists is available. In this way, the scheduler continues working when DB2 is offline. If a task list becomes unavailable, the scheduler continues to update the task list. When both task lists are available again, the scheduler automatically synchronizes them.

Architecture of the administrative task scheduler in a data sharing environment

In a data sharing environment, there is one administrative task scheduler for each DB2 for z/OS Version 9.1 or later member of a data sharing group, even when those members run in the same z/OS system. The schedulers share their resources and interface.

The scheduler task list is shared by all schedulers in a data sharing group, accessing a shared task file on shared storage (VSAM data set defaulting to DSNC910.TASKLIST, where DSNC910 is the DB2 catalog prefix) and a redundant task list in the DB2 system table SYSIBM.ADMIN_TASKS.

Tasks are not localized to a scheduler. They can be added, removed, or executed in any of the schedulers in the data sharing group with the same result. However, you can force the task to execute on a given scheduler by specifying the associated DB2 subsystem ID in the *DB2-SSID* parameter when you schedule the task. The

tasks that have no affinity to a given DB2 subsystem are executed among all schedulers. Their distribution cannot be predicted.

Security guidelines for the administrative task scheduler

The administrative task scheduler employs controls on access and execution to help maintain a secure environment.

Installation job DSNTIJRA is responsible for establishing the security environment for the administrative task scheduler. Installation job DSNTIJSG is responsible for establishing the security environment in DB2 for accessing the scheduler interface.

Related information

BPX.DAEMON FACILITY
For more information about BPX.DAEMON and setting up program control, see z/OS UNIX System Services Planning at http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/bpxzb270/16.3.2.2

User roles in the administrative task scheduler

Three user roles are involved in the use of the administrative task scheduler: the started task user, the interface users and the execution users.

The scheduler started task is associated to the user STARTUID in RACF, so that the scheduler is running in the security context of this user. This user, the started task user, should have access to the resources of the scheduler: it is granted UPDATE access on the DB2 table SYSIBM.ADMIN_TASKS and it is allowed to write into the VSAM data set containing the redundant task list.

The users or groups of users who have access to the SQL interface of the scheduler are allowed to add, remove or list scheduled tasks. To specify who is authorized to add, remove or list a scheduled task, use the GRANT command in DB2. All interface users are granted EXECUTE access on the scheduler stored procedures and user-defined table functions. They also are granted READ access on the DB2 table SYSIBM.ADMIN_TASKS.

Each scheduled task in the scheduler is associated with an execution user who will execute this task. When not explicitly given by the user, a default execution user DFLTUID defined in the scheduler started task is used. The scheduler execution threads switch to the security context of this user before executing the task.

Protection of the interface of the administrative task scheduler

The administrative task scheduler interface is protected against unauthorized access by other users. Credentials of a task are checked but not stored.

Users with EXECUTE rights on one of the stored procedures or user-defined table functions of the scheduler interface are allowed to execute the corresponding functionality: adding a scheduled task, removing a scheduled task, or listing the scheduled tasks or their execution status. The entire interface is configured by default with PUBLIC access rights during the installation.

Recommendations:

- Grant rights to groups or roles, rather than to individual authorization IDs.

- Restrict access to the ADMIN_TASK_ADD and ADMIN_TASK_REMOVE stored procedures to users with a business need for their use. Access to the user-defined table functions that list tasks and execution status can remain unrestricted.

The authorization ID of the DB2 thread that called the stored procedure ADMIN_TASK_ADD is passed to the scheduler and stored in the task list with the the task definition. The ADMIN_TASK_ADD stored procedure gathers the authorities granted to this authorization ID from the subsystem parameters and from the catalog table, and passes them over to the scheduler. The same mechanism is used in ADMIN_TASK_REMOVE to verify that the user is permitted to remove the task.

A task in the scheduler task list can be removed by the owner of the task, or by any user that has SYSOPR, SYSCTRL, or SYSADM privileges. The owner of a task is the CURRENT SQLID of the process at the time the task was added to the task list.

Protection of the resources of the administrative task scheduler

The administrative task scheduler task lists are protected against unauthorized use by other users than the started task execution user.

The VSAM resource (by default DSNC910.TASKLIST, where DSNC910 is the DB2 catalog prefix) that stores the administrative scheduler task list must be protected in RACF against unauthorized access. Only the administrative scheduler started task user has UPDATE authority on the VSAM resources. No other users should have any access.

A similar security concept is implemented for the SYSIBM.ADMIN_TASKS table, which stores a redundant copy of the scheduled tasks. Only the scheduler started tasks user has SELECT, INSERT, DELETE, or UPDATE authority on this resource. Users with EXECUTE rights on the user-defined functions ADMIN_TASK_LIST and ADMIN_TASK_STATUS have only SELECT authority on the table SYSIBM.ADMIN_TASKS.

Secure execution of tasks in the administrative task scheduler

The administrative task scheduler execution threads always switch to the security context of the execution user before executing a task. The user is authenticated through the use of passtickets.

The first action that is taken by the scheduler when starting a task execution is to switch its security context to the context of the execution user. The execution user can be explicitly identified by the ADMIN_TASK_ADD *user-ID* parameter or can be the default user.

If the task must run under a certain authority, including an authority different from the caller of the stored procedure, credentials are passed on to the administrative task scheduler. These credentials are not stored anywhere. They are validated by RACF to ensure that the caller of the stored procedure at the task definition time has the right to assume the desired security context. Therefore you can use a *passticket* (encrypted single-use password) in the password parameter of the ADMIN_TASK_ADD stored procedure. If no credentials are provided, then the administrative scheduler executes the tasks under its default execution user.

The scheduler generates and uses passtickets for executing the tasks under the corresponding authority. Each task executes after switching to the requested security context using the user ID and the generated passticket.

No password is stored in the task scheduler, but the scheduler is defined as a trusted program in RACF, and is allowed to get passtickets for any user. The scheduler sub-thread requires a passticket from RACF and logs in using this single-use password. The execution of the task then occurs in the switched security concept, allowing the task to have access to the resources defined for this execution user. After the execution, the security context is switched back to the scheduled task user.

The administrative scheduler started task module (DSNADMT0) uses the pthread_security_np() function to switch users. If the BPX.DAEMON facility class is active, then DSNADMT0 must be defined to RACF program control. If the BPX.DAEMON facility class is not active or if the scheduler is not defined to RACF program control, error EDC5139I is returned when trying to switch to another security level.

The scheduler resources should be protected from unintended impact when executing a task. Therefore, the scheduler started task user STARTUID, which has access to the scheduler resources, must not be used as default execution user DFLTUID, and it should not be specified in the *user-ID* parameter. The scheduler will not start if the started task user and the default execution user are identical. The default execution user should have as few rights as possible, to avoid impacting any resources if no user is defined for a task.

The default execution user has no authority except to attach to DB2 and write to the JES reader.

Execution of scheduled tasks in the administrative task scheduler

The administrative task scheduler manages the point in time, the security, the input parameters and the status of the task execution.

The scheduler work is based on scheduled tasks defined by the user. A task is mainly composed of a schedule definition and a work definition. The scheduler work is based on scheduled tasks, each scheduled task is a basic user-defined unit of work. Each task is associated with a unique task name. Up to 9999 tasks are supported in the scheduler at one time.

A scheduled task consists of a schedule and a work definition. The schedule part tells the scheduler when to execute the task. A user defines an execution window of time during which execution is permitted and either a time-based schedule or an event that triggers the execution of the job in this window. The work definition specifies what to execute, either a JCL job or a stored procedure, and the authority (user) under which to execute the task.

Multi-threading in the administrative task scheduler

The administrative task scheduler uses a pool of execution threads that allow it to execute many tasks simultaneously.

The scheduler is multi-threaded in order to be able to simultaneously execute different tasks: it starts the execution of scheduled tasks and then waits for their completion. The execution of a task is delegated by the scheduler to one of its

sub-threads that starts the execution, waits until the execution completes and gather the execution status. Each sub-thread can be used for any type of tasks.

The maximum number of sub-threads is determined by the parameter MAXTHD of the scheduler started task, which by default is 99. Thus, up to 99 tasks can be executed simultaneously by the scheduler. To reduce the memory usage of the scheduler, reduce the number of sub-threads by specifying a lower MAXTHD parameter value. The minimum permitted value for MAXTHD is 1, but should not be lower than the maximum number of tasks that you expect to execute simultaneously. If there are more tasks to be executed simultaneously than sub-threads available, some tasks will not start executing immediately. The scheduler will try to find an available sub-thread within one minute of when the task is scheduled for execution. As a result, multiple short tasks might be serialized in the same sub-thread, provided that their total execution time does not go over this minute.

If a task execution still cannot be started one minute after it should have been, the execution is skipped and the last execution status of this task is set to the NOTRUN state. The following message displays on the operator's console.

```
DSNA678I csect-name THE NUMBER OF TASKS TO BE CONCURRENTLY
EXECUTED BY THE ADMIN SCHEDULER proc-name EXCEEDS max-threads
```

If this happens, increase the MAXTHD parameter value and restart the scheduler.

Scheduled execution of a stored procedure

The scheduler gets parameter values for the stored procedure executing the given SQL select statement, then calls the stored procedure in DB2.

One execution sub-thread of the scheduler is used to execute a task that describes a stored procedure call. The execution sub-thread first connects to the DB2 member indicated in the *DB2SSID* parameter of the scheduler started task. If the connection cannot be established, the execution is skipped and the last execution status is set to the NOTRUN state.

After connection, the parameter values of the stored procedure are retrieved from DB2 using the select statement defined in the task parameter *procedure-input*. If an error occurs retrieving those parameter values, the last execution status of the task is set to the error returned by DB2, and the stored procedure itself is not called.

Otherwise, the stored procedure is called with the retrieved parameter values. The stored procedure name is concatenated from the task parameters *procedure-schema* and *procedure-name*. The SQL CALL command is synchronous and the execution thread is blocked until the stored procedure finishes execution. Then the last execution status is set to the values returned by DB2. Finally, a COMMIT statement is issued and the connection to DB2 is closed.

The execution status of DB2 stored procedures always contain null values in the JOB_ID, MAXRC, COMPLETION_TYPE, SYSTEM_ABENDCD and USER_ABENDCD fields. In the case of a DB2 error, the fields SQLCODE, SQLSTATE, SQLERRMC, and SQLERRP will contain what DB2 returned to the stored procedure.

How the administrative task scheduler works with Unicode

The administrative task scheduler is able to retrieve and pass Unicode parameters to a DB2 stored procedure.

If the stored procedure accepts Unicode parameters, or if it does not accept Unicode parameters but the retrieved parameter values do not contain any special character that cannot be expressed in EBCDIC or ASCII, no character will be broken.

However, the Unicode values must be retrieved through a select statement expressed in EBCDIC, so that special characters cannot be used in the table names or in the column names where the parameter values are retrieved.

Scheduled execution of a JCL job

The administrative task scheduler sends the JCL job to the JES reader. It can optionally wait for the job to terminate and purge it from the JES job list.

One execution sub-thread of the scheduler is used to execute a task that locates a data set containing a JCL job. The sub-thread reads the JCL job from the data set where it is stored, identified by the task parameters *JCL-library* and *JCL-member*. The data set can be sequential or partitioned. For a sequential data set, *JCL-member* is NULL.

In the case of an error, the error is written into the last execution status of the task. Otherwise, the job is submitted to the internal JES reader. According to the *job_wait* parameter, the sub-thread waits for the execution completion or not. When the sub-thread waits for completion, the last execution status includes the complete returned values provided by the JES reader. Otherwise, it contains the JCL job ID and a success message.

- If *job_wait* is set to NO, the sub-thread does not wait until the job completes execution and returns immediately after the job submission. The task execution status is set to the submission status, the result of the job execution itself will not be available.
- If *job_wait* is set to YES, the sub-thread simulates a synchronous execution of the JCL job. It waits until the job execution completes, get the job status from the JES reader and fills in the last execution status of the task.
- If *job_wait* is set toPURGE, the sub-thread purges the job output from the JES reader after execution. Execution is the same as for *job_wait*=YES.

JCL job execution status always contains a null value in the SQLCODE, SQLSTATE, SQLERRMC, and SQLERRP fields. If the job can be submitted successfully to the JES reader, the field JOB_ID contains the ID of the job in the JES reader. If the job is executed asynchronously, the MAXRC, COMPLETION_TYPE, SYSTEM_ABENDCD and USER_ABENDCD fields will also be null values, because the sub-thread does not wait for job completion before writing the status. If the job was executed synchronously, those fields contain the values returned by the JES reader.

Execution of scheduled tasks in a data sharing environment

All administrative task schedulers of the data sharing group cooperate in the execution of the scheduled tasks.

In a data sharing environment, several schedulers cooperate in the execution of the scheduled tasks. If a task has a member affinity, that is, if its parameter DB2-SSID contains the name of a DB2 member, only the scheduler that is associated with this DB2 member will execute this task. The task is executed as in a non-data sharing mode. If this scheduler is unavailable, the task will not be executed.

When a task has no member affinity, that is, if *DB2-SSID* is a null value, the scheduler that wakes first executes the task. If the task execution can complete on this scheduler, for example because its associated DB2 member is not running or because all of its execution threads are busy, other schedulers will not try executing this task. However, if the scheduler cannot start the execution, the other schedulers will try successively to start executing the task until one successes or all fail in executing the task.

The distribution of a task execution on the one or the other scheduler cannot be predicted. Successive executions of the same task can be done on different schedulers.

Chapter 17. Monitoring and controlling DB2 and its connections

The information under this heading, up to "Controlling IMS connections" on page 388, is General-use Programming Interface and Associated Guidance Information, as defined in "Notices" on page 1301.

Chapter 15, "Basic operation," on page 317 tells you how to start DB2, submit work to be processed, and stop DB2. The following operations, described in this chapter, require more understanding of what DB2 is doing:

"Controlling DB2 databases and buffer pools"
"Controlling user-defined functions" on page 373
"Controlling DB2 utilities" on page 375
"Controlling the IRLM" on page 376

This chapter also introduces the concept of a *thread*, a DB2 structure that makes the connection between another subsystem and DB2. A thread describes an application's connection, traces its progress, and delimits its accessibility to DB2 resources and services. Most DB2 functions execute under a thread structure. The use of threads in making, monitoring, and breaking connections is described in the following sections:

"Monitoring threads" on page 379
"Controlling TSO connections" on page 380
"Controlling CICS connections" on page 383
"Controlling IMS connections" on page 388
"Controlling RRS connections" on page 397
"Controlling connections to remote systems" on page 400

"Controlling traces" on page 418, tells you how to start and stop traces, and points to other books for help in analyzing their results.

"Controlling the resource limit facility (governor)" on page 421, tells how to start and stop the governor, and how to display its current status.

A final section, "Changing subsystem parameter values" on page 421, tells how to change subsystem parameters dynamically even when DB2 is running.

Important: Examples of commands in this chapter do not necessarily illustrate all the available options. For the complete syntax of any command or utility, see *DB2 Command Reference* or *DB2 Utility Guide and Reference*. For data sharing considerations, see *DB2 Data Sharing: Planning and Administration*.

Controlling DB2 databases and buffer pools

DB2 databases are controlled by the following commands:

START DATABASE
Makes a database, or individual partitions, available. Also removes pages from the logical page list (LPL). For its use, see "Starting databases" on page 364.

© Copyright IBM Corp. 1982, 2008

DISPLAY DATABASE
Displays status, user, and locking information for a database. For its use, see "Monitoring databases" on page 365.

STOP DATABASE
Makes a database, or individual partitions, unavailable after existing users have quiesced. DB2 also closes and deallocates the data sets. For its use, see "Stopping databases" on page 371.

The START and STOP DATABASE commands can be used with the SPACENAM and PART options to control table spaces, index spaces, or partitions. For example, the following command starts two partitions of table space DSN8S81E in the database DSN8D81A:

```
-START DATABASE (DSN8D81A) SPACENAM (DSN8S81E) PART (1,2)
```

Starting databases

The command START DATABASE (*) starts all databases for which you have the STARTDB privilege. The privilege can be explicitly granted, or can belong implicitly to a level of authority (DBMAINT and above, as shown in Figure 9 on page 139). The command starts the database, but not necessarily all the objects it contains. Any table spaces or index spaces in a restricted mode remain in a restricted mode and are not started.

START DATABASE (*) does not start the DB2 directory (DSNDB01), the DB2 catalog (DSNDB06), or the DB2 work file database (called DSNDB07, except in a data sharing environment). These databases have to be started explicitly using the SPACENAM option. Also, START DATABASE (*) does not start table spaces or index spaces that have been explicitly stopped by the STOP DATABASE command.

The PART keyword of the command START DATABASE can be used to start individual partitions of a table space. It can also be used to start individual partitions of a partitioning index or logical partitions of a nonpartitioning index. The started or stopped state of other partitions is unchanged.

Starting an object with a specific status

A database, table space, or an index space can be started with a specific status that limits access to it.

Status	Provides this access
RW	Read-write. This is the default value.
RO	Read only. You cannot change the data.
UT	Utility only. The object is available only to the DB2 utilities.

Databases, table spaces, and index spaces are started with RW status when they are created. You can make any of them unavailable by using the command STOP DATABASE. DB2 can also make them unavailable when it detects an error.

In cases when the object was explicitly stopped, you can make them available again using the command START DATABASE. For example, the following command starts all table spaces and index spaces in database DSN8D81A for read-only access:

```
-START DATABASE (DSN8D81A) SPACENAM(*) ACCESS(RO)
```

The system responds with this message:

```
DSN9022I - DSNTDDIS '-START DATABASE' NORMAL COMPLETION
```

Starting a table space or index space that has restrictions

DB2 can make an object unavailable for a variety of reasons. Typically, in those cases, the data is unreliable and the object needs some attention before it can be started. An example of such a restriction is when the table space is placed in *copy pending* status. That status makes a table space or partition unavailable until an image copy has been made of it.

These restrictions are a necessary part of protecting the integrity of the data. **If you start an object that has restrictions, the data in the object might not be reliable.**

However, in certain circumstances, it might be reasonable to force availability. For example, a table might contain test data whose consistency is not critical. In those cases, the objects can be started by using the ACCESS(FORCE) option of START DATABASE. For example:

```
-START DATABASE (DSN8D81A) SPACENAM (DSN8S81E) ACCESS(FORCE)
```

The command releases most restrictions for the named objects. These objects must be explicitly named in a list following the SPACENAM option.

DB2 cannot process the START DATABASE ACCESS(FORCE) request if postponed abort or indoubt URs exist. The RESTP (restart-pending) status and the AREST (advisory restart-pending) status remain in effect until either automatic backout processing completes or you perform one of the following actions:

- Issue the RECOVER POSTPONED command to complete backout activity.
- Issue the RECOVER POSTPONED CANCEL command to cancel all of the postponed abort units of recovery.
- Conditionally restart or cold start DB2.

DB2 cannot apply the START DATABASE ACCESS(FORCE) command to that object, if a utility from a previous release of DB2 places an object in one of the following restrictive states:

- UTRO (utility restrictive state, read-only access allowed)
- UTRW (utility restrictive state, read and write access allowed)
- UTUT (utility restrictive state, utility exclusive control)

To reset these restrictive states, you must start the release of DB2 that originally ran the utility and terminate the utility from that release.

For more information about resolving postponed units of recovery, see “Resolving postponed units of recovery” on page 452.

Monitoring databases

You can use the command DISPLAY DATABASE to obtain information about the status of databases and the table spaces and index spaces within each database. If applicable, the output also includes information about physical I/O errors for those objects.

Use DISPLAY DATABASE as follows:

```
-DISPLAY DATABASE (dbname)
```

This results in the following messages:

```
11:44:32 DSNT360I - ***************************************************
11:44:32 DSNT361I - *  DISPLAY DATABASE SUMMARY
11:44:32            *      report_type_list
11:44:32 DSNT360I - ***************************************************
11:44:32 DSNT362I -     DATABASE = dbname   STATUS = xx
                           DBD LENGTH = yyyy

11:44:32 DSNT397I -

NAME     TYPE PART  STATUS          PHYERRLO  PHYERRHI CATALOG  PIECE
-------- ---- ----- --------------- --------- -------- -------- -----
D1       TS         RW,UTRO
D2       TS         RW
D3       TS         STOP
D4       IX         RO
D5       IX         STOP
D6       IX         UT
LOB1     LS         RW
******* DISPLAY OF DATABASE dbname ENDED *********************
11:45:15 DSN9022I - DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
```

In the preceding messages:

- *Report_type_list* indicates which options were included when the DISPLAY DATABASE command was issued. See Chapter 2 of *DB2 Command Reference* for detailed descriptions of options.
- *dbname* is an 8-byte character string indicating the database name. The pattern-matching character, *, is allowed at the beginning, middle, and end of *dbname*.
- STATUS is a combination of one or more status codes delimited by a comma. The maximum length of the string is 17 characters. If the status exceeds 17 characters, those characters are wrapped onto the next status line. Anything that exceeds 17 characters on the second status line is truncated. See Chapter 2 of *DB2 Command Reference* for a list of status codes and their descriptions.

You can use the pattern-matching character, *, in the commands DISPLAY DATABASE, START DATABASE, and STOP DATABASE. The pattern-matching character can be used in the beginning, middle, and end of the database and table space names.

Using ONLY: The keyword ONLY can be added to the command DISPLAY DATABASE. When ONLY is specified with the DATABASE keyword but not the SPACENAM keyword, all other keywords except RESTRICT, LIMIT, and AFTER are ignored. Use DISPLAY DATABASE ONLY as follows:

```
-DISPLAY DATABASE(*S*DB*) ONLY
```

This results in the following messages:

```
11:44:32 DSNT360I - ***************************************************
11:44:32 DSNT361I - *  DISPLAY DATABASE SUMMARY
11:44:32            *      GLOBAL
11:44:32 DSNT360I - ***************************************************
11:44:32 DSNT362I -     DATABASE = DSNDB01   STATUS = RW
                           DBD LENGTH = 8066
11:44:32 DSNT360I - ***************************************************
11:44:32 DSNT362I -     DATABASE = DSNDB04  STATUS = RW
                           DBD LENGTH = 21294
11:44:32 DSNT360I - ***************************************************
11:44:32 DSNT362I -     DATABASE = DSNDB06  STATUS = RW
                           DBD LENGTH = 32985
11:45:15 DSN9022I - DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
```

In the preceding messages:

- DATABASE (*S*DB*) displays databases that begin with any letter, have the letter S followed by any letters, then the letters DB followed by any letters.
- ONLY restricts the display to databases names that fit the criteria.

See Chapter 2 of *DB2 Command Reference* for detailed descriptions of these and other options on the DISPLAY DATABASE command.

Using RESTRICT: You can use the RESTRICT option of the DISPLAY DATABASE command to limit the display to objects that are currently set to a restrictive status. You can additionally specify one or more keywords to limit the display further to include only objects that are set to a particular restrictive status. For information about resetting a restrictive status, see Appendix C of *DB2 Utility Guide and Reference*.

Using ADVISORY: You can use the ADVISORY option on the DISPLAY DATABASE command to limit the display to table spaces or indexes that require some corrective action. Use the DISPLAY DATABASE ADVISORY command without the RESTRICT option to determine when:

- An index space is in the informational copy pending (ICOPY) advisory status
- A base table space is in the auxiliary warning (AUXW) advisory status

For information about resetting an advisory status, see Appendix C of *DB2 Utility Guide and Reference*.

Using OVERVEW: To display all objects within a database, you can use the OVERVIEW option of the DISPLAY DATABASE command. This option shows each object in the database on one line, does not break down an object by partition, and does not show exception states. The OVERVIEW option displays only object types and the number of data set partitions in each object. OVERVIEW is mutually exclusive with all keywords other than SPACENAM, LIMIT, and AFTER. Use DISPLAY DATABASE OVERVIEW as follows:

```
-DISPLAY DATABASE(DB486A) SPACENAM(*) OVERVIEW
```

This results in the following messages:

```
DSNT360I = ****************************************
DSNT361I = *  DISPLAY DATABASE SUMMARY 483
           *    GLOBAL OVERVIEW
DSNT360I = ****************************************
DSNT362I =     DATABASE = DB486A STATUS = RW 485
               DBD LENGTH = 4028
DSNT397I = 486
NAME     TYPE PARTS
-------- ---- -----
TS486A   TS    0004
IX486A   IX   L0004
IX486B   IX    0004
TS486C   TS
IX486C   IX
******* DISPLAY OF DATABASE DB486A  ENDED      *********************
DSN9022I = DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
```

The display indicates that there are five objects in database DB486A, two table spaces and three indexes. Table space TS486A has four parts, and table space TS486C is nonpartitioned. Index IX486A is a nonpartitioning index for table space TS486A, and index IX486B is a partitioned index with four parts for table space TS486A. Index IX486C is a nonpartitioned index for table space TS486C.

Obtaining information about application programs

You can obtain various kinds of information about application programs using particular databases or table or index spaces with the DISPLAY DATABASE command. This section describes how you can identify who or what is using the object and what locks are being held on the objects.

Who and what is using the object? You can obtain the following information:

- The names of the application programs currently using the database or space
- The authorization IDs of the users of these application programs
- The logical unit of work IDs of the database access threads accessing data on behalf of the remote locations specified.

To obtain this information, issue a command, for example, that names partitions 2, 3, and 4 in table space TPAUGF01 in database DBAUGF01:

```
-DISPLAY DATABASE (DBAUGF01) SPACENAM (TPAUGF01) PART (2:4) USE
```

DB2 returns a list similar to this one:

```
DSNT360I : ************************************
DSNT361I : *  DISPLAY DATABASE SUMMARY
           *    GLOBAL USE
DSNT360I : ************************************
DSNT362I :     DATABASE = DBAUGF01  STATUS = RW
               DBD LENGTH = 8066
DSNT397I :
NAME     TYPE PART  STATUS            CONNID   CORRID       USERID
-------- ---- ----- ----------------- -------- ------------ --------
TPAUGF01 TS    0002 RW                BATCH    S3341209     ADMF001
    -                 MEMBER NAME V61A
TPAUGF01 TS    0003 RW                BATCH    S3341209     ADMF001
    -                 MEMBER NAME V61A
TPAUGF01 TS    0004 RW                BATCH    S3341209     ADMF001
    -                 MEMBER NAME V61A
******* DISPLAY OF DATABASE DBAUGF01 ENDED      **********************
DSN9022I : DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
```

Which programs are holding locks on the objects? To determine which application programs are currently holding locks on the database or space, issue a command like the following, which names table space TSPART in database DB01:

```
-DISPLAY DATABASE(DB01) SPACENAM(TSPART) LOCKS
```

DB2 returns a list similar to this one:

```
17:45:42 DSNT360I - ****************************************************
17:45:42 DSNT361I - *  DISPLAY DATABASE SUMMARY
17:45:42            *    GLOBAL LOCKS
17:45:42 DSNT360I - ****************************************************
17:45:42 DSNT362I -     DATABASE = DB01  STATUS = RW
17:45:42                DBD LENGTH = yyyy
17:45:42 DSNT397I -
 NAME     TYPE PART  STATUS            CONNID   CORRID       LOCKINFO
 -------- ---- ----- ----------------- -------- ------------ ---------
 TSPART   TS    0001 RW                LSS001   DSN2SQL      H-IX,P,C
 TSPART   TS    0002 RW                LSS001   DSN2SQL      H-IX,P,C
 TSPART   TS    0003 RW                LSS001   DSN2SQL      H-IX,P,C
 TSPART   TS    0004 RW                LSS001   DSN2SQL      H-IX,P,C
 ******* DISPLAY OF DATABASE DB01     ENDED      **********************
 17:45:44 DSN9022I . DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
```

For an explanation of the field LOCKINFO, see message DSNT396I in Part 2 of *DB2 Messages*.

Use the LOCKS ONLY keywords on the DISPLAY DATABASE command to display only spaces that have locks. The LOCKS keyword can be substituted with USE, CLAIMERS, LPL, or WEPR to display only databases that fit the criteria. Use DISPLAY DATABASE as follows:

```
-DISPLAY DATABASE (DSNDB06) SPACENAM(*) LOCKS ONLY
```

This results in the following messages:

```
11:44:32 DSNT360I - ***************************************************
11:44:32 DSNT361I - *  DISPLAY DATABASE SUMMARY
11:44:32            *      GLOBAL LOCKS
11:44:32 DSNT360I - ***************************************************
11:44:32 DSNT362I -     DATABASE = DSNDB06   STATUS = RW
                            DBD LENGTH = 60560
11:44:32 DSNT397I -
NAME     TYPE PART  STATUS            CONNID   CORRID       LOCKINFO
-------- ---- ----- ----------------- -------- ------------ ---------
SYSDBASE TS         RW                DSN      020.DBCMD 06 H-IS,S,C
******* DISPLAY OF DATABASE DSNDB06  ENDED      **********************
11:45:15 DSN9022I - DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
```

See Chapter 2 of *DB2 Command Reference* for detailed descriptions of these and other options of the DISPLAY DATABASE command.

Obtaining information about pages in error

There are two ways that pages can be in error:

- A page is logically in error if its problem can be fixed without redefining new disk tracks or volumes. For example, if DB2 cannot write a page to disk because of a connectivity problem, the page is logically in error. DB2 inserts entries for pages that are logically in error in a *logical page list* (LPL).
- A page is physically in error if there are physical errors, such as device errors. Such errors appear on the *write error page range* (WEPR). The range has a low and high page, which are the same if only one page has errors.

If the cause of the problem is undetermined, the error is first recorded in the LPL. If recovery from the LPL is unsuccessful, the error is then recorded on the error page range.

Write errors for large object data type (LOB) table spaces defined with LOG NO cause the unit of work to be rolled back. Because the pages are written during normal deferred write processing, they can appear in the LPL and WEPR. The LOB data pages for a LOB table space with the LOG NO attribute are not written to LPL or WEPR. The space map pages are written during normal deferred write processing and can appear in the LPL and WEPR.

A program that tries to read data from a page listed on the LPL or WEPR receives an SQLCODE for “resource unavailable”. To access the page (or pages in the error range), you must first recover the data from the existing database copy and the log.

Displaying the logical page list: You can check the existence of LPL entries by issuing the DISPLAY DATABASE command with the LPL option. The ONLY option restricts the output to objects that have LPL pages. For example:

```
-DISPLAY DATABASE(DBFW8401) SPACENAM(*) LPL ONLY
```

The following output is produced:

```
DSNT360I = ***********************************************************
DSNT361I = *  DISPLAY DATABASE SUMMARY
           *    GLOBAL LPL
DSNT360I = ***********************************************************
DSNT362I =     DATABASE = DBFW8401  STATUS = RW,LPL
               DBD LENGTH = 8066
DSNT397I =
NAME     TYPE PART  STATUS            LPL PAGES
-------- ---- ----- ----------------- ------------------
TPFW8401 TS    0001 RW,LPL            000000-000004
ICFW8401 IX   L0001 RW,LPL            000000,000003
IXFW8402 IX         RW,LPL            000000,000003-000005
----                                  000007,000008-00000B
----                                  000080-000090
******* DISPLAY OF DATABASE DBFW8401 ENDED     **********************
DSN9022I = DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
```

The display indicates that the pages listed in the LPL PAGES column are unavailable for access. For the syntax and description of DISPLAY DATABASE, see Chapter 2 of *DB2 Command Reference*.

Removing pages from the LPL: The DB2 subsystem always attempts automated recovery of LPL pages when the pages are added to the LPL. Manual recover can also be performed. When an object has pages on the LPL, there are several ways to manually remove those pages and make them available for access when DB2 is running:

- Start the object with access (RW) or (RO). That command is valid even if the table space is already started.

  When you issue the command START DATABASE, you see message DSNI006I, indicating that LPL recovery has begun. Message DSNI022I is issued periodically to give you the progress of the recovery. When recovery is complete, you see DSNI021I.

  When you issue the command START DATABASE for a LOB table space that is defined as LOG NO, and DB2 detects log records required for LPL recovery are missing due to the LOG NO attribute, the LOB table space is placed in AUXW status and the LOB is invalidated.
- Run the RECOVER or REBUILD INDEX utility on the object.

  The only exception to this is when a logical partition of a nonpartitioned index has both LPL and RECP status. If you want to recover the logical partition using REBUILD INDEX with the PART keyword, you must first use the command START DATABASE to clear the LPL pages.
- Run the LOAD utility with the REPLACE option on the object.
- Issue an SQL DROP statement for the object.

Only the following utilities can be run on an object with pages in the LPL:

LOAD with the REPLACE option
MERGECOPY
REBUILD INDEX
RECOVER, *except*:
    RECOVER...PAGE
    RECOVER...ERROR RANGE
REPAIR with the SET statement
REPORT

Displaying a write error page range: Use DISPLAY DATABASE to display the range of error pages. For example:

```
-DISPLAY DATABASE (DBPARTS) SPACENAM (TSPART01) WEPR
```

The preceding command might display the following list:

```
11:44:32 DSNT360I - ***************************************************
11:44:32 DSNT361I - *  DISPLAY DATABASE SUMMARY
11:44:32            *          GLOBAL WEPR
11:44:32 DSNT360I - ***************************************************
11:44:32 DSNT362I -     DATABASE = DBPARTS  STATUS = RW
                        DBD LENGTH = yyyy
11:44:32 DSNT397I -

NAME     TYPE PART  STATUS          PHYERRLO  PHYERRHI CATALOG  PIECE
-------- ---- ----- --------------- --------  -------- -------- -----
TSPART01 TS   0001  RW,UTRO         00000002  00000004 DSNCAT   000
TSPART01 TS   0002  RW,UTRO         00000009  00000013 DSNCAT   001
TSPART01 TS   0003  RO
TSPART01 TS   0004  STOP
TSPART01 TS   0005  UT
******* DISPLAY OF DATABASE DBPARTS ENDED       **********************
11:45:15 DSN9022I - DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
```

In the previous messages:

- PHYERRLO and PHYERRHI identify the range of pages that were being read when the I/O errors occurred. PHYERRLO is an 8-digit hexadecimal number representing the lowest page found in error, while PHYERRHI represents the highest page found in error.
- PIECE, a 3-digit integer, is a unique identifier for the data set supporting the page set that contains physical I/O errors.

For additional information about this list, see the description of message DSNT392I in Part 2 of *DB2 Messages*.

Stopping databases

Databases, table spaces, and index spaces can be made unavailable with the STOP DATABASE command. You can also use STOP DATABASE with the PART option to stop the following types of partitions:

- Physical partitions within a table space
- Physical partitions within an index space
- Logical partitions within a nonpartitioning index associated with a partitioned table space.

This prevents access to individual partitions within a table or index space while allowing access to the others. When you specify the PART option with STOP DATABASE on physically partitioned spaces, the data sets supporting the given physical partitions are closed and do not affect the remaining partitions. However, STOP DATABASE with the PART option does not close data sets associated with logically partitioned spaces. To close these data sets, you must execute STOP DATABASE without the PART option.

The AT(COMMIT) option of STOP DATABASE stops objects quickly. The AT(COMMIT) option interrupts threads that are bound with RELEASE(DEALLOCATE) and is useful when thread reuse is high.

If you specify AT(COMMIT), DB2 takes over access to an object when all jobs release their claims on it and when all utilities release their drain locks on it. If you do not specify AT(COMMIT), the objects are not stopped until all existing applications have deallocated. New transactions continue to be scheduled, but they receive SQLCODE -904 SQLSTATE '57011' (resource unavailable) on the first SQL statement that references the object or when the plan is prepared for execution. STOP DATABASE waits for a lock on an object that it is attempting to stop. If the

wait time limit for locks (15 timeouts) is exceeded, then the STOP DATABASE command terminates abnormally and leaves the object in stop pending status (STOPP).

Database DSNDB01 and table spaces DSNDB01.DBD01 and DSNDB01.SYSLGRNX must be started before stopping user-defined databases or the work file database. A DSNI003I message tells you that the command was unable to stop an object. You must resolve the problem indicated by this message and run the job again. If an object is in STOPP status, you must first issue the START DATABASE command to remove the STOPP status and then issue the STOP DATABASE command.

DB2 subsystem databases (catalog, directory, work file) can also be stopped. After the directory is stopped, installation SYSADM authority is required to restart it.

The following examples illustrate ways to use the command:

-STOP DATABASE (*)
Stops all databases for which you have STOPDB authorization, except the DB2 directory (DSNDB01), the DB2 catalog (DSNDB06), or the DB2 work file database (called DSNDB07, except in a data sharing environment), all of which must be stopped explicitly.

-STOP DATABASE (*dbname*)
Stops a database, and closes all of the data sets of the table spaces and index spaces in the database.

-STOP DATABASE (*dbname*, ...)
Stops the named databases and closes all of the table spaces and index spaces in the databases. If DSNDB01 is named in the database list, it should be last on the list because stopping the other databases requires that DSNDB01 be available.

-STOP DATABASE (*dbname*) SPACENAM (*)
Stops and closes all of the data sets of the table spaces and index spaces in the database. The status of the named database does not change.

-STOP DATABASE (*dbname*) SPACENAM (*space-name*, ...)
Stops and closes the data sets of the named table space or index space. The status of the named database does not change.

-STOP DATABASE (*dbname*) SPACENAM (*space-name*, ...) PART (*integer*)
Stops and closes the specified partition of the named table space or index space. The status of the named database does not change. If the named index space is nonpartitioned, DB2 cannot close the specified logical partition.

The data sets containing a table space are closed and deallocated by the preceding commands.

Altering buffer pools

DB2 stores buffer pool attributes in the DB2 bootstrap data set. You can use the ALTER BUFFERPOOL command to alter buffer pool attributes, which are initially defined during the DB2 installation process. These attributes include buffer pool sizes, sequential steal thresholds, deferred write thresholds, and parallel sequential thresholds. See Chapter 2 of *DB2 Command Reference* for descriptions of the options that you can use with this command. See "Tuning database buffer pools" on page 665 for guidance about using buffer pools and examples of ALTER BUFFERPOOL.

Monitoring buffer pools

Use the DISPLAY BUFFERPOOL command to display the current status for one or more active or inactive buffer pools. You can request a summary or detail report.

For example:

```
-DISPLAY BUFFERPOOL(BP0)
```

might produce a summary report such as this:

```
!DIS BUFFERPOOL(BP0)
DSNB401I ! BUFFERPOOL NAME BP0, BUFFERPOOL ID 0, USE COUNT 27
DSNB402I ! BUFFER POOL SIZE = 2000 BUFFERS
            ALLOCATED          =     2000   TO BE DELETED   =        0
            IN-USE/UPDATED     =        0
DSNB406I ! PAGE STEALING METHOD = LRU
DSNB404I ! THRESHOLDS -
           VP SEQUENTIAL       = 80
           DEFERRED WRITE      = 85   VERTICAL DEFERRED WRT  = 10,15
           PARALLEL SEQUENTIAL = 50   ASSISTING PARALLEL SEQT=  0

DSN9022I ! DSNB1CMD '-DISPLAY BUFFERPOOL' NORMAL COMPLETION
```

See Chapter 2 of *DB2 Command Reference* for descriptions of the options you can use with this command and the information you find in the summary and detail reports.

Controlling user-defined functions

User-defined functions are extensions to the SQL language. You can invoke user-defined functions in an SQL statement wherever you can use expressions or built-in functions. User-defined functions, like stored procedures, run in WLM-established address spaces. For information about creating, defining and invoking user-defined functions, see *DB2 SQL Reference*. For information about working with WLM-established address spaces, see "Monitoring and controlling stored procedures" on page 413.

DB2 user-defined functions are controlled by the following commands described in Chapter 2 of *DB2 Command Reference*:

START FUNCTION SPECIFIC
Activates an external function that is stopped.

DISPLAY FUNCTION SPECIFIC
Displays statistics about external user-defined functions accessed by DB2 applications.

STOP FUNCTION SPECIFIC
Prevents DB2 from accepting SQL statements with invocations of the specified functions.

Starting user-defined functions

The DB2 command START FUNCTION SPECIFIC activates an external function that is stopped. Built-in functions or user-defined functions that are sourced on another function cannot be started.

Use the START FUNCTION SPECIFIC command to activate all or a specific set of stopped external functions. For example, issue a command like the following, which starts functions USERFN1 and USERFN2 in the PAYROLL schema:

```
START FUNCTION SPECIFIC(PAYROLL.USERFN1,PAYROLL.USERFN2)
```

The following output is produced:

```
DSNX973I START FUNCTION SPECIFIC SUCCESSFUL FOR PAYROLL.USERFN1
DSNX973I START FUNCTION SPECIFIC SUCCESSFUL FOR PAYROLL.USERFN2
```

Monitoring user-defined functions

The DB2 command DISPLAY FUNCTION SPECIFIC displays statistics about external user-defined functions accessed by DB2 applications. This command displays one output line for each function that has been accessed by a DB2 application. Information returned by this command reflect a dynamic status. By the time DB2 displays the information, the status might have changed. Built-in functions or user-defined functions that are sourced on another function are not applicable to this command.

Use the DISPLAY FUNCTION SPECIFIC command to list the range of functions that are stopped because of a STOP FUNCTION SPECIFIC command. For example, issue a command like the following, which displays information about functions in the PAYROLL schema and the HRPROD schema:

```
-DISPLAY FUNCTION SPECIFIC(PAYROLL.*,HRPROD.*)
```

The following output is produced:

```
DSNX975I DSNX9DIS - DISPLAY FUNCTION SPECIFIC REPORT FOLLOWS-

------ SCHEMA=PAYROLL
FUNCTION           STATUS   ACTIVE  QUED  MAXQ  TIMEOUT  FAIL  WLM_ENV
PAYRFNC1
                   STARTED       0     0     1        0     0  PAYROLL
PAYRFNC2
                   STOPQUE       0     5     5        3     0  PAYROLL
PAYRFNC3
                   STARTED       2     0     6        0     0  PAYROLL
USERFNC4
                   STOPREJ       0     0     1        0     1  SANDBOX

------ SCHEMA=HRPROD
FUNCTION           STATUS   ACTIVE  QUED  MAXQ  TIMEOUT  FAIL  WLM_ENV
HRFNC1
                   STARTED       0     0     1        0     0  HRFUNCS
HRFNC2
                   STOPREJ       0     0     1        0     0  HRFUNCS

DSNX9DIS DISPLAY FUNCTION SPECIFIC REPORT COMPLETE
DSN9022I - DSNX9COM '-DISPLAY FUNC' NORMAL COMPLETION
```

Stopping user-defined functions

The DB2 command STOP FUNCTION SPECIFIC prevents DB2 from accepting SQL statements with invocations of the specified functions. This command does not prevent SQL statements with invocations of the functions from running if they have already been queued or scheduled by DB2. You cannot stop built-in functions or user-defined functions that are sourced on another function. While the STOP FUNCTION SPECIFIC command is in effect, attempts to execute the stopped functions are queued.

Use the STOP FUNCTION SPECIFIC command to stop access to all or a specific set of external functions. For example, issue a command like the following, which stops functions USERFN1 and USERFN3 in the PAYROLL schema:

```
STOP FUNCTION SPECIFIC(PAYROLL.USERFN1,PAYROLL.USERFN3)
```

The following output is produced:

```
DSNX974I STOP FUNCTION SPECIFIC SUCCESSFUL FOR PAYROLL.USERFN1
DSNX974I STOP FUNCTION SPECIFIC SUCCESSFUL FOR PAYROLL.USERFN3
```

Controlling DB2 utilities

You can run DB2 utilities against databases, table spaces, indexes, index spaces, and partitions.

DB2 utilities are classified into two groups: online and stand-alone. The online utilities require DB2 to be running and can be invoked in several different ways. The stand-alone utilities do not require DB2 to be up, and they can be invoked only by means of JCL. The online utilities are described in Part 2 of *DB2 Utility Guide and Reference*, and the stand-alone utilities are described in Part 3 of *DB2 Utility Guide and Reference*.

Starting online utilities

To start a DB2 utility, prepare an appropriate set of JCL statements for a utility job and include DB2 utility control statements in the input stream for that job. DB2 utilities can dynamically create object lists from a pattern-matching expression and can dynamically allocate the data sets required to process those objects.

Monitoring online utilities

The following commands for monitoring and changing DB2 utility jobs are described in Chapter 2 of *DB2 Command Reference*.

ALTER UTILITY
Alters parameter values of an active REORG utility.

DISPLAY UTILITY
Displays the status of utility jobs.

TERM UTILITY
Terminates a utility job before its normal completion.

If a utility is not running, you need to determine whether the type of utility access is allowed on an object of a specific status. Table 93 shows the compatibility of utility types and object status.

Table 93. Compatibility of utility types and object status

Utility type	Object access
Read-only	RO
All	RW[1]
DB2	UT

Note:

1. RW is the default access type for an object.

To change the status of an object, use the ACCESS option of the START DATABASE command to start the object with a new status. For example:

```
-START DATABASE (DSN8D61A) ACCESS(RO)
```

For more information about concurrency and compatibility of individual online utilities, see Part 2 of *DB2 Utility Guide and Reference*. For a general discussion about controlling concurrency for utilities, see Part 5 (Volume 2) of *DB2 Administration Guide*.

Stand-alone utilities

The following stand-alone utilities can be run only by means of JCL:

DSN1CHKR
DSN1COPY
DSN1COMP
DSN1PRNT
DSN1SDMP
DSN1LOGP
DSNJLOGF
DSNJU003 (change log inventory)
DSNJU004 (print log map)

Most of the stand-alone utilities can be used while DB2 is running. However, for consistency of output, the table spaces and index spaces must be stopped first because these utilities do not have access to the DB2 buffer pools. In some cases, DB2 *must* be running or stopped before you invoke the utility. See Part 3 of *DB2 Utility Guide and Reference* for detailed environmental information about these utilities.

Stand-alone utility job streams require that you code specific data set names in the JCL. To determine the fifth qualifier in the data set name, you need to query the DB2 catalog tables SYSIBM.SYSTABLEPART and SYSIBM.SYSINDEXPART to determine the IPREFIX column that corresponds to the required data set.

The *change log inventory utility* (DSNJU003) enables you to change the contents of the bootstrap data set (BSDS). This utility cannot be run while DB2 is running because inconsistencies could result. Use STOP DB2 MODE(QUIESCE) to stop the DB2 subsystem, run the utility, and then restart DB2 with the START DB2 command.

The *print log map utility* (DSNJU004) enables you to print the the bootstrap data set contents. The utility can be run when DB2 is active or inactive; however, when it is run with DB2 active, the user's JCL and the DB2 started task must both specify DISP=SHR for the BSDS data sets.

Controlling the IRLM

The internal resource lock manager (IRLM) subsystem manages DB2 locks. The particular IRLM to which DB2 is connected is specified in DB2's load module for subsystem parameters. It is also identified as a z/OS subsystem in the SYS1.PARMLIB member IEFSSN*xx*. That name is used as the IRLM procedure name (*irlmproc*) in z/OS commands.

IMS and DB2 must use separate instances of IRLM.

Data sharing: In a data sharing environment, the IRLM handles global locking, and each DB2 member has its own corresponding IRLM. See *DB2 Data Sharing: Planning and Administration* for more information about configuring IRLM in a data sharing environment.

You can use the following z/OS commands to control the IRLM. *irlmproc* is the IRLM procedure name, and *irlmnm* is the IRLM subsystem name. See Chapter 2 of *DB2 Command Reference* for more information about these commands.

MODIFY *irlmproc*,**ABEND,DUMP**
Abends the IRLM and generates a dump.

MODIFY *irlmproc*,**ABEND,NODUMP**
Abends the IRLM but does not generate a dump.

MODIFY *irlmproc*,**DIAG**
Initiates diagnostic dumps for IRLM subsystems in a data sharing group when there is a delay.

MODIFY *irlmproc*,**SET**
Sets dynamically the maximum amount of PVT storage or the number of trace buffers used for this IRLM.

MODIFY *irlmproc*,**STATUS**
Displays the status for the subsystems on this IRLM.

START *irlmproc*
Starts the IRLM.

STOP *irlmproc*
Stops the IRLM normally.

TRACE CT,OFF,COMP=*irlmnm*
Stops IRLM tracing.

TRACE CT,ON,COMP=*irlmnm*
Starts IRLM tracing for all subtypes (DBM,SLM,XIT,XCF).

TRACE CT,ON,COMP=*irlmnm*,**SUB=(***subname***)**
Starts IRLM tracing for a single subtype.

Starting the IRLM

The IRLM must be available when you start DB2.

When DB2 is installed, you normally specify that the IRLM be started automatically. Then, if the IRLM is not available when DB2 is started, DB2 starts it, and periodically checks whether it is up before attempting to connect. If the attempt to start the IRLM fails, DB2 terminates.

If an automatic IRLM start has not been specified, start the IRLM before starting DB2, using the z/OS START *irlmproc* command.

When started, the IRLM issues this message to the z/OS console:

```
DXR117I irlmnm INITIALIZATION COMPLETE
```

Consider starting the IRLM manually if you are having problems starting DB2 for either of these reasons:
- An IDENTIFY or CONNECT to a data sharing group fails.
- DB2 experiences a failure that involves the IRLM.

When you start the IRLM manually, you can generate a dump to collect diagnostic information because IRLM does not stop automatically.

Modifying the IRLM

You can use the following z/OS commands to modify the IRLM. See Chapter 2 of *DB2 Command Reference* for more information about these commands.

MODIFY *irlmproc*,**SET,PVT=***nnn*
Sets the maximum amount of PVT storage that this IRLM can use for lock control structures.

MODIFY *irlmproc*,**SET,DEADLOCK=***nnnn*
Sets the time for the local deadlock detection cycle.

MODIFY *irlmproc*,**SET,LTE=***nnnn*
Sets the number of LOCK HASH entries that this IRLM can use on the next connect to the XCF LOCK structure. Use only for data sharing.

MODIFY *irlmproc*,**SET,TIMEOUT=***nnnn,subsystem-name*
Sets the timeout value for the specified DB2 subsystem. Display the *subsystem-name* by using MODIFY *irlmproc*,STATUS.

MODIFY *irlmproc*,**SET,TRACE=***nnn*
Sets the maximum number of trace buffers used for this IRLM.

Monitoring the IRLM connection

You can use the following z/OS commands to monitor the IRLM connection. See Chapter 2 of *DB2 Command Reference* for more information about these commands.

MODIFY *irlmproc*,**STATUS,***irlmnm*
Displays the status of a specific IRLM.

MODIFY *irlmproc*,**STATUS,ALLD**
Displays the status of all subsystems known to this IRLM in the data sharing group.

MODIFY *irlmproc*,**STATUS,ALLI**
Displays the status of all IRLMs known to this IRLM in the data sharing group.

MODIFY *irlmproc*,**STATUS,MAINT**
Displays the maintenance levels of IRLM load module CSECTs for the specified IRLM instance.

MODIFY *irlmproc*,**STATUS,STOR**
Displays the current and *high water* allocation for PVT storage, as well as storage that is above the 2 GB bar.

MODIFY *irlmproc*,**STATUS,TRACE**
Displays information about trace types of IRLM subcomponents.

Stopping the IRLM

If the IRLM is started automatically by DB2, it stops automatically when DB2 is stopped. If the IRLM is not started automatically, you must stop it after DB2 stops.

If you try to stop the IRLM while DB2 or IMS is still using it, you get the following message:

```
DXR105E irlmnm STOP COMMAND REJECTED. AN IDENTIFIED SUBSYSTEM
IS STILL ACTIVE
```

If that happens, issue the STOP *irlmproc* command again, when the subsystems are finished with the IRLM.

Alternatively, if you must stop the IRLM immediately, enter the following command to force the stop:

```
MODIFY irlmproc,ABEND,NODUMP
```

The system responds with this message:

```
DXR165I KRLM TERMINATED VIA IRLM MODIFY COMMAND.
DXR121I KRLM END-OF-TASK CLEANUP SUCCESSFUL - HI-CSA 335K
-  HI-ACCT-CSA      0K
```

DB2 abends. An IMS subsystem using the IRLM does not abend and can be reconnected.

IRLM uses the z/OS Automatic Restart Manager (ARM) services. However, it de-registers from ARM for normal shutdowns. IRLM registers with ARM during initialization and provides ARM with an event exit. The event exit must be in the link list. It is part of the IRLM DXRRL183 load module. The event exit will make sure that the IRLM name is defined to z/OS when ARM restarts IRLM on a target z/OS that is different from the failing z/OS. The IRLM element name used for the ARM registration depends on the IRLM mode. For local mode IRLM, the element name is a concatenation of the IRLM subsystem name and the IRLM ID. For global mode IRLM, the element name is a concatenation of the IRLM data sharing group name, IRLM subsystem name, and the IRLM ID.

IRLM de-registers from ARM when one of the following events occurs:
- PURGE *irlmproc* is issued.
- MODIFY *irlmproc*,ABEND,NODUMP is issued.
- DB2 automatically stops IRLM.

The command MODIFY *irlmproc*,ABEND,NODUMP specifies that IRLM de-register from ARM before terminating, which prevents ARM from restarting IRLM. However, it does not prevent ARM from restarting DB2, and, if you set the automatic restart manager to restart IRLM, DB2 automatically starts IRLM.

Monitoring threads

The DB2 command DISPLAY THREAD displays current information about the status of threads, including information about:
- Threads that are processing locally
- Threads that are processing distributed requests
- Stored procedures or user-defined functions if the thread is executing one of those
- Parallel tasks

Threads can be active or pooled:
- An active allied thread is a thread that is connected to DB2 from TSO, BATCH, IMS, CICS, CAF or RRSAF.
- An active database access thread is a thread connected through a network with another system and performing work on behalf of that system.
- A pooled database access thread is an idle thread that is waiting for a new unit of work from a connection to another system to begin. Pooled threads hold no database locks.

The output of the command DISPLAY THREAD can also indicate that a system quiesce is in effect as a result of the ARCHIVE LOG command. For more information, see “Archiving the log” on page 430.

The command DISPLAY THREAD allows you to select which type of information you wish to include in the display using one or more of the following standards:
- Active, indoubt, postponed abort, or pooled threads

- Allied threads associated with the address spaces whose connection-names are specified
- Allied threads
- Distributed threads
- Distributed threads associated with a specific remote location
- Detailed information about connections with remote locations
- A specific logical unit of work ID (LUWID).

The information returned by the DISPLAY THREAD command reflects a dynamic status. By the time the information is displayed, it is possible that the status could have changed. Moreover, the information is consistent only within one address space and is not necessarily consistent across all address spaces.

To use the TYPE, LOCATION, DETAIL, and LUWID keywords, you must have SYSOPR authority or higher. For detailed information, see Chapter 2 of *DB2 Command Reference*.

Example: The DISPLAY THREAD command displays information about active and pooled threads in the following format:

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS:
DSNV402I - ACTIVE THREADS:
  NAME   ST  A     REQ       ID      AUTHID    PLAN     ASID    TOKEN
conn-name s  *   req-ct   corr-id    auth-id   pname    asid    token
conn-name s  *   req-ct   corr-id    auth-id   pname    asid    token
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I - module_name '-DISPLAY THREAD' NORMAL COMPLETION
```

Example: The DISPLAY THREAD command displays information about indoubt threads in the following format:

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV406I - INDOUBT THREADS -
COORDINATOR              STATUS      RESET  URID         AUTHID
coordinator-name         status      yes/no urid         authid
DISPLAY INDOUBT REPORT COMPLETE
DSN9022I - DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
```

Example: The DISPLAY THREAD command displays information about postponed aborted threads in the following format:

```
DSNV401I ! DISPLAY THREAD REPORT FOLLOWS -
DSNV431I ! POSTPONED ABORT THREADS -
COORDINATOR              STATUS      RESET URID         AUTHID
coordinator-name         ABORT-P           uridauthid
DISPLAY POSTPONED ABORT REPORT COMPLETE
DSN9022I ! DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
```

More information about how to interpret this output can be found in the sections describing the individual connections and in the description of message DSNV408I in Part 2 of *DB2 Messages*.

Controlling TSO connections

z/OS provides no commands for controlling or monitoring a connection to DB2. The connection is monitored instead by the DB2 command DISPLAY THREAD, which displays information about connections to DB2 (from other subsystems as well as from z/OS).

The command is generally entered from an z/OS console or an administrator's TSO session. See "Monitoring threads" on page 379 for more examples of its use.

Connecting to DB2 from TSO

The z/OS operator is not involved in starting and stopping TSO connections. Those connections are made through the DSN command processor, which is invoked either

- Explicitly, by the DSN command, or
- Implicitly, through DB2I (DB2 Interactive)

When a DSN session is active, you can enter DSN subcommands, DB2 commands, and TSO commands, as described under "Running TSO application programs" on page 327.

The DSN command can be given in the foreground or background, when running under the TSO terminal monitor program (TMP). The full syntax of the command is:

```
DSN SYSTEM (subsystemid) RETRY (n1) TEST (n2)
```

The parameters are optional, and have the following meanings:

subsystemid
: Is the subsystem ID of the DB2 subsystem to be connected

n1
: Is the number of times to attempt the connection if DB2 is not running (one attempt every 30 seconds)

n2
: Is the DSN tracing system control that can be used if a problem is suspected

For example, this invokes a DSN session, requesting 5 retries at 30-second intervals:

```
DSN SYSTEM (DB2) RETRY (5)
```

DB2I invokes a DSN session when you select any of these operations:

- SQL statements using SPUFI
- DCLGEN
- BIND/REBIND/FREE
- RUN
- DB2 commands
- Program preparation and execution

In carrying out those operations, the DB2I panels invoke CLISTs, which start the DSN session and invoke appropriate subcommands.

Monitoring TSO and CAF connections

To display information about connections that use the TSO attach facility and call attach facility (CAF), issue the command DISPLAY THREAD. Table 94 summarizes how DISPLAY THREAD output differs for a TSO online application, a TSO batch application, a QMF session, and a call attach facility application.

Table 94. Differences in display thread information for TSO and batch

Connection	Name	AUTHID	Corr-ID[1]	Plan[1]
DSN (TSO Online)	TSO	Logon ID	Logon ID	RUN .. Plan(x)
DSN (TSO Batch)	BATCH	Job USER=	Job Name	RUN .. Plan(x)
QMF	DB2CALL	Logon ID	Logon ID	'QMFvr0'
CAF	DB2CALL	Logon ID	Logon ID	OPEN parm

Table 94. Differences in display thread information for TSO and batch (continued)

Connection	Name	AUTHID	Corr-ID[1]	Plan[1]

Notes:

1. After the application has connected to DB2 but before a plan has been allocated, this field is blank.

The name of the connection can have one of the following values:

Name	Connection to
TSO	Program running in TSO foreground
BATCH	Program running in TSO background
DB2CALL	Program using the call attachment facility and running in the same address space as a program using the TSO attachment facility

The correlation ID, *corr-id*, is either the foreground authorization ID or the background job name. For a complete description of the DISPLAY THREAD status information displayed, see the description of message DSNV404I in Part 2 of *DB2 Messages*.

The following command displays information about TSO and CAF threads, including those processing requests to or from remote locations:

```
-DISPLAY THREAD(BATCH,TSO,DB2CALL)
```

```
 DSNV401I = DISPLAY THREAD REPORT FOLLOWS -
 DSNV402I = ACTIVE THREADS -
 NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
1 BATCH    T  *  2997 TEP2         SYSADM   DSNTEP41 0019 18818
2 BATCH    RA *  1246 BINETEP2     SYSADM   DSNTEP44 0022 20556
 V445-DB2NET.LUND1.AB0C8FB44C4D=20556 ACCESSING DATA FOR SAN_JOSE
3 TSO      T       12 SYSADM       SYSADM   DSNESPRR 0028  5570
4 DB2CALL  T  * 18472 CAFCOB2      SYSADM   CAFCOB2  001A 24979
5 BATCH    T  *     1 PUPPY        SYSADM   DSNTEP51 0025 20499
6          PT *   641 PUPPY        SYSADM   DSNTEP51 002D 20500
7          PT *   592 PUPPY        SYSADM   DSNTEP51 002D 20501
 DISPLAY ACTIVE REPORT COMPLETE
 DSN9022I = DSNVDT '-DIS THREAD' NORMAL COMPLETION
```

Key:

1. This is a TSO batch application.
2. This is a TSO batch application running at a remote location and accessing tables at this location.
3. This is a TSO online application.
4. This is a call attachment facility application.
5. This is an originating thread for a TSO batch application.
6. This is a parallel thread for the originating TSO batch application thread.
7. This is a parallel thread for the originating TSO batch application thread.

Figure 27. Display thread showing TSO and CAF connections

Detailed information for assisting the console operator in identifying threads involved in distributed processing can be found in “Monitoring threads” on page 379.

Disconnecting from DB2 while under TSO

The connection to DB2 ends, and the thread is terminated, when:

- You enter the END subcommand.
- You enter DSN again. (A new connection is established immediately.)
- You enter the CANCEL THREAD command (for threads that are active or suspended in DB2).
- You enter the MVS CANCEL command.
- You press the attention key (PA1).
- Any of the following operations end:
  - SQL statements using SPUFI
  - DCLGEN
  - BIND/REBIND/FREE
  - RUN
- You are using any of the preceding operations and you enter END or RETURN.

A simple session: For example, the following command and subcommands establish a connection to DB2, run a program, and terminate the connection:

TSO displays:

```
READY
```

You enter:

```
DSN SYSTEM (DSN)
```

DSN displays:

```
DSN
```

You enter:

```
RUN PROGRAM (MYPROG)
```

DSN displays:

```
DSN
```

You enter:

```
END
```

TSO displays:

```
READY
```

Controlling CICS connections

The following CICS attachment facility commands can be entered from a CICS terminal to control and monitor connections between CICS and DB2:

DSNC DISCONNECT
Terminates threads using a specific DB2 plan.

DSNC DISPLAY
Displays thread information or statistics.

DSNC MODIFY
Modifies the maximum number of threads for a transaction or group.

DSNC STOP
Disconnects CICS from DB2.

DSNC STRT
Starts the CICS attachment facility.

CICS command responses are sent to the terminal from which the corresponding command was entered, unless the DSNC DISPLAY command specifies an alternative destination. For details on specifying alternate destinations for output, see the DSNC DISPLAY command in the *DB2 Command Reference*.

For detailed information about controlling CICS connections, see "*Defining the CICS DB2 connection*" in the CICS DB2 Guide.

Connecting from CICS

A connection to DB2 can be started or restarted at any time after CICS initialization. The CICS attachment facility is a set of modules DB2 provides that are loaded into the CICS address space. Use the following command to start the attachment facility:

```
DSNC STRT ssid
```

ssid specifies a DB2 subsystem ID to override that specified in the CICS INITPARM macro.

You can also start the attachment facility automatically at CICS initialization using a program list table (PLT). For details, see Part 2 of *DB2 Installation Guide*.

Restarting CICS

One function of the CICS attachment facility is to keep data in synchronization between the two systems. If DB2 completes phase 1 but does not start phase 2 of the commit process, the units of recovery being committed are termed *indoubt*. An indoubt unit of recovery might occur if DB2 terminates abnormally after completing phase 1 of the commit process. CICS might commit or roll back work without DB2's knowledge.

DB2 cannot resolve those indoubt units of recovery (that is, commit or roll back the changes made to DB2 resources) until the connection to CICS is restarted. This means that CICS should always be auto-started (START=AUTO in the DFHSIT table) to get all necessary information for indoubt thread resolution available from its log. Avoid cold starting. The START option can be specified in the DFHSIT table, as described in *CICS Transaction Server for z/OS Resource Definition Guide*.

If there are CICS requests active in DB2 when a DB2 connection terminates, the corresponding CICS tasks might remain suspended even after CICS is reconnected to DB2. You should purge those tasks from CICS using a CICS-supplied transaction such as:

```
CEMT SET TASK(nn) FORCE
```

See *CICS Transaction Server for z/OS CICS Supplied Transactions* for more information about transactions that CICS supplies.

If any unit of work is indoubt when the failure occurs, the CICS attachment facility automatically attempts to resolve the unit of work when CICS is reconnected to DB2. Under some circumstances, however, CICS cannot resolve indoubt units of recovery. You must manually recover these indoubt units of recovery (see “Recovering indoubt units of recovery manually” on page 385 for more information).

Displaying indoubt units of recovery

To display a list of indoubt units of recovery, issue the command:

```
-DISPLAY THREAD (connection-name) TYPE (INDOUBT)
```

The command produces messages similar to these:

```
DSNV407I -STR INDOUBT THREADS - 480
COORDINATOR              STATUS      RESET URID           AUTHID
CICS41                   INDOUBT         00019B8ADE9E   ADMF001
 V449-HAS NID= CICS41.AACC9B739F125184 AND ID=GT00LE39
DISPLAY INDOUBT REPORT COMPLETE
DSN9022I -STR DSNVDT '-DIS THD' NORMAL COMPLETION
```

For an explanation of the displayed list, see the description of message DSNV408I in Part 2 of *DB2 Messages*.

Recovering indoubt units of recovery manually

To recover an indoubt unit of recovery, issue one of the following commands:

```
-RECOVER INDOUBT (connection-name) ACTION (COMMIT) ID (correlation-id)
-RECOVER INDOUBT (connection-name) ACTION (ABORT) ID (correlation-id)
```

The default value for *connection-name* is the connection name from which you entered the command. *Correlation-id* is the correlation ID of the thread to be recovered. It can be determined by issuing the command DISPLAY THREAD. Your choice for the ACTION parameter tells whether to commit or roll back the associated unit of recovery. For more details, see “Resolving indoubt units of recovery” on page 459.

One of the following messages can be used after you use the RECOVER command:

```
DSNV414I - THREAD correlation-id COMMIT SCHEDULED
DSNV415I - THREAD correlation-id ABORT SCHEDULED
```

For more information about manually resolving indoubt units of recovery, see “Manually recovering CICS indoubt units of recovery” on page 527. For information about the two-phase commit process, as well as indoubt units of recovery, see “Multiple system consistency” on page 455.

Displaying postponed units of recovery

To display a list of postponed units of recovery, issue the command:

```
-DISPLAY THREAD (connection-name) TYPE (POSTPONED)
```

The command produces messages similar to these:

```
DSNV431I -POSTPONED ABORT THREADS - 480
COORDINATOR              STATUS      RESET URID           AUTHID
CICS41                   P-ABORT         00019B8ADE9E   ADMF001
V449-HAS NID= CICS41.AACC9B739F125184 AND ID=GT00LE39
DISPLAY POSTPONED ABORT REPORT COMPLETE
DSN9022I -STR DSNVDT '-DIS THD' NORMAL COMPLETION
```

For an explanation of the displayed list, see the description of message DSNV408I in Part 2 of *DB2 Messages*.

Controlling CICS connections

This section describes how CICS threads are defined, how you can monitor those threads, and ways you can disconnect those threads.

Defining CICS threads

Every CICS transaction that accesses DB2 requires a thread to service the DB2 requests. Each thread uses one z/OS subtask to execute DB2 code for the CICS application.

The DSNC STRT command starts the CICS DB2 attachment facility, which allows CICS application programs to access DB2 databases.

Threads are created at the first DB2 request from the application if there is not one already available for the specific DB2 plan.

For complete information about defining CICS threads with DB2, see *CICS DB2 Guide*.

Monitoring the threads

No operator intervention is required for connecting applications; CICS handles the threads dynamically. You can monitor threads using CICS attachment facility commands or DB2 commands.

Using CICS attachment facility commands: Any authorized CICS user can monitor the threads and change the connection parameters as needed. Operators can use the following CICS attachment facility commands to monitor one of the following the threads:

```
DSNC DISPLAY PLAN plan-name destination
DSNC DISPLAY TRANSACTION transaction-id destination
```

These commands display the threads that the resource or transaction is using. The following information is provided for each created thread:

- Authorization ID for the plan associated with the transaction (8 characters).
- PLAN/TRAN name (8 characters).
- A or I (1 character).

  If **A** is displayed, the thread is within a unit of work. If **I** is displayed, the thread is waiting for a unit of work, and the authorization ID is blank.

The following CICS attachment facility command is used to monitor CICS :

```
DSNC DISPLAY STATISTICS distination
```

Disconnecting applications

There is no way to disconnect a particular CICS transaction from DB2 without abending the transaction. Two ways to disconnect an application are described here:

- The DB2 command CANCEL THREAD can be used to cancel a particular thread. CANCEL THREAD requires that you know the *token* for any thread you want to cancel. Enter the following command to cancel the thread identified by the token indicated in the display output.

  ```
  -CANCEL THREAD(46)
  ```

  When you issue CANCEL THREAD for a thread, that thread is scheduled to be terminated in DB2.
- The command DSNC DISCONNECT terminates the threads allocated to a plan ID, but it does not prevent new threads from being created. This command frees DB2 resources shared by the CICS transactions and allows exclusive access to them for special-purpose processes such as utilities or data definition statements.

  The thread is not canceled until the application releases it for reuse, either at SYNCPOINT or end-of-task.

For complete information about the use of CICS attachment commands with DB2, see *CICS DB2 Guide*.

Disconnecting from CICS

This section describes how to do both an orderly and forced disconnection of the attachment to CICS.

Orderly termination

It is recommended that you do orderly termination whenever possible. An orderly termination of the connection allows each CICS transaction to terminate before thread subtasks are detached. This means there should be no indoubt units of recovery at reconnection time. An orderly termination occurs when you:

- Enter the DSNC STOP QUIESCE command. CICS and DB2 remain active.
- Enter the CICS command CEMT PERFORM SHUTDOWN, and the CICS attachment facility is also named to shut down during program list table (PLT) processing. DB2 remains active. For information about the CEMT PERFORM SHUTDOWN command, see *CICS for MVS/ESA CICS-Supplied Transactions*.
- Enter the DB2 command CANCEL THREAD. The thread is abended.

The following example stops the DB2 subsystem (QUIESCE). allows the currently identified tasks to continue normal execution, and does not allow new tasks to identify themselves to DB2:

```
-STOP DB2 MODE (QUIESCE)
```

This message appears when the stop process starts and frees the entering terminal (option QUIESCE):

```
DSNC012I THE ATTACHMENT FACILITY STOP QUIESCE IS PROCEEDING
```

When the stop process ends and the connection is terminated, this message is added to the output from the CICS job:

```
DSNC025I THE ATTACHMENT FACILITY IS INACTIVE
```

Forced termination

Although it is not recommended, there might be times when it is necessary to force the connection to end. A forced termination of the connection can abend CICS transactions connected to DB2. Therefore, indoubt units of recovery can exist at reconnect. A forced termination occurs in the following situations:

- You enter the DSNC STOP FORCE command. This command waits 15 seconds before detaching the thread subtasks, and, in some cases, can achieve an orderly termination. DB2 and CICS remain active.
- You enter the CICS command CEMT PERFORM SHUTDOWN IMMEDIATE. For information about this command, see *CICS for MVS/ESA CICS-Supplied Transactions*. DB2 remains active.
- You enter the DB2 command STOP DB2 MODE (FORCE). CICS remains active.
- A DB2 abend occurs. CICS remains active.
- A CICS abend occurs. DB2 remains active.
- STOP is issued to the DB2 or CICS attachment facility, and the CICS transaction overflows to the pool. The transaction isssues an intermediate commit. The thread is terminated at commit time, and further DB2 access is not allowed.

This message appears when the stop process starts and frees the entering terminal (option FORCE):

```
DSNC022I THE ATTACHMENT FACILITY STOP FORCE IS PROCEEDING
```

When the stop process ends and the connection is terminated, this message is added to the output from the CICS job:

```
DSNC025I THE ATTACHMENT FACILITY IS INACTIVE
```

Controlling IMS connections

IMS provides these operator commands for controlling and monitoring the connection to DB2:

/START SUBSYS
Connects the IMS control region to a DB2 subsystem.

/TRACE
Controls the IMS trace.

/DISPLAY SUBSYS
Displays connection status and thread activity.

/DISPLAY OASN SUBSYS
Displays outstanding units of recovery.

/CHANGE SUBSYS
Deletes an indoubt unit of recovery from IMS.

/STOP SUBSYS
Disconnects IMS from a DB2 subsystem.

For more information about those commands, see Chapter 2 of *DB2 Command Reference* or, in the IMS library, to *IMS Command Reference*.

IMS command responses are sent to the terminal from which the corresponding command was entered. Authorization to enter IMS commands is based on IMS security.

Connecting to the IMS control region

IMS makes one connection to its control region from each DB2 subsystem. IMS can make the connection either:

- Automatically during IMS cold start initialization or at warm start of IMS if a DB2 connection was active when IMS is shut down
- In response to the command /START SUBSYS *ssid*, where *ssid* is the DB2 subsystem identifier

  The command causes the following message to be displayed at the logical terminal (LTERM):

  ```
  DFS058   START COMMAND COMPLETED
  ```

  The message is issued regardless of whether DB2 is active and does not imply that the connection is established.

The order of starting IMS and DB2 is not vital. If IMS is started first, then when DB2 comes up, it posts the control region modify task, and IMS again tries to reconnect.

If DB2 is stopped by the STOP DB2 command, the /STOP SUBSYS command, or a DB2 abend, then IMS cannot reconnect automatically. You must make the connection by using the /START command.

The following messages can be produced when IMS attempts to connect a DB2 subsystem:

- If DB2 is active, these messages are sent:
  - To the z/OS console:

    ```
    DFS3613I ESS TCB INITIALIZATION COMPLETE
    ```

  - To the IMS master terminal:

    ```
    DSNM001I IMS/TM imsid CONNECTED TO SUBSYSTEM ssnm
    ```

- If DB2 is not active, this message is sent to the master terminal:

  ```
  DSNM003I IMS/TM imsid FAILED TO CONNECT TO SUBSYSTEM ssnm
           RC=00   imsid
  ```

  imsid is the IMS connection name. RC=00 means that a notify request has been queued. When DB2 starts, IMS is also notified.

  No message goes to the z/OS console.

Thread attachment

Execution of the program's first SQL statement causes the IMS attachment facility to create a thread and allocate a plan, whose name is associated with the IMS application program module name. DB2 sets up control blocks for the thread and loads the plan.

Using the DB2 command DISPLAY THREAD: The DB2 command DISPLAY THREAD can be used to display IMS attachment facility threads.

DISPLAY THREAD output for DB2 connections to IMS differs depending on whether DB2 is connected to a DL/I batch program, a control region, a message-driven program, or a nonmessage-driven program. Table 95 summarizes these differences.

Table 95. Differences in DISPLAY THREAD information for IMS connections

Connection	Name	AUTHID[2]	ID[1,2]	Plan[1,2]
DL/I Batch	DDITV02 statement	JOBUSER=	Job Name	DDITV02 statement
Control Region	IMSID	N/A	N/A	N/A
Message Driven	IMSID	Signon ID or ltermid	PST+ PSB	RTT or program
Non-message Driven	IMSID	AXBUSER or PSBNAME	PST+ PSB	RTT or program

Notes:

1. After the application has connected to DB2 but before sign-on processing has completed, this field is blank.
2. After sign-on processing has completed but before a plan has been allocated, this field is blank.

The following command displays information about IMS threads, including those accessing data at remote locations:

```
-DISPLAY THREAD(imsid)
```

```
DSNV401I -STR DISPLAY THREAD REPORT FOLLOWS -
DSNV402I -STR ACTIVE THREADS -
NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
1 SYS3     T  *     3 0002BMP255   ADMF001  PROGHR1  0019    99
           SYS3     T  *     4 0001BMP255   ADMF001  PROGHR2  0018    97
2 SYS3     N        5              SYSADM            0065     0
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I -STR DSNVDT '-DIS THD' NORMAL COMPLETION
```

Key:

1 This is a message-driven BMP.

2 This thread has completed sign-on processing, but a DB2 plan has not been allocated.

Figure 28. DISPLAY THREAD output showing IMS connections

Thread termination

When an application terminates, IMS invokes an exit routine to disconnect the application from DB2. There is no way to terminate a thread without abending the IMS application with which it is associated. Two ways of terminating an IMS application are described here:

- Termination of the application

  The IMS commands /STOP REGION *reg#* ABDUMP or /STOP REGION *reg#* CANCEL can be used to terminate an application running in an online environment. For an application running in the DL/I batch environment, the z/OS command CANCEL can be used. See *IMS Command Reference* for more information about terminating IMS applications.
- Use of the DB2 command CANCEL THREAD

  CANCEL THREAD can be used to cancel a particular thread or set of threads. CANCEL THREAD requires that you know the *token* for any thread you want to cancel. Enter the following command to cancel the thread identified by a token in the display output:

  ```
  -CANCEL THREAD(46)
  ```

  When you issue CANCEL THREAD for a thread, that thread is scheduled to be terminated in DB2.

Displaying indoubt units of recovery

General-use Programming Interface

One function of the thread connecting DB2 to IMS is to keep data in synchronization between the two systems. If the application program requires it, a change to IMS data must also be made to DB2 data. If DB2 abends while connected to IMS, it is possible for IMS to commit or back out work without DB2 being aware of it. When DB2 restarts, that work is termed *indoubt*. Typically, some decision must be made about the status of the work.

The subject of indoubt units of recovery is treated in detail in Chapter 19, “Restarting DB2 after termination,” on page 443.

To display a list of indoubt units of recovery, issue the command:

```
-DISPLAY THREAD (imsid) TYPE (INDOUBT)
```

The command produces messages similar to these:

```
DSNV401I -STR DISPLAY THREAD REPORT FOLLOWS -
DSNV406I -STR POSTPONED ABORTT THREADS - 920
COORDINATOR               STATUS      RESET URID         AUTHID
SYS3                      P-ABORT           00017854FF6B ADMF001
V449-HAS NID= SYS3.400000000 AND ID= 0001BMP255
BATCH                     P-ABORT           00017854A8A0 ADMF001
V449-HAS NID= DSN:0001.0 AND ID= RUNP10
BATCH                     P-ABORT           00017854AA2E ADMF001
V449-HAS NID= DSN:0002.0 AND ID= RUNP90
BATCH                     P-ABORT           0001785CD711 ADMF001
V449-HAS NID= DSN:0004.0 AND ID= RUNP12
DISPLAY POSTPONED ABORT REPORT COMPLETE
DSN9022I -STR DSNVDT '-DIS THD' NORMAL COMPLETION
```

For an explanation of the list displayed, see the description of message DSNV408I in Part 2 of *DB2 Messages*.

End of General-use Programming Interface

Recovering indoubt units of recovery

General-use Programming Interface

To recover an indoubt unit, issue one of the following commands:

```
-RECOVER INDOUBT (imsid) ACTION (COMMIT) ID (pst#.psbname)
-RECOVER INDOUBT (imsid) ACTION (ABORT) ID (pst#.psbname)
```

Here *imsid* is the connection name and *pst#.psbname* is the correlation ID listed by the command DISPLAY THREAD. Your choice of the ACTION parameter tells whether to commit or roll back the associated unit of recovery. For more details, see "Resolving indoubt units of recovery" on page 459.

One of the following messages can be used after you issue the RECOVER command:

```
DSNV414I - THREAD pst#.psbname COMMIT SCHEDULED
DSNV415I - THREAD pst#.psbname ABORT SCHEDULED
```

End of General-use Programming Interface

Displaying postponed units of recovery

General-use Programming Interface

The subject of postponed units of recovery is explained in detail in Chapter 19, "Restarting DB2 after termination," on page 443. This chapter describes the operational steps that are used to list and recover postponed units in relatively simple cases.

To display a list of postponed units of recovery, issue the command:

```
-DISPLAY THREAD (imsid) TYPE (POSTPONED)
```

The command produces messages similar to these:

```
DSNV401I -STR DISPLAY THREAD REPORT FOLLOWS -
DSNV406I -STR POSTPONED ABORTT THREADS - 920
COORDINATOR               STATUS      RESET URID         AUTHID
SYS3                      P-ABORT           00017854FF6B ADMF001
V449-HAS NID= SYS3.400000000 AND ID= 0001BMP255
BATCH                     P-ABORT           00017854A8A0 ADMF001
```

```
V449-HAS NID= DSN:0001.0 AND ID= RUNP10
BATCH                    P-ABORT          00017854AA2E ADMF001
V449-HAS NID= DSN:0002.0 AND ID= RUNP90
BATCH                    P-ABORT          0001785CD711 ADMF001
V449-HAS NID= DSN:0004.0 AND ID= RUNP12
DISPLAY POSTPONED ABORT REPORT COMPLETE
DSN9022I -STR DSNVDT '-DIS THD' NORMAL COMPLETION
```

For an explanation of the displayed list, see the description of messages in Part 2 of *DB2 Messages*.

End of General-use Programming Interface

Duplicate correlation IDs

General-use Programming Interface

Two threads can have the same correlation ID (*pst#.psbname*) if all of these conditions occur:

- Connections have been broken several times.
- Indoubt units of recovery were not recovered.
- Applications were subsequently scheduled in the same region.

To uniquely identify threads which have the same correlation ID (*pst#.psbnam*e) requires that you be able to identify and understand the network ID (NID). For connections with IMS, you should also be able to identify and understand the IMS originating sequence number (OASN).

The NID is shown in a condensed form on the messages issued by the DB2 DISPLAY THREAD command processor. The IMS subsystem name (*imsid*) is displayed as the *net_node*. The *net_node* is followed by the 8-byte OASN, displayed in hexadecimal format (16 characters), with all leading zeros omitted. The *net_node* and the OASN are separated by a period.

For example, if the *net_node* is IMSA, and the OASN is 0003CA670000006E, the NID is displayed as IMSA.3CA670000006E on the DB2 DISPLAY THREAD command output.

If two threads have the same *corr-id*, use the NID instead of *corr-id* on the RECOVER INDOUBT command. The NID uniquely identifies the work unit.

The OASN is a 4-byte number that represents the number of IMS scheduling since the last IMS cold start. The OASN is occasionally found in an 8-byte format, where the first four bytes contain the scheduling number, and the last four bytes contain the number of IMS sync points (commits) during this schedule. The OASN is part of the NID.

The NID is a 16-byte network ID that originates from IMS. The NID contains the 4-byte IMS subsystem name, followed by four bytes of blanks, followed by the 8-byte version of the OASN. In communications between IMS and DB2, the NID serves as the recovery token.

End of General-use Programming Interface

Resolving residual recovery entries

At given times, IMS builds a list of *residual recovery entries* (RREs). RREs are units of recovery about which DB2 could be in doubt. They arise in several situations:

- If DB2 is not operational, IMS has RREs that cannot be resolved until DB2 is operational. Those are not a problem.
- If DB2 is operational and connected to IMS, and if IMS rolled back the work that DB2 has committed, the IMS attachment facility issues message DSNM005I. If the data in the two systems must be consistent, this is a problem situation. Its resolution is discussed in "Resolution of indoubt units of recovery" on page 522.
- If DB2 is operational and connected to IMS, RREs can still exist, even though no messages have informed you of this problem. The only way to recognize this problem is to issue the IMS /DISPLAY OASN SUBSYS command after the DB2 connection to IMS has been established.

  To display the RRE information, issue the command:

  ```
  /DISPLAY OASN SUBSYS subsystem-name
  ```

  To purge the RRE, issue one of these commands:

  ```
  /CHANGE SUBSYS subsystem-name RESET
  /CHANGE SUBSYS subsystem-name RESET OASN nnnn
  ```

  where *nnnn* is the originating application sequence number listed in the display. That is the schedule number of the program instance, telling its place in the sequence of invocations of that program since the last cold start of IMS. IMS cannot have two indoubt units of recovery with the same schedule number.

  Those commands reset the status of IMS; they do not result in any communication with DB2.

Controlling IMS dependent region connections

Controlling IMS dependent region connections involves three activities:
- Connecting from dependent regions
- Monitoring the activity on connections
- Disconnecting from dependent regions

Connecting from dependent regions

The IMS attachment facility used in the control region is also loaded into dependent regions. A connection is made from each dependent region to DB2. This connection is used to pass SQL statements and to coordinate the commitment of DB2 and IMS work. The following process is used by IMS to initialize and connect.

1. Read the SSM from IMS.PROCLIB.

   A subsystem member can be specified on the dependent region EXEC parameter. If it is not specified, the control region SSM is used. If the region will never connect to DB2, specify a member with no entries to avoid loading the attachment facility.
2. Load the DB2 attachment facility from *prefix*.SDSNLOAD

   For a batch message processing (BMP) program, the load is not done until the application issues its first SQL statement. At that time, IMS attempts to make the connection.

   For a message processing program (MPP) region or IMS Fast Path (IFP) region, the connection is made when the IMS region is initialized, and an IMS transaction is available for scheduling in that region.

   An IMS dependent region establishes two connections to DB2: a region connection and an application connection, which occurs at execution of the first SQL statement.

If DB2 is not active, or if resources are not available when the first SQL statement is issued from an application program, the action taken depends on the error option specified on the SSM user entry. The options are:

Option Action

R The appropriate return code is sent to the application, and the SQL code is returned.

Q The application is abended. This is a PSTOP transaction type; the input transaction is re-queued for processing and new transactions are queued.

A The application is abended. This is a STOP transaction type; the input transaction is discarded and new transactions are not queued.

The region error option can be overridden at the program level via the resource translation table (RTT). See Part 2 of *DB2 Installation Guide* for further details.

Monitoring the activity on connections

The information under this heading, up to "Disconnecting from dependent regions" on page 396, is General-use Programming Interface and Associated Guidance Information, as defined in "Notices" on page 1301.

A thread is established from a dependent region when an application makes its first successful DB2 request. Information about connections and the applications currently using them can be displayed by issuing one of these commands:

From DB2:

```
-DISPLAY THREAD (imsid)
```

From IMS:

```
/SSR -DISPLAY THREAD (imsid)
```

Either command produces the following messages:

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV402I - ACTIVE THREADS -
NAME      ST  A  REQ     ID          AUTHID     PLAN     ASID    TOKEN
conn-name s   *  req-ct  corr-id     auth-id    pname    asid    token
conn-name s   *  req-ct  corr-id     auth-id    pname    asid    token
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I - DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
```

For an explanation of the DISPLAY THREAD status information displayed, see the description of message DSNV404I in Part 2 of *DB2 Messages*. More detailed information regarding use of this command and the reports it produces is available in "The command DISPLAY THREAD" on page 405.

IMS provides a display command to monitor the connection to DB2. In addition to showing which program is active on each dependent region connection, the display also shows the LTERM user name and gives the control region connection status. The command is:

```
/DISPLAY SUBSYS subsystem-name
```

The connection between IMS and DB2 is shown as one of the following states:

```
CONNECTED
NOT CONNECTED
CONNECT IN PROGRESS
STOPPED
```

```
STOP IN PROGRESS
INVALID SUBSYSTEM NAME=name
SUBSYSTEM name NOT DEFINED BUT RECOVERY OUTSTANDING
```

The thread status from each dependent region is show as one of the following states:

```
CONN
CONN, ACTIVE (includes LTERM of user)
```

The following four examples show the output that might be generated when an IMS /DISPLAY SUBSYS command is issued.

Figure 29 shows the output that is returned for a DSN subsystem that is not connected. The IMS attachment facility issues message DSNM003I in this example.

```
0000 15.49.57           R 45,/DIS SUBSYS NEW
0000 15.49.57           IEE600I REPLY TO 45 IS;/DIS SUBSYS END
0000 15.49.57 JOB   56  DFS000I DSNM003I IMS/TM V1 SYS3 FAILED TO CONNECT TO SUBSYSTEM DSN RC=00  SYS3
0000 15.49.57 JOB   56  DFS000I     SUBSYS   CRC REGID PROGRAM  LTERM    STATUS   SYS3
0000 15.49.57 JOB   56  DFS000I     DSN      :                          NON CONN    SYS3
0000 15.49.57 JOB   56  DFS000I     *83228/154957*   SYS3
0000 15.49.57 JOB   56 *46 DFS996I *IMS READY*  SYS3
```

Figure 29. Example of output from IMS /DISPLAY SUBSYS processing

Figure 30 shows the output that is returned for a DSN subsystem that is connected. The IMS attachment facility issues message DSNM001I in this example.

```
0000 15.58.59           R 46,/DIS SUBSYS ALL
0000 15.58.59           IEE600I REPLY TO 46 IS;/DIS SUBSYS ALL
0000 15.59.01 JOB   56  DFS551I MESSAGE REGION MPP1     STARTED ID=0001 TIME=1551 CLASS=001,002,003,004
0000 15.59.01 JOB   56  DFS000I DSNM001I IMS/TM=V1 SYS3 CONNECTED TO SUBSYSTEM DSN   SYS3
0000 15.59.01 JOB   56  DFS000I     SUBSYS   CRC REGID PROGRAM  LTERM    STATUS   SYS3
0000 15.59.01 JOB   56  DFS000I     DSN      :                         CONN    SYS3
0000 15.59.01 JOB   56  DFS000I     *83228/155900*   SYS3
0000 15.59.01 JOB   56 *47 DFS996I *IMS READY*  SYS3
```

Figure 30. Example of output from IMS /DISPLAY SUBSYS processing

Figure 31 shows the output that is returned for a DSN subsystem that is in a stopped status. The IMS attachment facility issues message DSNM002I in this example.

```
0000 15.59.28           R 47,/STO SUBSYS ALL
0000 15.59.28           IEE600I REPLY TO 47 IS;/STO SUBSYS ALL
0000 15.59.37 JOB   56  DFS058I 15:59:37 STOP COMMAND IN PROGRESS   SYS3
0000 15.59.37 JOB   56 *48 DFS996I *IMS READY*  SYS3
0000 15.59.44           R 48,/DIS SUBSYS ALL
0000 15.59.44           IEE600I REPLY TO 48 IS;/DIS SUBSYS ALL
0000 15.59.45 JOB   56  DFS000I DSNM002I IMS/TM V1 SYS3 DISCONNECTED FROM SUBSYSTEM DSN RC = E.  SYS3
0000 15.59.45 JOB   56  DFS000I     SUBSYS   CRC REGID PROGRAM  LTERM    STATUS   SYS3
0000 15.59.45 JOB   56  DFS000I     DSN      :                          STOPPED    SYS3
0000 15.59.45 JOB   56  DFS000I     *83228/155945*   SYS3
0000 15.59.45 JOB   56 *49 DFS996I *IMS READY*  SYS3
```

Figure 31. Example of output from the IMS /DISPLAY SUBSYS command

Figure 32 on page 396 shows the output that is returned for a DSN subsystem that is connected and region 1. You can use the values from the REGID and the PROGRAM fields to correlate the output of the command to the LTERM that is involved.

```
0000 16.09.35 JOB   56  R 59,/DIS SUBSYS ALL
0000 16.09.35 JOB   56  IEE600I REPLY TO 59 IS;/DIS SUBSYS ALL
0000 16.09.38 JOB   56  DFS000I     SUBSYS   CRC REGID PROGRAM  LTERM    STATUS   SYS3
0000 16.09.38 JOB   56  DFS000I     DSN      :                           CONN     SYS3
0000 16.09.38 JOB   56  DFS000I                         1                CONN     SYS3
0000 16.09.38 JOB   56  DFS000I     *83228/160938*   SYS3
0000 16.09.38 JOB   56 *60 DFS996I *IMS READY*  SYS3
0000 16.09.38 JOB   56
```

Figure 32. Example of output from IMS /DISPLAY SUBSYS processing for a DSN subsystem that is connected and the region ID (1) that is included.

Disconnecting from dependent regions

Usually, IMS master terminal operators do not want to disconnect a dependent region explicitly. However, they might want to change values in the SSM member of IMS.PROCLIB. To do that, they can issue /STOP REGION, update the SSM member, and issue /START REGION.

Disconnecting from IMS

The connection is ended when either IMS or DB2 terminates. Alternatively, the IMS master terminal operator can explicitly break the connection by entering this command:

```
/STOP SUBSYS subsystem-name
```

That command sends the following message to the terminal that entered it, usually the master terminal operator (MTO):

```
DFS058I STOP COMMAND IN PROGRESS
```

The /START SUBSYS *subsystem-name* command is required to reestablish the connection.

In implicit or explicit disconnect, this message is sent to the IMS master terminal:

```
DSNM002I IMS/TM imsid DISCONNECTED FROM SUBSYSTEM subsystem-name - RC=z
```

That message uses the following reason codes (RC):

Code	Meaning
A	IMS/TM is terminating normally (for instance, /CHE FREEZE³DUMPQ³PURGE). Connected threads complete.
B	IMS is abending. Connected threads are rolled back. DB2 data is backed out now; DL/I data is backed out on IMS restart.
C	DB2 is terminating normally after a STOP DB2 MODE (QUIESCE) command. Connected threads complete.
D	DB2 is terminating normally after a STOP DB2 MODE (FORCE) command, or DB2 is abending. Connected threads are rolled back. DL/I data is backed out now. DB2 data is backed out now if DB2 terminated normally; otherwise, at restart.
E	IMS is ending the connection because of a /STOP SUBSYS *subsystem-name* command. Connected threads complete.

If an application attempts to access DB2 after the connection ended and before a thread is established, the attempt is handled according to the region error option specification (R, Q, or A).

Controlling RRS connections

General-use Programming Interface

Application programs can use the following Resource Recovery Services attachment facility (RRSAF) functions to control connections to DB2:

IDENTIFY
Establishes the task (TCB) as a user of the named DB2 subsystem. When the first task within an address space issues a connection request, the address space is initialized as a user of DB2.

SIGNON
Provides a user ID and optionally, one or more secondary authorization IDs to be associated with the connection. Invokes the sign-on exit routine. Optionally, lets a thread join a global transaction. See "Recommendations for application design" on page 814 for more information about global transactions.

AUTH SIGNON
Provides a user ID, an ACEE, and optionally, one or more secondary authorization IDs to be associated with the connection. Invokes the sign-on exit.

CREATE THREAD
Allocates a plan. If you provide a plan name, DB2 allocates that plan. If you provide a collection name, DB2 allocates a special plan named ?RRSAF and a package list that contains the collection name.

After CREATE THREAD completes, DB2 can execute SQL statements.

TERMINATE THREAD
Deallocates the plan.

TERMINATE IDENTIFY
Removes the task as a user of DB2. If this is the last or only task in the address space with a DB2 connection, TERMINATE IDENTIFY terminates the address space connection to DB2.

TRANSLATE
Returns an SQL code and printable text, in the SQLCA, that describes a DB2 error reason code.

End of General-use Programming Interface

For more information about those functions, see Part 6 of *DB2 Application Programming and SQL Guide*.

Connecting to RRS using RRSAF

The information under this heading up to "Controlling connections to remote systems" on page 400 is General-use Programming Interface and Associated Guidance Information, as defined in "Notices" on page 1301.

An RRSAF connection can be started or restarted at any time after RRS is started. If RRS is not started, an IDENTIFY request fails with reason code X'00F30091'.

Restarting DB2 and RRS

If DB2 abnormally terminates but RRS remains active, RRS might commit or roll back work without DB2's knowledge. In a similar manner, if RRS abnormally terminates after DB2 has completed phase 1 of commit processing for an application, then DB2 does not know whether to commit or roll back the work. In either case, when DB2 restarts, that work is termed *indoubt*.

DB2 cannot resolve those indoubt units of recovery (that is, commit or roll back the changes made to DB2 resources) until DB2 restarts with RRS.

If any unit of work is indoubt when a failure occurs, DB2 and RRS automatically resolve the unit of work when DB2 restarts with RRS.

Displaying indoubt units of recovery

To display a list of indoubt units of recovery, issue the command:

```
-DISPLAY THREAD (RRSAF) TYPE (INDOUBT)
```

The command produces output similar to this:

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV406I - INDOUBT THREADS -
COORDINATOR              STATUS      RESET URID           AUTHID
RRSAF                    INDOUBT           00019B8ADE9E   ADMF001
 V449-HAS NID= AD64101C7EED90000000000101010000 AND ID= ST47653RRS
DISPLAY INDOUBT REPORT COMPLETE
DSN9022I - DSNVDT '-DIS THD' NORMAL COMPLETION
```

For RRSAF connections, a network ID is the z/OS RRS Unit of Recovery ID (URID) that uniquely identifies a unit of work. A z/OS RRS URID is a 32 character number. For an explanation of the output, see the description of message DSNV408I in Part 2 of *DB2 Messages*.

Recovering indoubt units of recovery manually

Manual recovery of an indoubt unit of recovery might be required if the RRS log is lost. When that happens, message DSN3011I is displayed on the z/OS console.

To recover an indoubt unit of recovery, issue one of the following commands:

```
-RECOVER INDOUBT (RRSAF) ACTION (COMMIT) ID (correlation-id)
```

or

```
-RECOVER INDOUBT (RRSAF) ACTION (ABORT) ID (correlation-id)
```

correlation-id is the correlation ID of the thread to be recovered. You can determine the correlation ID by issuing the DISPLAY THREAD command.

The ACTION parameter indicates whether to commit or roll back the associated unit of recovery. For more details, see “Resolving indoubt units of recovery” on page 459.

If you recover a thread that is part of a global transaction, all threads in the global transaction are recovered.

The following messages can occur when you issue the RECOVER INDOUBT command:

```
DSNV414I - THREAD correlation-id COMMIT SCHEDULED
DSNV415I - THREAD correlation-id ABORT SCHEDULED
```

If the following DSNV418I message is issued:

```
DSNV418I - RECOVER INDOUBT REJECTED FOR ID=correlation-id
```

You must use the following NID option of RECOVER INDOUBT:

```
-RECOVER INDOUBT(RRSAF) ACTION(action) NID(nid)
```

where *nid* is the 32-character field displayed in the DSNV449I message.

For information about the two-phase commit process, as well as indoubt units of recovery, see "Multiple system consistency" on page 455.

Displaying postponed units of recovery

To display a list of postponed units of recovery, issue the command:

```
-DISPLAY THREAD (RRSAF) TYPE (POSTPONED)
```

The command produces output similar to this:

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV406I - POSTPONED ABORT THREADS -
COORDINATOR              STATUS      RESET URID           AUTHID
RRSAF                    P-ABORT           00019B8ADE9E   ADMF001
V449-HAS NID= AD64101C7EED90000000000101010000 AND ID= ST47653RRS
DISPLAY POSTPONED ABORT REPORT COMPLETE
DSN9022I - DSNVDT '-DIS THD' NORMAL COMPLETION
```

For RRSAF connections, a network ID is the z/OS RRS Unit of Recovery ID (URID) that uniquely identifies a unit of work. A z/OS RRS URID is a 32-character number. For an explanation of the output, see the description of message DSNV408I in Part 2 of *DB2 Messages*.

Monitoring RRSAF connections

RRSAF allows an application or application monitor to disassociate a DB2 thread from a TCB and later associate the thread with the same or different TCB within the same address space. RRSAF uses the RRS Switch Context (CTXSWCH) service to do this. Only authorized programs can execute CTXSWCH.

DB2 stores information in an RRS CONTEXT about an RRSAF thread so that DB2 can locate the thread later. An application or application monitor can then invoke CTXSWCH to dissociate the CONTEXT from the current TCB and then associate the CONTEXT with the same TCB or a different TCB.

Displaying RRSAF connections

The following command displays information about RRSAF threads, including those that access data at remote locations:

```
-DISPLAY THREAD(RRSAF)
```

The command produces output similar to this:

```
DSNV401I = DISPLAY THREAD REPORT FOLLOWS -
DSNV402I = ACTIVE THREADS -
NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
1 RRSAF    T        4 RRSTEST2-111 ADMF001  ?RRSAF   0024    13
2 RRSAF    T        6 RRSCDBTEST01 USRT001  TESTDBD  0024    63
3 RRSAF    DI       3 RRSTEST2-100 USRT002  ?RRSAF   001B    99
4 RRSAF    TR       9 GT01XP05     SYSADM   TESTP05  001B   235
           V444-DB2NET.LUND0.AA8007132465=16 ACCESSING DATA AT
           V446-SAN_JOSE:LUND1
DISPLAY ACTIVE REPORT COMPLETE
```

Key:

1 This is an application that used CREATE THREAD to allocate the special plan used by RRSAF (plan name = ?RRSAF).

2 This is an application that connected to DB2 and allocated a plan with the name TESTDBD.

3 This is an application that is currently not connected to a TCB (shown by status DI).

4 This is an active connection that is running plan TESTP05. The thread is accessing data at a remote site.

Figure 33. DISPLAY THREAD output showing RRSAF connections

Disconnecting RRSAF applications from DB2

You cannot disconnect an RRSAF transaction from DB2 without abending the transaction. You can use the DB2 command CANCEL THREAD to cancel a particular thread. CANCEL THREAD requires that you know the *token* for any thread that you want to cancel. Issue the command DISPLAY THREAD to obtain the token number, then enter the following command to cancel the thread:

```
-CANCEL THREAD(token)
```

When you issue CANCEL THREAD, DB2 schedules the thread for termination.

Controlling connections to remote systems

The information under this heading, up to “Using NetView to monitor errors” on page 416, is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page 1301.

You can control connections to remote systems, which use distributed data, by controlling the threads. Two types of threads are involved with connecting to other systems, *allied threads* and *database access threads*. An allied thread is a thread that is connected locally to your DB2 subsystem, that is from TSO, CICS, IMS, or a stored procedures address space. A database access thread is a thread initiated by a remote DBMS to your DB2 subsystem. The following topics are covered here:

“Starting the DDF” on page 401
“Suspending and resuming DDF server activity” on page 401
“Monitoring connections to other systems” on page 402, which describes the use of the following commands:
- DISPLAY DDF
- DISPLAY LOCATION
- DISPLAY THREAD
- CANCEL THREAD
- VARY NET,TERM (VTAM command)

“Monitoring and controlling stored procedures” on page 413
“Using NetView to monitor errors” on page 416
“Stopping the DDF” on page 417

Related information: The following topics in this book contain information about distributed connections:

"Resolving indoubt units of recovery" on page 459
"Database access thread failure recovery" on page 555
Chapter 36, "Tuning and monitoring in a distributed environment," on page 1003

Starting the DDF

To start the distributed data facility (DDF), if it has not already been started, use the following command:

```
-START DDF
```

When DDF is started and is responsible for indoubt thread resolution with remote partners, one or both of messages DSNL432I and DSNL433I is generated. These messages summarize DDF's responsibility for indoubt thread resolution with remote partners. See Chapter 20, "Maintaining consistency across multiple systems," on page 455 for information about resolving indoubt threads.

Using the START DDF command requires authority of SYSOPR or higher. The following messages are associated with this command:

```
DSNL003I - DDF IS STARTING

DSNL004I - DDF START COMPLETE LOCATION locname
           LU netname.luname

           GENERICLU netname.gluname
           DOMAIN domain
           TCPPORT tcpport
           RESPORT resport
```

If the distributed data facility has not been properly installed, the START DDF command fails, and message `DSN9032I, - REQUESTED FUNCTION IS NOT AVAILABLE` is issued. If the distributed data facility has already been started, the START DDF command fails, and message `DSNL001I, - DDF IS ALREADY STARTED` is issued. Use the DISPLAY DDF command to display the status of DDF.

When you install DB2, you can request that the distributed data facility start automatically when DB2 starts. For information about starting the distributed data facility automatically, see Part 2 of *DB2 Installation Guide*.

Suspending and resuming DDF server activity

You can use the STOP DDF MODE(SUSPEND) command to suspend DDF server threads temporarily. Suspending DDF server threads frees all resources held by the server threads and lets the following operations complete:

- CREATE
- ALTER
- DROP
- GRANT
- REVOKE

When you issue STOP DDF MODE(SUSPEND), DB2 waits for all active DDF database access threads to become pooled or to terminate. Two optional keywords on this command, WAIT and CANCEL, let you control how long DB2 waits and what action DB2 takes after a specified time period. To resume suspended DDF

server threads, issue the START DDF command. For more detailed information about the STOP DDF MODE(SUSPEND) command, see Chapter 2 of *DB2 Command Reference*.

Monitoring connections to other systems

The following DB2 commands give you information about distributed threads:

DISPLAY DDF
Displays information about the status and configuration of the distributed data facility (DDF), and about the connections or threads controlled by DDF. For its use, see "The command DISPLAY DDF."

DISPLAY LOCATION
Displays statistics about threads and conversations between remote DB2 subsystem and the local subsystem. For its use, see "The command DISPLAY LOCATION" on page 403.

DISPLAY THREAD
Displays information about DB2, distributed subsystem connections, and parallel tasks. For its use, see "The command DISPLAY THREAD" on page 405.

The command DISPLAY DDF

The command DISPLAY DDF displays information regarding the status of DDF and, in addition, any information that is displayed when DDF is started, such as the location name, the LU name, the IP address, and domain names. Using the DETAIL keyword provides additional configuration and statistical information.

To issue the DISPLAY DDF command, you must have SYSOPR authority or higher. Issue the following command:

```
-DISPLAY DDF
```

DB2 returns output similar to this sample when DDF is **not** started:

```
DSNL080I - DSNLTDDF DISPLAY DDF REPORT FOLLOWS-
DSNL081I  1 STATUS=STOPDQ
DSNL082I  2 LOCATION           3 LUNAME            4 GENERICLU
DSNL083I    SVL650A              -NONE.SYEC650A      -NONE
DSNL084I  5 IPADDR             6 TCPPORT           7 RESPORT
DSNL085I    -NONE                447                 5002
DSNL086I  8 SQL    DOMAIN=-NONE
DSNL086I  9 RESYNC DOMAIN=-NONE
DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE
```

DB2 returns output similar to this sample when DDF is started:

```
DSNL080I - DSNLTDDF DISPLAY DDF REPORT FOLLOWS-
DSNL081I  1 STATUS=STARTD
DSNL082I  2 LOCATION           3 LUNAME            4 GENERICLU
DSNL083I    SVL650A              USIBMSY.SYEC650A    -NONE
DSNL084I  5 IPADDR             6 TCPPORT           7 RESPORT
DSNL085I    8.110.115.106        447                 5002
DSNL086I  8 SQL    DOMAIN=v8ec103.svl.ibm.com
DSNL086I  9 RESYNC DOMAIN=v8ec103.svl.ibm.com
DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE
```

Key	Description
1	The status of the distributed data facility (DDF)
2	The location name of DDF defined in the BSDS
3	The fully qualified LU name for DDF (that is, the network ID and LUNAME)
4	The fully qualified generic LU name for DDF

5 The IP address of DDF
6 The SQL listener TCP/IP port number
7 The two-phase commit resynchronization (resync) listener TCP/IP port number
8 The domain that accepts inbound SQL requests from remote partners
9 The domain that accepts inbound two-phase commit resynchronization requests

To use the DETAIL option, enter the following command:

```
-DISPLAY DDF DETAIL
```

DB2 returns additional lines of output:

```
DSNL080I - DSNLTDDF DISPLAY DDF REPORT FOLLOWS-
DSNL081I STATUS=STARTD
...
DSNL090I  10 DT=      A  11 CONDBAT=  64  12 MDBAT=   64
DSNL092I  13 ADBAT=   1  14 QUEDBAT=   0  15 IN1DBAT=  0  16 CONQUED=      0
DSNL093I  17 DSCDBAT= 0  18 IN2CONS=   0
DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE
```

Key	Description
10	The DDF thread value: **A** Indicates that DDF is configured with DDF THREADS ACTIVE **I** Indicates that DDF is configured with DDF THREADS INACTIVE
11	The maximum number of inbound connections for database access threads
12	The maximum number of concurrent active DBATs that can execute SQL
13	The current number of active database access threads
14	The total number of queued database access threads. This count is cumulative and resets only when DB2 restarts.
15	The current number of inactive DBATs (type 1 inactive threads)
16	The current number of connection requests that are queued and waiting
17	The current number of pooled database access threads
18	The current number of inactive connections (type 2 inactive threads)

For more DISPLAY DDF message information, see Part 2 of *DB2 Messages*.

The DISPLAY DDF DETAIL command is especially useful because it reflects the presence of new inbound connections that are not reflected by other commands. For example, if DDF is in INACTIVE MODE, as denoted by a DT value of **I** in the DSNL090I message, and DDF is stopped, suspended, or the maximum number of active data base access threads has been reached, then new inbound connections are not yet reflected in the DISPLAY THREAD report. However, the presence of these new connections is reflected in the DISPLAY DDF DETAIL report, although specific details regarding the origin of the connections, such as the client system IP address or LU name, are not available until the connections are actually associated with a database access thread.

The command DISPLAY LOCATION

The command DISPLAY LOCATION displays summary information about connections with other locations and can be used to display detailed information about DB2 system conversations. System conversations are used either for DB2 private protocol access or for supporting functions with DRDA access. Location names, SNA LU names or IP addresses, can be specified, and the DETAIL keyword is supported. To issue the DISPLAY LOCATION command, you must have SYSOPR authority or higher. Issue the following command:

```
-DISPLAY LOCATION(*)
```

DB2 returns output similar to this sample:

```
DSNL200I - DISPLAY LOCATION REPORT FOLLOWS-
LOCATION         PRDID    LINKNAME        REQUESTERS SERVERS CONVS
USIBMSTODB22     DSN05010 LUND0                    1       0      3
USIBMSTODB23     DSN04010 LUND1                    0       0      0
DRDALOC          SQL03030 124.63.51.17             3       0      3
124.63.51.17     SQL03030 124.63.51.17             0      15     15
DISPLAY LOCATION REPORT COMPLETE
```

You can use an asterisk (*) in place of the end characters of a location name. For example, use DISPLAY LOCATION(SAN*) to display information about all active connections between your DB2 and a remote location that begins with “SAN”. This includes the number of conversations and the role for each non-system conversation, requester or server.

When DB2 connects with a remote location, information about that location, including LOCATION, PRDID and LINKNAME (LUNAME or IP address), persists in the report even if no active connections exist.

The DISPLAY LOCATION command displays the following types of information for each DBMS that has active threads, except for the local subsystem:

- The location name (or RDB_NAME) of the other connected system. If the RDBNAME is not known, the LOCATION column contains one of the following identifiers:
  - A VTAM LU name in this format: ’<luname>’.
  - A dotted decimal IP address in this format: ’nnn.nnn.nnn.nnn’.
- The PRDID, which identifies the database product at the location in the form *nnnvvrrm*:
  - *nnn* - identifies the database product
  - *vv* - product version
  - *rr* - product release
  - *m* - product modification level.
- The corresponding LUNAME or IP address of the system.
- The number of threads at the local system that are requesting data from the remote system.
- The number of threads at the local system that are acting as a server to the remote system.
- The total number of conversations in use between the local system and the remote system. For USIBMSTODB23, in the preceding sample output, the locations are connected and system conversations have been allocated, but currently there are no active threads between the two sites.

DB2 does not receive a location name from non-DB2 requesting DBMSs that are connected to DB2. In this case, it displays instead the LUNAME of the requesting DBMS, enclosed in less-than (<) and greater-than (>) symbols.

For example, suppose there are two threads at location USIBMSTODB21. One is a distributed access thread from a non-DB2 DBMS, and the other is an allied thread going from USIBMSTODB21 to the non-DB2 DBMS. The DISPLAY LOCATION command issued at USIBMSTODB21 displays the following output:

```
DSNL200I - DISPLAY LOCATION REPORT FOLLOWS -
LOCATION         PRDID    LINKNAME        REQUESTERS SERVERS CONVS
NONDB2DBMS                LUND1                     1       0      1
<LULA>           DSN04010 LULA                      0       1      1
DISPLAY LOCATION REPORT COMPLETE
```

The following output shows the result of a DISPLAY LOCATION(*) command when DB2 is connected to the following DRDA partners:

- DB2A is connected to this DB2, using TCP/IP for DRDA connections and SNA for DB2 private protocol connections.
- DB2SERV is connected to this DB2 using only SNA.

```
DSNL200I - DISPLAY LOCATION REPORT FOLLOWS -
LOCATION         PRDID    LINKNAME        REQUESTERS SERVERS CONVS
DB2A             DSN05010 LUDB2A                   3       4      9
DB2A             DSN05010 124.38.54.16             2       1      3
DB2SERV          DSN04010 LULA                     1       1      3
DISPLAY LOCATION REPORT COMPLETE
```

The DISPLAY LOCATION command displays information for each remote location that currently is, or once was, in contact with DB2. If a location is displayed with zero conversations, one of the following conditions exist:

- Sessions currently exist with the partner location but there are currently no active conversations allocated to any of the sessions.
- Sessions no longer exist with the partner because contact with the partner has been lost.

If you use the DETAIL parameter, each line is followed by information about conversations owned by DB2 system threads, including those used for resynchronization of indoubt units of work.

The command DISPLAY THREAD

Displaying information by location: Use the LOCATION keyword, followed by a list of location names, to display thread information for particular locations.

You can use an asterisk (*) after the THD and LOCATION keywords just as in the DISPLAY LOCATION command previously described. For example, enter:

```
-DISPLAY THREAD(*) LOCATION(*) DETAIL
```

DB2 returns messages like these:

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV402I - ACTIVE THREADS -
   NAME     ST 1 A 2    REQ ID        AUTHID   PLAN     ASID TOKEN
   SERVER   RA   *     2923 DB2BP     ADMF001  DISTSERV 0036    20 3
    V437-WORKSTATION=ARRAKIS, USERID=ADMF001,
          APPLICATION NAME=DB2BP
    V436-PGM=NULLID.SQLC27A4, SEC=201, STMNT=210
    V445-09707265.01BE.889C28200037=203 ACCESSING DATA FOR 9.112.12.101
    V447-LOCATION            SESSID       A ST TIME
    V448-9.112.12.101 4      446:1300 5   W S2 9802812045091
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I - DSNVDT '-DIS THD' NORMAL COMPLETION
```

Key	Description
1	The ST (status) column contains characters that indicate the connection status of the local site. The *TR* indicates that an allied, distributed thread has been established. The *RA* indicates that a distributed thread has been established and is in receive mode. The *RD* indicates that a distributed thread is performing a remote access on behalf of another location (R) and is performing an operation involving DCE services (D). Currently, DB2 supports the optional use of DCE services to authenticate remote users.
2	The A (active) column contains an asterisk indicating that the thread is active within DB2. It is blank when the thread is inactive within DB2 (active or waiting within the application).
3	This LUWID is unique across all connected systems. This thread has a token of *20* (it appears in two places in the display output).
4	This is the location of the data that the local application is accessing. If the RDBNAME is not known, the location column contains either a VTAM LUNAME or a dotted decimal IP address.
5	If the connection uses TCP/IP, the sessid column contains "local:remote", where "local" specifies DB2's TCP/IP port number and "remote" specifies the partner's TCP/IP port number.

For more information about this sample output and connection status codes, see message DSNV404I, DSNV444I, and DSNV446I, in Part 2 of *DB2 Messages*.

Displaying information for non-DB2 locations: Because DB2 does not receive a location name from non-DB2 locations, you must enter the LUNAME or IP address of the location for which you want to display information. The LUNAME is enclosed by the less-than (<) and greater-than (>) symbols. The IP address is in the dotted decimal format. For example, if you wanted to display information about a non-DB2 DBMS with the LUNAME of LUSFOS2, you would enter the following command:

```
-DISPLAY THREAD (*) LOCATION (<LUSFOS2>)
```

DB2 uses the <LUNAME> notation or dotted decimal format in messages displaying information about non-DB2 requesters.

Displaying conversation-level information about threads: Use the DETAIL keyword with the LOCATION keyword to give you information about conversation activity when distribution information is displayed for active threads. This keyword has no effect on the display of indoubt threads. See Chapter 2 of *DB2 Command Reference* for more information about the DETAIL keyword.

For example, issue:

```
-DISPLAY THREAD(*) LOCATION(*) DETAIL
```

DB2 returns the following message, indicating that the local site application is waiting for a conversation to be allocated in DB2, and a DB2 server that is accessed by a DRDA client using TCP/IP.

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV402I - ACTIVE THREADS -
   NAME     ST A   REQ ID           AUTHID   PLAN     ASID   TOKEN
   TSO      TR *     3 SYSADM       SYSADM   DSNESPRR 002E       2
    V436-PGM=DSNESPRR.DSNESM68, SEC=1, STMNT=116
    V444-DB2NET.LUND0.A238216C2FAE=2 ACCESSING DATA AT
    V446-USIBMSTODB22:LUND1
    V447--LOCATION        SESSID        1 A ST    TIME
    V448--USIBMSTODB22    0000000000000000 V A1 2  9015816504776
   TSO      RA *    11 SYSADM       SYSADM   DSNESPRR 001A      15
    V445-STLDRIV.SSLU.A23555366A29=15 ACCESSING DATA FOR 123.34.101.98
    V447--LOCATION        SESSID           A ST    TIME
    V448--123.34.101.98   446:3171  3        S2    9015611253108
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I - DSNVDT '-DIS THD' NORMAL COMPLETION
```

Key	Description
1	The information on this line is part of message DSNV447I. The conversation A (active) column for the server is useful in determining when a DB2 thread is hung and whether processing is waiting in VTAM or in DB2. A value of *W* indicates that the thread is suspended in DB2 and is waiting for notification from VTAM that the event has completed. A value of*V* indicates that control of the conversation is in VTAM.
2	The information on this line is part of message DSNV448I. The *A* in the conversation ST (status) column for a serving site indicates a conversation is being allocated in DB2. The *1* indicates that the thread uses DB2 private protocol access. A *2* would indicate DRDA access. An *R* in the status column would indicate that the conversation is receiving or waiting to receive a request or reply. An *S* in this column for a server indicates that the application is sending or preparing to send a request or reply.
3	The information on this line is part of message DSNV448I. The SESSID column has changed as follows. If the connection uses VTAM, the SESSID column contains a VTAM session identifier. If the connection uses TCP/IP, the sessid column contains "local:remote", where "local" specifies the DB2 TCP/IP port number, and "remote" specifies the partner's TCP/IP port number.

For more DISPLAY THREAD message information, see messages DSNV447I and DSNV448I, Part 2 of *DB2 Messages*.

Monitoring all DBMSs in a transaction: The DETAIL keyword of the command DISPLAY THREAD allows you to monitor all of the requesting and serving DBMSs involved in a transaction.

For example, you could monitor an application running at USIBMSTODB21 requesting information from USIBMSTODB22, which must establish conversations with secondary servers USIBMSTODB23 and USIBMSTODB24 to provide the requested information. Figure 34 on page 408 depicts such an example. In this example, ADA refers to DRDA access and SDA refers to DB2 private protocol access. USIBMSTODB21 is considered to be *upstream* from USIBMSTODB22. USIBMSTODB22 is considered to be upstream from USIBMSTODB23. Conversely, USIBMSTODB23 and USIBMSTODB22 are *downstream* from USIBMSTODB22 and USIBMSTODB21 respectively.

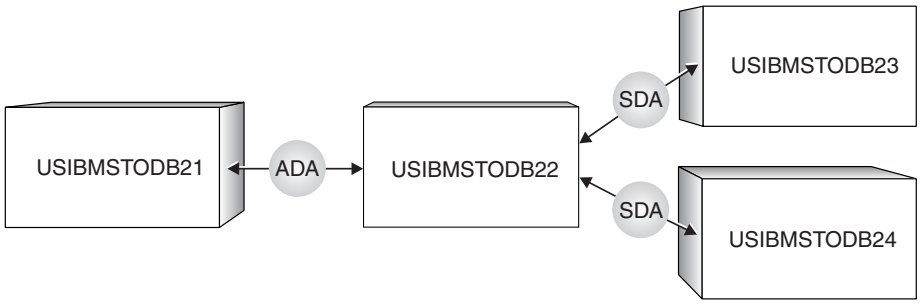


Figure 34. Example of a DB2 transaction involving four sites.

The application running at USIBMSTODB21 is connected to a server at USIBMSTODB22, using DRDA access. If you enter the DISPLAY THREAD command with the DETAIL keyword from USIBMSTODB21, you receive the output that Figure 35 depicts.

```
-DIS THD(*) LOC(*) DET
   DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
   DSNV402I - ACTIVE THREADS -
   NAME     ST A   REQ ID           AUTHID PLAN     ASID   TOKEN
   BATCH    TR *     6 BKH2C        SYSADM YW1019C  0009       2
    V436-PGM=BKH2C.BKH2C, SEC=1, STMNT=4
    V444-USIBMSY.SSLU.A23555366A29=2 ACCESSING DATA AT
    V446-USIBMSTODB22:SSURLU
    V447--LOCATION         SESSID           A ST TIME
    V448--USIBMSTODB22     0000000300000004 V R2 9015611253116
   DISPLAY ACTIVE REPORT COMPLETE
  11:26:23 DSN9022I - DSNVDT '-DIS THD' NORMAL COMPLETION
```

Figure 35. DISPLAY THREAD DETAIL output from USIBMSTODB21

This output indicates that the application is waiting for data to be returned by the server at USIBMSTODB22.

The server at USIBMSTODB22 is running a package on behalf of the application at USIBMSTODB21, in order to access data at USIBMSTODB23 and USIBMSTODB24 by DB2 private protocol access. If you enter the DISPLAY THREAD command with the DETAIL keyword from USIBMSTODB22, you receive the output that Figure 36depicts.

```
-DIS THD(*) LOC(*) DET
   DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
   DSNV402I - ACTIVE THREADS -
   NAME     ST A   REQ ID           AUTHID PLAN     ASID   TOKEN
   BATCH    RA *     0 BKH2C        SYSADM YW1019C  0008       2
    V436-PGM=BKH2C.BKH2C, SEC=1, STMNT=4
    V445-STLDRIV.SSLU.A23555366A29=2 ACCESSING DATA FOR
         USIBMSTODB21:SSLU
    V444-STLDRIV.SSLU.A23555366A29=2 ACCESSING DATA AT
    V446-USIBMSTODB23:OSSLU USIBMSTODB24:OSSURLU
    V447--LOCATION         SESSID           A ST TIME
    V448--USIBMSTODB21     0000000300000004   S2 9015611253108
    V448--USIBMSTODB23     0000000600000002   S1 9015611253077
    V448--USIBMSTODB24     0000000900000005 V R1 9015611253907
   DISPLAY ACTIVE REPORT COMPLETE
  11:26:34 DSN9022I - DSNVDT '-DIS THD' NORMAL COMPLETION
```

Figure 36. DISPLAY THREAD DETAIL output from USIBMSTODB22

This output indicates that the server at USIBMSTODB22 is waiting for data to be returned by the secondary server at USIBMSTODB24.

The secondary server at USIBMSTODB23 is accessing data for the primary server at USIBMSTODB22. If you enter the DISPLAY THREAD command with the DETAIL keyword from USIBMSTODB23, you receive the output that Figure 37 depicts.

```
-DIS THD(*) LOC(*) DET
   DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
   DSNV402I - ACTIVE THREADS -
   NAME     ST A   REQ ID           AUTHID PLAN     ASID   TOKEN
   BATCH    RA *     2 BKH2C        SYSADM YW1019C  0006       1
    V445-STLDRIV.SSLU.A23555366A29=1 ACCESSING DATA FOR
         USIBMSTODB22:SSURLU
    V447--LOCATION         SESSID           A ST TIME
    V448--USIBMSTODB22     0000000600000002 W R1 9015611252369
   DISPLAY ACTIVE REPORT COMPLETE
  11:27:25 DSN9022I - DSNVDT '-DIS THD' NORMAL COMPLETION
```

Figure 37. DISPLAY THREAD DETAIL output from USIBMSTODB23

This output indicates that the secondary server at USIBMSTODB23 is not currently active.

The secondary server at USIBMSTODB24 is also accessing data for the primary server at USIBMSTODB22. If you enter the DISPLAY THREAD command with the DETAIL keyword from USIBMSTODB24, you receive the output that Figure 38:

```
-DIS THD(*) LOC(*) DET
   DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
   DSNV402I - ACTIVE THREADS -
   NAME     ST A   REQ ID           AUTHID PLAN     ASID   TOKEN
   BATCH    RA *     2 BKH2C        SYSADM YW1019C  0006       1
    V436-PGM=*.BKH2C, SEC=1, STMNT=1
    V445-STLDRIV.SSLU.A23555366A29=1 ACCESSING DATA FOR
         USIBMSTODB22:SSURLU
    V447--LOCATION         SESSID           A ST TIME
    V448--USIBMSTODB22     0000000900000005   S1 9015611253075
   DISPLAY ACTIVE REPORT COMPLETE
  11:27:32 DSN9022I - DSNVDT '-DIS THD' NORMAL COMPLETION
```

Figure 38. DISPLAY THREAD DETAIL output from USIBMSTODB24

This output indicates that the secondary server at USIBMSTODB24 is currently active.

The conversation status might not change for a long time. The conversation could be hung, or the processing could just be taking a long time. To see whether the conversation is hung, issue DISPLAY THREAD again and compare the new timestamp to the timestamps from previous output messages. If the timestamp is changing, but the status is not, the job is still processing. If you need to terminate a distributed job, perhaps because it is hung and has been holding database locks for a long period of time, you can use the CANCEL DDF THREAD command if the thread is in DB2 (whether active or suspended) or the VARY NET TERM command if the thread is within VTAM. See “The command CANCEL THREAD” on page 411.

Displaying threads by LUWIDs: Use the LUWID optional keyword, which is only valid when DDF has been started, to display threads by logical unit of work identifiers. The LUWIDs are assigned to the thread by the site that originated the thread.

You can use an asterisk (*) in an LUWID as in a LOCATION name. For example, use -DISPLAY THREAD TYPE(INDOUBT) LUWID(NET1.*) to display all the indoubt threads whose LUWID has a network name of NET1. The command

DISPLAY THREAD TYPE(INDOUBT) LUWID(IBM.NEW*) displays all indoubt threads whose LUWID has a network name of ″IBM″ and whose LUNAME begins with ″NEW.″

The DETAIL keyword can also be used with the DISPLAY THREAD LUWID command to show the status of every conversation connected to each thread displayed and to indicate whether a conversation is using DRDA access or DB2 private protocol access.

To issue this command, enter:

```
-DIS THD(*) LUWID (luwid) DETAIL
```

DB2 returns the message that Figure 39 depicts.

```
-DIS THD(*) LUWID (luwid) DET
  DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
  DSNV402I - ACTIVE THREADS -
  NAME     ST A   REQ ID           AUTHID   PLAN     ASID   TOKEN
  BATCH    TR       5 TC3923S0     SYSADM   TC392    000D       2
   V436-PGM=*.TC3923S0, SEC=1, STMNT=116
   V444-DB2NET.LUNSITE0.A11A7D7B2057=2  1 ACCESSING DATA AT
   V446-USIBMSTODB22:LUNSITE1
   V447--LOCATION         SESSID           A ST    TIME
   V448--USIBMSTODB22     00C3F4228C5A244C   S2 2  8929612225354
  DISPLAY ACTIVE REPORT COMPLETE
  DSN9022I - DSNVDT '-DIS THD' NORMAL COMPLETION
```

Key	Description
1	In the preceding display output, you can see that the LUWID has been assigned a token of *2*. You can use this token instead of the long version of the LUWID to cancel or display the given thread. For example: `-DIS THD(*) LUWID(2) DET`
2	In addition, the status column for the serving site contains a value of *S2*. The *S* means that this thread can send a request or response, and the *2* means that this is an DRDA access conversation.

Figure 39. DISPLAY THREAD LUWID output

Canceling dynamic SQL from a client application

You can use the CLI and ODBC function SQLCancel() or the JDBC cancel method to cancel a remote SQL request from a client application. To cancel SQL that is running on a remote DB2 server, use the following procedure:

1. Establish an additional remote connection from your application to the remote DB2 server.
2. From the connection you establish in step 1, issue SQLCancel() or invoke the JDBC cancel method.

When you cancel an SQL statement from a client application, you do not eliminate the original connection to the remote server. The original connection remains active to process additional SQL requests. Any cursor that is associated with the canceled statement is closed and the DB2 server returns an SQLCODE of -952 to the client application when you cancel a statement by using this method.

You can cancel only dynamic SQL that excludes transaction level statements (CONNECT, COMMIT, ROLLBACK) and bind statements from a client application.

For more information about SQLCancel(), see *DB2 ODBC Guide and Reference*. For more information about the JDBC cancel method, see *DB2 Application Programming Guide and Reference for Java*.

The command CANCEL THREAD

You can use the command CANCEL THREAD to terminate threads that are active or suspended in DB2. The command has no effect if the thread is not active or suspended in DB2. If the thread is suspended in VTAM, you can use VTAM commands to terminate the conversations, as described in "Using VTAM commands to cancel threads" on page 412.

A database access thread can also be in the prepared state waiting for the commit decision from the coordinator. When you issue CANCEL THREAD for a database access thread in the prepared state, the thread is converted from active to indoubt. The conversation with the coordinator, and all conversations with downstream participants, are terminated and message DSNL450I is returned. The resources held by the thread are not released until the indoubt state is resolved. This is accomplished automatically by the coordinator or by using the command RECOVER INDOUBT. See "Resolving indoubt units of recovery" on page 459 for more information.

DISPLAY THREAD can be used to determine if a thread is hung in DB2 or VTAM. If in VTAM, there is no reason to use the CANCEL command.

Using CANCEL THREAD requires SYSOPR authority or higher.

When the command is entered at the DB2 subsystem that has a database access thread servicing requests from a DB2 subsystem that owns the allied thread, the database access thread is terminated. Any active SQL request, and all later requests, from the allied thread result in a "resource not available" return code.

To issue this command, enter:

```
-CANCEL THREAD (token)
```

Alternatively, you can use the following version of the command with either the token or LUW ID:

```
-CANCEL DDF THREAD (token or luwid)
```

The *token* is a 1-character to 5-character number that identifies the thread. When DB2 schedules the thread for termination, you will see the following message for a distributed thread:

```
DSNL010I - DDF THREAD token or luwid HAS BEEN CANCELED
```

For a non-distributed thread you will see the following message:

```
DSNV426I - csect THREAD token HAS BEEN CANCELED
```

For more information about CANCEL THREAD, see Chapter 2 of *DB2 Command Reference*.

Diagnostic dumps: CANCEL THREAD allows you to specify that a diagnostic dump be taken.

For more detailed information about diagnosing DDF failures, see Part 3 of *DB2 Diagnosis Guide and Reference*.

Messages: As a result of entering CANCEL THREAD, the following messages can be displayed:

DSNL009I
DSNL010I

DSNL022I

Using VTAM commands to cancel threads

If the command CANCEL THREAD does not terminate the thread, it is possible that it is hung up in VTAM, not in DB2. Use the VTAM VARY NET,TERM command to cancel the thread's VTAM sessions. The VTAM commands only work with SNA VTAM connections, not TCP/IP connections.

To do this, you need to know the VTAM session IDs that correspond to the thread. Follow these steps:

1. Issue the DB2 command DISPLAY THREAD(*nnnn*) LOC(*) DETAIL.

   This gives you the VTAM session IDs that must be canceled. As is shown in the DISPLAY THREAD output in Figure 40, these sessions are identified by the column header SESSID.

```
-DIS THD LOC(*) DETAIL

DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV402I - ACTIVE THREADS -
NAME     ST A   REQ ID           AUTHID    PLAN      ASID    TOKEN
BATCH    TR *     5 BKH2C        SYSADM    BKH2      000D      123
 V436-PGM=*.BKH2C, SEC, STMNT=116
 V445-DB2NET.LUND0.9F6D9F459E92=123 ACCESSING DATA FOR
      USIBMSTODB21:LUND1
 V447--LOCATION         SESSID             A ST TIME
 V448--USIBMSTODB21   v 00D3590EA1E89701     S1 8832108460302
 V448--USIBMSTODB21     00D3590EA1E89822   V R1 8832108460431
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I - DSNVDT '-DIS THD' NORMAL COMPLETION
```

Figure 40. Sample DISPLAY THREAD output

2. Record positions 3 through 16 of SESSID for the threads to be canceled. (In the preceding DISPLAY THREAD output, the values are D3590EA1E89701 and D3590EA1E89822.)
3. Issue the VTAM command DISPLAY NET to display the VTAM session IDs (SIDs). The ones you want to cancel match the SESSIDs in positions 3 through 16. In Figure 41, the corresponding session IDs (DSD3590EA1E89701 and D2D3590EA1E89822) are shown in bold.

```
D NET,ID=LUND0,SCOPE=ACT

IST097I DISPLAY ACCEPTED
IST075I NAME = LUND0, TYPE = APPL
IST486I STATUS= ACTIV, DESIRED STATE= ACTIV
  IST171I ACTIVE SESSIONS = 0000000010, SESSION REQUESTS = 0000
  IST206I SESSIONS:
  IST634I NAME     STATUS           SID          SEND   RECV
  IST635I LUND1    ACTIV-S     D24B171032B76E65  0051   0043
  IST635I LUND1    ACTIV-S     D24B171032B32545  0051   0043
  IST635I LUND1    ACTIV-S     D24B171032144565  0051   0043
  IST635I LUND1    ACTIV-S     D24B171032B73465  0051   0043
  IST635I LUND1    ACTIV-S     D24B171032B88865  0051   0043
  IST635I LUND1    ACTIV-R     D2D3590EA1E89701  0022   0031
  IST635I LUND1    ACTIV-R     D2D3590EA1E89802  0022   0031
  IST635I LUND1    ACTIV-R     D2D3590EA1E89809  0022   0031
  IST635I LUND1    ACTIV-R     D2D3590EA1E89821  0022   0031
  IST635I LUND1    ACTIV-R     D2D3590EA1E89822  0022   0031
  IST314I END
```

Figure 41. Sample output for VTAM DISPLAY NET command

4. Issue the VTAM command VARY NET,TERM SID= for each of the VTAM SIDs associated with the DB2 thread. For more information about VTAM commands, see *VTAM for MVS/ESA Operation*.

Monitoring and controlling stored procedures

Stored procedures are user-written SQL programs that run at a DB2 server. Stored procedures can run in DB2-established or WLM-established address spaces. To monitor and control stored procedures in WLM-established address spaces, you might need to use WLM commands, rather than DB2 commands. When you execute a WLM command on an z/OS system that is part of a Sysplex, the scope of that command is the Sysplex.

This section discusses the following topics:
- "Displaying information about stored procedures and their environment"
- "Refreshing the environment for stored procedures or user-defined functions" on page 415
- "Obtaining diagnostic information about stored procedures" on page 415

For more information about stored procedures, see Part 6 of *DB2 Application Programming and SQL Guide*.

Displaying information about stored procedures and their environment

Use the DB2 commands DISPLAY PROCEDURE and DISPLAY THREAD to obtain information about a stored procedure while it is running. In the WLM-established environment, use the z/OS command DISPLAY WLM to obtain information about the application environment in which a stored procedure runs.

The DB2 command DISPLAY PROCEDURE: This command can display the following information about stored procedures:
- Status (started, stop-queue, stop-reject, stop-abend)
- Number of requests currently running and queued
- Maximum number of threads running a stored procedure load module and queued
- Count of timed-out SQL CALLs

The following command displays information about all stored procedures in all schemas that have been accessed by DB2 applications:

```
-DISPLAY PROCEDURE
DSNX940I csect - DISPLAY PROCEDURE REPORT FOLLOWS-

------ SCHEMA=PAYROLL
PROCEDURE          STATUS    ACTIVE  QUED  MAXQ  TIMEOUT  FAIL  WLM_ENV
PAYRPRC1
                   STARTED        0     0     1       0      0  PAYROLL
PAYRPRC2
                   STOPQUE        0     5     5       3      0  PAYROLL
PAYRPRC3
                   STARTED        2     0     6       0      0  PAYROLL
USERPRC4
                   STOPREJ        0     0     1       0      1  SANDBOX

------ SCHEMA=HRPROD
PROCEDURE          STATUS    ACTIVE  QUED  MAXQ  TIMEOUT  FAIL  WLM_ENV
HRPRC1
                   STARTED        0     0     1       0      1  HRPROCS
HRPRC2
                   STOPREJ        0     0     1       0      0  HRPROCS
DISPLAY PROCEDURE REPORT COMPLETE
DSN9022I  = DSNX9COM '-DISPLAY PROC' NORMAL COMPLETION
```

This example shows two schemas (PAYROLL and HRPROD) that have been accessed by DB2 applications. You can also display information about specific stored procedures.

The DB2 command DISPLAY THREAD: This command tells whether:
- A thread is waiting for a stored procedure to be scheduled
- A thread is executing within a stored procedure

Here is an example of DISPLAY THREAD output that shows a thread that is executing a stored procedure:

```
!display thread(*) det
DSNV401I ! DISPLAY THREAD REPORT FOLLOWS -
DSNV402I ! ACTIVE THREADS -
NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
BATCH    SP       3 CALLWLM      SYSADM   PLNAPPLX 0022     5
  V436-PGM=*.MYPROG, SEC=2, STMNT=1
  V429 CALLING PROCEDURE=SYSADM  .WLMSP             ,
       PROC=V61AWLM1, ASID=0085, WLM_ENV=WLMENV1
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I ! DSNVDT '-DIS THD' NORMAL COMPLETION
```

The SP status indicates that the thread is executing within the stored procedure. An SW status indicates that the thread is waiting for the stored procedure to be scheduled.

Here is an example of DISPLAY THREAD output that shows a thread that is executing a user-defined function:

```
!display thd(*) det
DSNV401I ! DISPLAY THREAD REPORT FOLLOWS -
DSNV402I ! ACTIVE THREADS -
NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
BATCH    SP      27 LI33FN1      SYSADM   DSNTEP3  0021     4
 V436-PGM=*.MYPROG, SEC=2, STMNT=1
 V429 CALLING FUNCTION =SYSADM  .FUNC1             ,
      PROC=V61AWLM1, ASID=0085, WLM_ENV=WLMENV1
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I ! DSNVDT '-DISPLAY THD' NORMAL COMPLETION
```

The z/OS command DISPLAY WLM: Use the command DISPLAY WLM to determine the status of an application environment in which a stored procedure runs. The output from DISPLAY WLM lets you determine whether a stored procedure can be scheduled in an application environment.

For example, you can issue this command to determine the status of application environment WLMENV1:

```
D WLM,APPLENV=WLMENV1
```

You might get results like this:

```
IWM029I  15.22.22  WLM DISPLAY
APPLICATION ENVIRONMENT NAME     STATE     STATE DATA
WLMENV1                          AVAILABLE
ATTRIBUTES: PROC=DSNWLM1 SUBSYSTEM TYPE: DB2
```

The output tells you that WLMENV1 is available, so WLM can schedule stored procedures for execution in that environment.

Refreshing the environment for stored procedures or user-defined functions

Depending on what has changed in a stored procedures environment, you might need to perform one or more of these tasks:

- Refresh Language Environment®.

  Do this when someone has modified a load module for a stored procedure, and that load module is cached in a stored procedures address space. When you refresh Language Environment, the cached load module is purged. On the next invocation of the stored procedure, the new load module is loaded.
- Restart a stored procedures address space.

  You might stop and then start a stored procedures address space because you need to make a change to the startup JCL for a stored procedures address space. You might need to start a stored procedures address space because the address space has abnormally terminated.

The method that you use to perform these tasks for stored procedures depends on whether you are using WLM-established or DB2-established address spaces.

For DB2-established address spaces: Use the DB2 commands START PROCEDURE and STOP PROCEDURE to perform all of these tasks.

For WLM-established address spaces: You use the VARY WLM command to complete the following tasks,

- To refresh the Language Environment when you need to load a new version of a stored procedure, use the z/OS command

  ```
  VARY WLM,APPLENV=name,REFRESH
  ```

  In this command, *name* represents the name of a WLM application environment that is associated with a group of stored procedures. You affect all stored procedures that are associated with the application environment when you refresh the Language Environment.
- To stop all stored procedures address spaces that are associated with WLM application environment *name*, use the following z/OS command:

  ```
  VARY WLM,APPLENV=name,QUIESCE
  ```
- To start all stored procedures address spaces that are associated with WLM application environment *name*, use the following z/OS command:

  ```
  VARY WLM,APPLENV=name,RESUME
  ```

  You also need to use the VARY WLM command with the RESUME option when WLM puts an application environment in the unavailable state. An application environment in which stored procedures run becomes unavailable when WLM detects five abnormal terminations within 10 minutes. When an application environment is in the unavailable state, WLM does not schedule stored procedures for execution in it.

See *z/OS MVS Planning: Workload Management* for more information about the command VARY WLM.

Obtaining diagnostic information about stored procedures

If the startup procedures for your stored procedures address spaces contain a DD statement for CEEDUMP, Language Environment writes a small diagnostic dump to CEEDUMP when a stored procedure terminates abnormally. The output waits to print until the stored procedures address space terminates.

You can obtain the dump information by stopping the stored procedures address space in which the stored procedure is running. See “Refreshing the environment for stored procedures or user-defined functions” on page 415 for information about how to stop and start stored procedures address spaces in the DB2-established and WLM-established environments.

Using NetView to monitor errors

The NetView® program lets you have a single focal point from which to view problems in the network. DDF sends an alert to NetView when a remote location is either involved in the cause of the failure or affected by the failure. The following major events generate alerts:

- Conversation failures
- Distributed security failures
- DDF abends
- DDM protocol errors
- Database access thread abends
- Distributed allied thread abends

Alerts for DDF are displayed on NetView’s Hardware Monitor panels and are logged in the hardware monitor database. Figure 42 is an example of the Alerts-Static panel in NetView.

```
N E T V I E W          SESSION DOMAIN: CNM01    OPER2     11/03/89 10:29:55
NPDA-30B                     * ALERTS-STATIC *

SEL# DOMAIN RESNAME TYPE TIME  ALERT DESCRIPTION:PROBABLE CAUSE
( 1) CNM01 AS      *RQST 09:58 SOFTWARE PROGRAM ERROR:COMM/REMOTE NODE
( 2) CNM01 AR      *SRVR 09:58 SOFTWARE PROGRAM ERROR:SNA COMMUNICATIONS
( 3) CNM01 P13008   CTRL 12:11 LINK ERROR:REMOTE DCE INTERFACE CABLE         +
( 4) CNM01 P13008   CTRL 12:11 RLSD OFF DETECTED:OUTBOUND LINE
( 5) CNM01 P13008   CTRL 12:11 LINK ERROR:REMOTE DCE INTERFACE CABLE         +
( 6) CNM01 P13008   CTRL 12:11 LINK ERROR:INBOUND LINE                       +
( 7) CNM01 P13008   CTRL 12:10 LINK ERROR:REMOTE DCE INTERFACE CABLE         +
( 8) CNM01 P13008   CTRL 12:10 LINK ERROR:REMOTE DCE INTERFACE CABLE         +
( 9) CNM01 P13008   CTRL 12:10 LINK ERROR:INBOUND LINE                       +
(10) CNM01 P13008   CTRL 12:10 LINK ERROR:REMOTE DCE INTERFACE CABLE         +
(11) CNM01 P13008   CTRL 12:10 LINK ERROR:REMOTE DCE INTERFACE CABLE         +
(12) CNM01 P13008   CTRL 12:10 LINK ERROR:REMOTE DCE INTERFACE CABLE         +
(13) CNM01 P13008   CTRL 12:10 LINK ERROR:REMOTE DCE INTERFACE CABLE         +
(14) CNM01 P13008   CTRL 12:10 LINK ERROR:REMOTE DCE INTERFACE CABLE         +
(15) CNM01 P13008   CTRL 12:10 LINK ERROR:REMOTE DCE INTERFACE CABLE         +
 PRESS ENTER KEY TO VIEW ALERTS-DYNAMIC OR ENTER A TO VIEW ALERTS-HISTORY
 ENTER SEL# (ACTION),OR SEL# PLUS M (MOST RECENT), P (PROBLEM), DEL (DELETE)
```

Figure 42. Alerts-static panel in NetView. DDF errors are denoted by the resource name AS (server) and AR (requester). For DB2-only connections, the resource names would be RS (server) and RQ (requester).

To see the recommended action for solving a particular problem, enter the selection number, and then press ENTER. This displays the Recommended Action for Selected Event panel shown in Figure 43 on page 417.

```
N E T V I E W          SESSION DOMAIN: CNM01    OPER2     11/03/89 10:30:06
NPDA-45A           * RECOMMENDED ACTION FOR SELECTED EVENT *     PAGE  1 OF  1
 CNM01       AR  1           AS  2
            +--------+   +--------+
 DOMAIN     ³  RQST  ³---³  SRVR  ³
            +--------+   +--------+

USER    CAUSED - NONE

INSTALL CAUSED - NONE

FAILURE CAUSED - SNA COMMUNICATIONS ERROR:
                     RCPRI=0008 RCSEC=0001                1
                FAILURE OCCURRED ON RELATIONAL DATA BASE USIBMSTODB21
       ACTIONS - I008 - PERFORM PROBLEM DETERMINATION PROCEDURE FOR REASON
                 CODE  3 00D31029                   2
                I168 - FOR RELATIONAL DATA BASE USIBMSTODB22
                REPORT THE FOLLOWING LOGICAL UNIT OF WORK IDENTIFIER
                 DB2NET.LUND0.A1283FFB0476.0001

ENTER  DM  (DETAIL MENU) OR  D  (EVENT DETAIL)
```

Figure 43. Recommended action for selected event panel in NetView. In this example, the AR (USIBMSTODB21) is reporting the problem, which is affecting the AS (USIBMSTODB22).

Key	Description
1	The system reporting the error. The system reporting the error is always on the left side of the panel. That system's name appears first in the messages. Depending on who is reporting the error, either the LUNAME or the location name is used.
2	The system affected by the error. The system affected by the error is always displayed to the right of the system reporting the error. The affected system's name appears second in the messages. Depending on what type of system is reporting the error, either the LUNAME or the location name is used. If no other system is affected by the error, then this system will not appear on the panel.
3	DB2 reason code. For information about DB2 reason codes, see Part 3 of *DB2 Codes*. For diagnostic information, see Part 3 of *DB2 Diagnosis Guide and Reference*.

For more information about using NetView, see *Tivoli NetView for z/OS User's Guide*.

Stopping the DDF

General-use Programming Interface

You need SYSOPR authority or higher to stop the distributed data facility. Use one of the following commands:

```
-STOP DDF MODE (QUIESCE)
-STOP DDF MODE (FORCE)
```

Use the QUIESCE option whenever possible; it is the default. With QUIESCE, the STOP DDF command does not complete until all VTAM or TCP/IP requests have completed. In this case, no resynchronization work is necessary when you restart DDF. If there are indoubt units of work that require resynchronization, the QUIESCE option produces message DSNL035I. Use the FORCE option only when you must stop DDF quickly. Restart times are longer if you use FORCE.

When DDF is stopped with the FORCE option, and DDF has indoubt thread responsibilities with remote partners, one or both of messages DSNL432I and DSNL433I is generated.

DSNL432I shows the number of threads that DDF has coordination responsibility over with remote participants who could have indoubt threads. At these participants, database resources that are unavailable because of the indoubt threads remain unavailable until DDF is started and resolution occurs.

DSNL433I shows the number of threads that are indoubt locally and need resolution from remote coordinators. At the DDF location, database resources are unavailable because the indoubt threads remain unavailable until DDF is started and resolution occurs.

To force the completion of outstanding VTAM or TCP/IP requests, use the FORCE option, which cancels the threads associated with distributed requests.

When the FORCE option is specified with STOP DDF, database access threads in the prepared state that are waiting for the commit or abort decision from the coordinator are logically converted to the indoubt state. The conversation with the coordinator is terminated. If the thread is also a coordinator of downstream participants, these conversations are terminated. Automatic indoubt resolution is initiated when DDF is restarted. See “Resolving indoubt units of recovery” on page 459 for more information about this topic.

The STOP DDF command causes the following messages to appear:

```
DSNL005I - DDF IS STOPPING
DSNL006I - DDF STOP COMPLETE
```

If the distributed data facility has already been stopped, the STOP DDF command fails and message `DSNL002I - DDF IS ALREADY STOPPED` appears.

Stopping DDF using VTAM commands: Another way to force DDF to stop is to issue the VTAM VARY NET,INACT command. This command makes VTAM unavailable and terminates DDF. VTAM forces the completion of any outstanding VTAM requests immediately.

The syntax for the command is as follows:

```
VARY NET,INACT,ID=db2lu,FORCE
```

where *db2lu* is the VTAM LU name for the local DB2 system.

When DDF has stopped, the following command must be issued before START DDF can be attempted:

```
VARY NET,ACT,ID=db2lu
```

End of General-use Programming Interface

Controlling traces

These traces can be used for problem determination:

- DB2 trace
- IMS attachment facility trace
- CICS trace
- Three TSO attachment facility traces

CAF trace stream
RRS trace stream
z/OS component trace used for IRLM

Controlling the DB2 trace

General-use Programming Interface

DB2 trace allows you to trace and record subsystem data and events. There are five different types of trace. For classes of events traced by each type see the description of the START TRACE command in Chapter 2 of *DB2 Command Reference*. For more information about the trace output produced, see Appendix D, "Interpreting DB2 trace output," on page 1135. In brief, DB2 records the following types of data:

Statistics
Data that allows you to conduct DB2 capacity planning and to tune the entire set of DB2 programs.

Accounting
Data that allows you to conduct DB2 capacity planning and to tune the entire set of DB2 programs.

Performance
Data about subsystem events, which can be used to do program, resource, user, and subsystem-related tuning.

Audit Data that can be used to monitor DB2 security and access to data.

Monitor
Data that can be used to monitor DB2 security and access to data.

DB2 provides commands for controlling the collection of this data. To use the trace commands you must have one of the following types of authority:
- SYSADM or SYSOPR authority
- Authorization to issue start and stop trace commands (the TRACE privilege)
- Authorization to issue the display trace command (the DISPLAY privilege).

The trace commands include:

START TRACE
Invokes one or more different types of trace.

DISPLAY TRACE
Displays the trace options that are in effect.

STOP TRACE
Stops any trace that was started by either the START TRACE command or the parameters specified when installing or migrating.

MODIFY TRACE
Changes the trace events (IFCIDs) being traced for a specified active trace.

Several parameters can be specified to further qualify the scope of a trace. Specific events within a trace type can be traced as well as events within specific DB2 plans, authorization IDs, resource manager IDs and location. The destination to which trace data is sent can also be controlled. For a discussion of trace commands, see Chapter 2 of *DB2 Command Reference*.

When you install DB2, you can request that any trace type and class start automatically when DB2 starts. For information about starting traces automatically, see Part 2 of *DB2 Installation Guide*.

End of General-use Programming Interface

Diagnostic traces for attachment facilities

The following trace facilities are for diagnostic purposes only:

- IMS provides a trace facility that shows the flow of requests across the connections from the control and dependent regions to DB2. The trace is recorded on the IMS log if the appropriate options are specified, and then it is printed with DFSERA10 plus a formatting exit module. For more information about this trace facility, see *IMS Utilities Reference: System*.

  In addition, the IMS attachment facility of DB2 provides an internal wrap-around trace table that is always active. When certain unusual error conditions occur, these trace entries are externalized on the IMS log.
- You can use the CICS trace facility to trace the CICS attachment facility.

  Use the transaction CETR to control the CICS trace facility. CETR gives you a series of menus that you can use to set CICS trace options. For CICS 4.1 and later, to trace the CICS attachment facility, set these values in the Component Trace Options panel:
  - For CICS 4.1, specify the value 2 in the FC field.
  - For later releases, specify the value 2 in the RI field.

  For information about using the CETR transaction to control CICS tracing, see *CICS Transaction Server for z/OS CICS Supplied Transactions*.
- The TSO attachment facility provides three tracing mechanisms:
  The DSN trace stream
  The CLIST trace facility
  The SPUFI trace stream
- The call attachment facility trace stream uses the same *ddname* as the TSO DSN trace stream, but is independent of TSO.
- The RRSAF trace stream uses the same *ddname* as the TSO DSN trace stream, but is independent of TSO. An RRSAF internal trace will be included in any ABEND dump produced by RRSAF. This tracing facility provides a history of RRSAF usage that can aid in diagnosing errors in RRSAF.

Diagnostic trace for the IRLM

The following z/OS commands control diagnostic traces for the IRLM:

MODIFY *irlmproc*,**SET,TRACE**
: Sets dynamically the maximum number of trace buffers for each trace type. This value is used only when the external component trace writer is not activated.

MODIFY *irlmproc*,**STATUS,TRACE**
: Displays the status of traces and the number of trace buffers used for each trace type. Also displays whether or not the external component trace writer is active for the trace.

START *irlmproc*,**TRACE=YES**
: Captures traces in wrap-around IRLM buffers at IRLM startup.

TRACE CT
Starts, stops, or modifies a diagnostic trace for IRLM. The TRACE CT command does not know about traces that are started automatically during IRLM startup.

Recommendations:

- Do not use the external component trace writer to write traces to the data set.
- Activate all traces during IRLM startup. Use the command START *irlmproc*,TRACE=YES to activate all traces.

See Chapter 2 of *DB2 Command Reference* for detailed information.

Controlling the resource limit facility (governor)

General-use Programming Interface

The governor allows the system administrator to limit the amount of time permitted for the execution of the SELECT, UPDATE, DELETE, and INSERT dynamic SQL statements.

DB2 provides these commands for controlling the governor:

START RLIMIT
Starts the governor and identifies a resource limit specification table. You can also use START RLIMIT to switch resource limit specification tables.

DISPLAY RLIMIT
Displays the current status of the governor. If the governor has been started, it also identifies the resource limit specification table.

STOP RLIMIT
Stops the governor and removes any set limits.

The limits are defined in resource limit specification tables and can vary for different users. One resource limit specification table is used for each invocation of the governor and is identified on the START RLIMIT command.

See “Resource limit facility (governor)” on page 716 for more information about the governor.

End of General-use Programming Interface

When you install DB2, you can request that the governor start automatically when DB2 starts. For information about starting the governor automatically, see Part 2 of *DB2 Installation Guide*.

Changing subsystem parameter values

You can modify the values of subsystem parameters dynamically even while DB2 is running by using the following procedure:

1. Run the installation process in UPDATE mode, specifying any new parameter values. This process produces a new DSNTIJUZ job with the new values; it also saves these values in the file specified as the output member name on panel DSNTIPA1.

2. Assemble and link-edit the DSNTIJUZ job produced in Step 1, and then submit the job to create the new load module with the new subsystem parameter values.
3. Issue the SET SYSPARM command to change the subsystem parameters dynamically:

   ```
   SET SYSPARM LOAD(load-module-name)
   ```

   where you specify the *load-module-name* to be the same as the output member name in Step 1.

   If you want to specify the load module name that is used during DB2 start up, you can issue the following command:

   ```
   SET SYSPARM RELOAD
   ```

   For more information, see Part 2 of *DB2 Installation Guide* and Chapter 2 of *DB2 Command Reference*.

Chapter 18. Managing the log and the bootstrap data set

The DB2 log registers data changes and significant events as they occur. The bootstrap data set (BSDS) is a repository of information about the data sets that contain the log.

DB2 writes each log record to a disk data set called the *active log*. When the active log is full, DB2 copies its contents to a disk or tape data set called the *archive log*. That process is called *offloading*. This chapter describes:

"How database changes are made"
"Establishing the logging environment" on page 425
"Managing the bootstrap data set (BSDS)" on page 436
"Discarding archive log records" on page 438

For information about the physical and logical records that make up the log, see Appendix C, "Reading log records," on page 1111. That appendix also contains information about how to write a program to read log records.

How database changes are made

Before you can fully understand how logging works, you need to be familiar with how database changes are made to ensure consistency. This section discusses, *units of recovery* and *rollbacks*.

Units of recovery

A *unit of recovery* is the work, done by a single DB2 DBMS for an application, that changes DB2 data from one point of consistency to another. A *point of consistency* (also, *sync point* or *commit point*) is a time when all recoverable data that an application program accesses is consistent with other data. (For an explanation of maintaining consistency between DB2 and another subsystem such as IMS or CICS, see "Multiple system consistency" on page 455.)

A unit of recovery begins with the first change to the data after the beginning of the job or following the last point of consistency and ends at a later point of consistency. An example of units of recovery within an application program is shown in Figure 44.

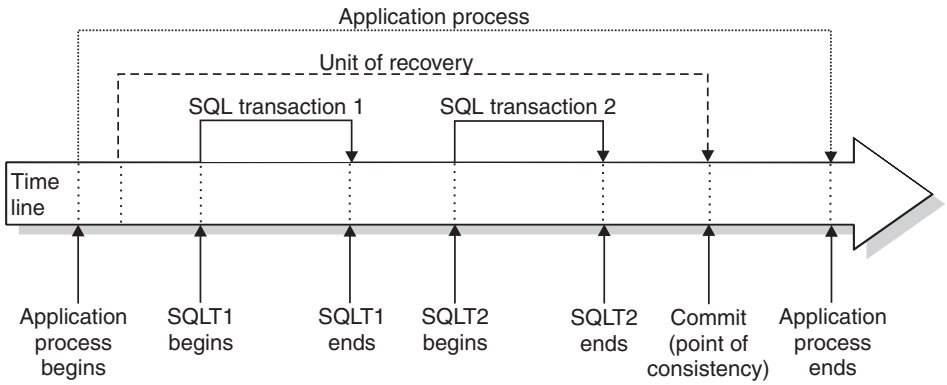


Figure 44. A unit of recovery within an application process

© Copyright IBM Corp. 1982, 2008

In this example, the application process makes changes to databases at SQL transaction 1 and 2. The application process can include a number of units of recovery or just one, but any complete unit of recovery ends with a commit point.

For example, a bank transaction might transfer funds from account A to account B. First, the program subtracts the amount from account A. Next, it adds the amount to account B. After subtracting the amount from account A, the two accounts are inconsistent. These accounts are inconsistent until the amount is added to account B. When both steps are complete, the program can announce a point of consistency and thereby make the changes visible to other application programs.

Normal termination of an application program automatically causes a point of consistency. The SQL COMMIT statement causes a point of consistency during program execution under TSO. A sync point causes a point of consistency in CICS and IMS programs.

Rolling back work

If failure occurs within a unit of recovery, DB2 backs out any changes to data, returning the data to its state at the start of the unit of recovery; that is, DB2 *undoes* the work. The events are shown in Figure 45. The SQL ROLLBACK statement, and deadlocks and timeouts (reported as SQLCODE -911, SQLSTATE 40001), cause the same events.

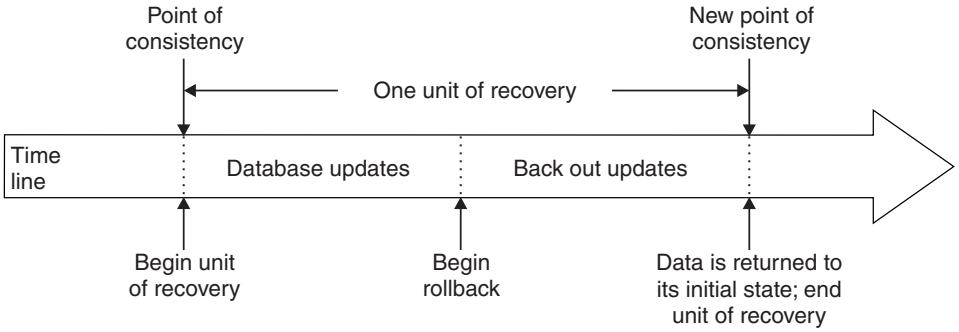


Figure 45. Unit of recovery (rollback)

The effects of inserts, updates, and deletes to large object (LOB) values are backed out along with all the other changes made during the unit of work being rolled back, even if the LOB values that were changed reside in a LOB table space with the LOG NO attribute.

An operator or an application can issue the CANCEL THREAD command with the NOBACKOUT option to cancel long running threads without backing out data changes. DB2 backs out changes to catalog and directory tables regardless of the NOBACKOUT option. As a result, DB2 does not read the log records and does not write or apply the compensation log records. After CANCEL THREAD NOBACKOUT processing, DB2 marks all objects associated with the thread as refresh pending (REFP) and puts the objects in a logical page list (LPL). For information about how to reset the REFP status, see *DB2 Utility Guide and Reference*.

The NOBACKOUT request might fail for either of the following two reasons:

- DB2 does not completely back out updates of the catalog or directory (message DSNI032I with reason 00C900CC).
- The thread is part of a global transaction (message DSNV439I).

Establishing the logging environment

The DB2 logging environment is established by using installation panels to specify options, such as whether to have dual active logs (strongly recommended), what media to use for archive log volumes, and how many log buffers to have. For details of the installation process, see Part 2 of *DB2 Installation Guide*.

Creation of log records

Log records typically go through the following cycle:

1. DB2 registers changes to data and significant events in recovery log records.
2. DB2 processes recovery log records and breaks them into segments if necessary.
3. Log records are placed sequentially in *output log buffers*, which are formatted as VSAM control intervals (CIs). Each log record is identified by a continuously increasing RBA in the range 0 to 2^{48}-1, where 2^{48} represents 2 to the 48th power. (In a data sharing environment, a log record sequence number (LRSN) is used to identify log records. See *DB2 Data Sharing: Planning and Administration* for more information.)
4. The CIs are written to a set of predefined disk *active log data sets*, which are used sequentially and recycled.
5. As each active log data set becomes full, its contents are automatically *offloaded* to a new *archive log data set*.

If you change or create data that is compressed, the data logged is also compressed. Changes to compressed rows like inserts, updates, and deletes are also logged as compressed data.

Retrieval of log records

Log records are retrieved through the following events:

1. A log record is requested using its RBA.
2. DB2 searches for the log record in the following locations in the order they are presented:
   a. The log buffers.
   b. The active logs. The bootstrap data set registers which log RBAs apply to each active or archive log data set. If the record is in an active log, DB2 dynamically acquires a buffer, reads one or more CIs, and returns one record for each request.
   c. The archive logs. DB2 determines which archive volume contains the CIs, dynamically allocates the archive volume, acquires a buffer, and reads the CIs.

Writing the active log

The log buffers are written to an active log data set when they become full, when the write threshold is reached (as specified on the DSNTIPL panel), or, more often, when the DB2 subsystem forces the log buffer to be written (such as at commit time). In the last case, the same control interval can be written several times to the same location.

Be sure to set your ZPARMs are set so that there are enough log buffers to avoid the need to wait for a buffer to become available (DSN6LOGP OUTBUFF parameter). Switching log data sets may also cause a temporary performance impact when the switch takes place and the associated recovery checkpoint is taken. This can be minimized by ensuring that the active log data sets are large

enough to avoid frequent switching. In addition, some events can cause log buffers to be written before the ZPARM-defined threshold has been reached. These events include, but are not limited to:

- Phase 1 commit
- Abort processing
- GBP-dependent index split
- Mass delete in a data-sharing environment
- Use of GBPCACHE NO
- All log buffers being filled

Consider the probable frequency of these events when you determine how often to commit changes.

When DB2 is initialized, the active log data sets named in the BSDS are dynamically allocated for *exclusive* use by DB2 and remain allocated exclusively to DB2 (the data sets were allocated as DISP=OLD) until DB2 terminates. Those active log data sets cannot be replaced, nor can new ones be added, without terminating and restarting DB2. The size and number of log data sets is indicated by what was specified by installation panel DSNTIPL. The use of dual active logs increases availability as well as the reliability of recovery by eliminating a single point of failure.

Writing the archive log (offloading)

The process of copying active logs to archive logs is called *offloading*. The relation of offloading to other logging events is shown schematically in Figure 46.

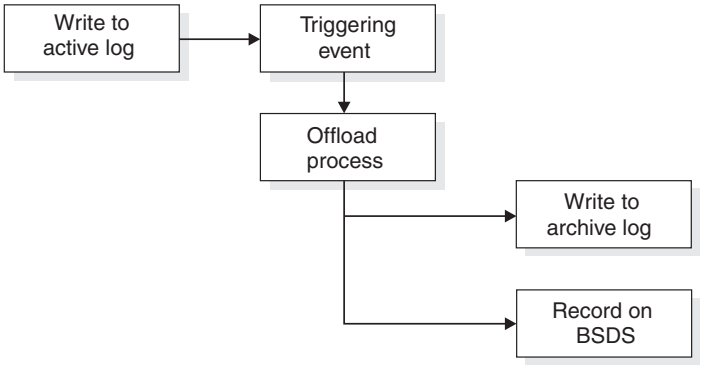


Figure 46. The offloading process

Triggering offload

An offload of an active log to an archive log can be triggered by several events. The most common are when:

- An active log data set is full
- Starting DB2 and an active log data set is full
- The command ARCHIVE LOG is issued

An offload is also triggered by two uncommon events:

- An error occurring while writing to an active log data set. The data set is truncated before the point of failure, and the record that failed to write becomes the first record of the next data set. An offload is triggered for the truncated data

set as in normal end-of-file. If there are dual active logs, both copies are truncated so the two copies remain synchronized.

- Filling of the last unarchived active log data set. Message DSNJ110E is issued, stating the percentage of its capacity in use; IFCID trace record 0330 is also issued if statistics class 3 is active. If all active logs become full, DB2 stops processing until offloading occurs and issues this message:

  ```
  DSNJ111E - OUT OF SPACE IN ACTIVE LOG DATA SETS
  ```

The offloading process

During the process, DB2 determines which data set to offload. Using the last log RBA offloaded, as registered in the BSDS, DB2 calculates the log RBA at which to start. DB2 also determines the log RBA at which to end, from the RBA of the last log record in the data set, and registers that RBA in the BSDS.

When all active logs become full, the DB2 subsystem runs an offload and halts processing until the offload is completed. If the offload processing fails when the active logs are full, then DB2 cannot continue doing any work that requires writing to the log. For additional information, see "Active log failure recovery" on page 531.

Offload operator request: When an active log is ready to be offloaded, a request can be sent to the z/OS console operator to mount a tape or prepare a disk unit. The value of the field WRITE TO OPER of the DSNTIPA installation panel determines whether the request is received. If the value is YES, the request is preceded by a WTOR (message number DSNJ008E) informing the operator to prepare an archive log data set for allocating.

The operator need not respond to message DSNJ008E immediately. However, delaying the response delays the offload process. It does not affect DB2 performance unless the operator delays response for so long that DB2 runs out of active logs.

The operator can respond by canceling the offload. In that case, if the allocation is for the first copy of dual archive data sets, the offload is merely delayed until the next active log data set becomes full. If the allocation is for the second copy, the archive process switches to single copy mode, but for the one data set only.

Delay of log offload task: When DB2 switches active logs and finds that the offload task has been active since the last log switch, it issues the following message to notify the operator that there might be an outstanding tape mount or some other problem that prevents the offload of the previous active log data set.

```
DSNJ017E - csect-name WARNING - OFFLOAD TASK HAS BEEN ACTIVE SINCE
           date-time AND MAY HAVE STALLED
```

DB2 continues processing. Issue the ARCHIVE LOG CANCEL OFFLOAD command to cancel and restart the offload task. If the offload task remains stalled, the active logs eventually become full and DB2 stops database update activity. To view the status of the offload task, issue the DISPLAY LOG command.

Messages returned during offloading: The following messages are sent to the z/OS console by DB2 and the offload process. With the exception of the DSNJ139I message, these messages can be used to find the RBA ranges in the various log data sets.

- The following message appears during DB2 initialization when the current active log data set is found, and after a data set switch. During initialization, the

STARTRBA value in the message does not refer to the beginning of the data set, but to the position in the log where logging will begin.

```
DSNJ001I - csect-name CURRENT COPY n ACTIVE LOG DATA SET IS
           DSNAME=...,  STARTRBA=..., ENDRBA=...
```

- The following message appears when an active data set is full:

```
DSNJ002I - FULL ACTIVE LOG DATA SET DSNAME=...,
           STARTRBA=..., ENDRBA=...
```

- The following message appears when offload reaches end-of-volume or end-of-data-set in an archive log data set:

  Non-data sharing version is:

```
DSNJ003I - FULL ARCHIVE LOG VOLUME DSNAME=..., STARTRBA=..., ENDRBA=...,
           STARTTIME=..., ENDTIME=..., UNIT=..., COPYnVOL=...,
           VOLSPAN=..., CATLG=...
```

  Data sharing version is:

```
DSNJ003I - FULL ARCHIVE LOG VOLUME DSNAME=..., STARTRBA=..., ENDRBA=...,
           STARTLRSN=..., ENDLRSN=..., UNIT=..., COPYnVOL=...,
           VOLSPAN=..., CATLG=...
```

- The following message appears when one data set of the next pair of active logs is not available because of a delay in offloading, and logging continues on one copy only:

```
DSNJ004I - ACTIVE LOG COPY n INACTIVE, LOG IN SINGLE MODE,
           ENDRBA=...
```

- The following message appears when dual active logging resumes after logging has been carried on with one copy only:

```
DSNJ005I - ACTIVE LOG COPY n IS ACTIVE, LOG IN DUAL MODE,
           STARTRBA=...
```

- The following message indicates that the offload task has ended:

```
DSNJ139I LOG OFFLOAD TASK ENDED
```

Interruptions and errors while offloading: Here is how DB2 handles the following interruptions in the offloading process:

- The command STOP DB2 does not take effect until offloading is finished.
- A DB2 failure during offload causes offload to begin again from the previous start RBA when DB2 is restarted.
- Offload handling of read I/O errors on the active log is described under "Active log failure recovery" on page 531, or write I/O errors on the archive log, under "Archive log failure recovery" on page 535.
- An unknown problem that causes the offload task to *hang* means that DB2 cannot continue processing the log. This problem might be resolved by retrying the offload, which you can do by using the option CANCEL OFFLOAD of the command ARCHIVE LOG, described in "Canceling log off-loads" on page 432.

Archive log data sets

Archive log data sets can be placed on standard label tapes or disks and can be managed by DFSMShsm (Data Facility Hierarchical Storage Manager). They are always written by QSAM. Archive logs on tape are read by BSAM; those on disk are read by BDAM. Each z/OS logical record in an archive log data set is a VSAM CI from the active log data set. The block size is a multiple of 4 KB. (For more information, see installation panel DSNTIPA in Part 2 of *DB2 Installation Guide*.)

Output archive log data sets are dynamically allocated, with names chosen by DB2. The data set name prefix, block size, unit name, and disk sizes needed for allocation are specified when DB2 is installed, and recorded in the DSNZP*xxx*

module. You can also choose, at installation time, to have DB2 add a date and time to the archive log data set name. See installation panel DSNTIPH in Part 2 of *DB2 Installation Guide* for more information.

Restrictions: Consider the following restrictions for archive log data sets and volumes:

- You cannot specify specific volumes for new archive logs. If allocation errors occur, offloading is postponed until the next time loading is triggered.
- Do not use partitioned data set extended (PDSE) for archive log data. PDSEs are not supported for archive logs.

Using dual archive logging: If you specify dual archive logs at installation time, each log CI retrieved from the active log is written to two archive log data sets. The log records that are contained on a pair of dual archive log data sets are identical, but end-of-volumes are not synchronized for multivolume data sets.

Archiving to disk offers faster recoverability but is more expensive than archiving to tape. If you use dual logging, you can specify on installation panel DSNTIPA that the primary copy of the archive log go to disk and the secondary copy go to tape.

This feature increases recovery speed without using as much disk. The second tape is intended as a backup or can be sent to a remote site in preparation for disaster recovery. To make recovering from the COPY2 archive tape faster at the remote site, use the new installation parameter, ARC2FRST, to specify that COPY2 archive log should be read first. Otherwise, DB2 always attempts to read the primary copy of the archive log data set first.

Archiving to tape: If the unit name reflects a tape device, DB2 can extend to a maximum of twenty volumes. DB2 passes a file sequence number of 1 on the catalog request for the first file on the next volume. Though that might appear to be an error in the integrated catalog facility catalog, it causes no problems in DB2 processing.

If you choose to offload to tape, consider adjusting the size of your active log data sets such that each set contains the amount of space that can be stored on a nearly full tape volume. That adjustment minimizes tape handling and volume mounts and maximizes the use of tape resources. However, such an adjustment is not always necessary.

If you want the active log data set to fit on one tape volume, consider placing a copy of the BSDS on the same tape volume as the copy of the active log data set. Adjust the size of the active log data set downward to offset the space required for the BSDS.

Archiving to disk volumes: All archive log data sets allocated on disk must be cataloged. If you choose to archive to disk, then the field CATALOG DATA of installation panel DSNTIPA must contain YES. If this field contains NO, and you decide to place archive log data sets on disk, you receive message DSNJ072E each time an archive log data set is allocated, although the DB2 subsystem still catalogs the data set.

If you use disk storage, be sure that the primary and secondary space quantities and block size and allocation unit are large enough so that the disk archive log data set does not attempt to extend beyond 15 volumes. That minimizes the possibility of unwanted z/OS B37 or E37 abends during the offload process.

Primary space allocation is set with the PRIMARY QUANTITY field of the DSNTIPA installation panel. The primary space quantity must be less than 64K tracks because of the DFSMS Direct Access Device Space Management limit of 64K tracks on a single volume when allocating a sequential disk data set.

Using SMS to manage archive log data sets: You can use DFSMS (Data Facility Storage Management Subsystem) to manage archive log data sets. When archiving to disk, DB2 uses the number of online storage volumes for the specified unit to determine a count of candidate volumes, up to a maximum of 15 volumes. If you are using SMS to direct archive log data set allocation, you should override this candidate volume count by specifying YES for the field SINGLE VOLUME on installation panel DSNTIPA. This will allow SMS to manage the allocation volume count appropriately when creating multi-volume disk archive log data sets.

Because SMS requires disk data sets to be cataloged, you must make sure the field CATALOG DATA on installation panel DSNTIPA contains YES. Even if it does not, message DSNJ072E is returned and the data set is forced to be cataloged by DB2.

DB2 uses the basic direct access method (BDAM) to read archive logs from disk. DFSMS does not support reading of compressed data sets using BDAM. You should not, therefore, use DFSMS hardware compression on your archive log data sets. Also, BDAM does not support extended sequential format data sets (which are used for striped or compressed data), so do not have DFSMS assign the extended sequential format to your archive log data sets.

Ensure that DFSMS does not alter the LRECL, BLKSIZE, or RECFM of the archive log data sets. Altering these attributes could result in read errors when DB2 attempts to access the log data.

Attention: DB2 does not issue an error or a warning if you write or alter archive data to an unreadable format. For example, if DB2 successfully writes archive log data to an extended format data set, DB2 issues an error message only when you attempt to read that data.

Controlling the log

You can control and monitor log activity through several commands and a utility as discussed in the following sections:

"Archiving the log"
"Dynamically changing the checkpoint frequency" on page 433
"Setting limits for archive log tape units" on page 433
"Displaying log information" on page 434

Archiving the log

General-use Programming Interface

A properly authorized operator can archive the current DB2 active log data sets, whenever required, by issuing the ARCHIVE LOG command. Using ARCHIVE LOG can help with diagnosis by allowing you to quickly offload the active log to the archive log where you can use DSN1LOGP to further analyze the problem.

To issue this command, you must have either SYSADM authority, or have been granted the ARCHIVE privilege.

```
-ARCHIVE LOG
```

When you issue the preceding command, DB2 truncates the current active log data sets, then runs an asynchronous offload, and updates the BSDS with a record of the offload. The RBA that is recorded in the BSDS is the beginning of the last complete log record written in the active log data set being truncated.

You could use the ARCHIVE LOG command as follows to capture a point of consistency for the MSTR01 and XUSR17 databases:

```
-STOP DATABASE (MSTR01,XUSR17)
-ARCHIVE LOG
-START DATABASE (MSTR01,XUSR17)
```

In this simple example, the STOP command stops activity for the databases before archiving the log.

Quiescing activity before offloading: Another method of ensuring that activity has stopped before the log is archived is the MODE(QUIESCE) option of ARCHIVE LOG. With this option, DB2 users are quiesced after a commit point, and the resulting point of consistency is captured in the current active log before it is offloaded. Unlike the QUIESCE utility, ARCHIVE LOG MODE(QUIESCE) does not force all changed buffers to be written to disk and does not record the log RBA in SYSIBM.SYSCOPY. It does record the log RBA in the bootstrap data set.

Consider using MODE(QUIESCE) when planning for offsite recovery. It creates a system-wide point of consistency, which can minimize the number of data inconsistencies when the archive log is used with the most current image copy during recovery.

In a data sharing group, ARCHIVE LOG MODE(QUIESCE) might result in a delay before activity on all members has stopped. If this delay is unacceptable to you, consider using ARCHIVE LOG SCOPE(GROUP) instead. This command causes truncation and offload of the logs for each active member of a data sharing group. Although the resulting archive log data sets do not reflect a point of consistency, all the archive logs are made at nearly the same time and have similar LRSN values in their last log records. When you use this set of archive logs to recover the data sharing group, you can use the ENDLRSN option in the CRESTART statement of the change log inventory utility (DSNJU003) to truncate all the logs in the group to the same point in time. See *DB2 Data Sharing: Planning and Administration* for more information.

The MODE(QUIESCE) option suspends all new update activity on DB2 up to the maximum period of time specified on the installation panel DSNTIPA, described in Part 2 of *DB2 Installation Guide*. If the time needed to quiesce is less than the time specified, then the command completes successfully; otherwise, the command fails when the time period expires. This time amount can be overridden when you issue the command, by using the TIME option:

```
-ARCHIVE LOG MODE(QUIESCE) TIME(60)
```

The preceding command allows for a quiesce period of up to 60 seconds before archive log processing occurs.

Important

Use of this option during prime time, or when time is critical, can cause a significant disruption in DB2 availability for all jobs and users that use DB2 resources.

By default, the command is processed asynchronously from the time you submit the command. (To process the command synchronously with other DB2 commands, use the WAIT(YES) option with QUIESCE; the z/OS console is then locked from DB2 command input for the entire QUIESCE period.)

During the quiesce period:

- Jobs and users on DB2 are allowed to go through commit processing, but are suspended if they try to update any DB2 resource after the commit.
- Jobs and users that only read data can be affected, because they can be waiting for locks held by jobs or users that were suspended.
- New tasks can start, but they are not allowed to update data.

As shown in the following example, the DISPLAY THREAD output issues message DSNV400I to indicate that a quiesce is in effect:

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV400I - ARCHIVE LOG QUIESCE CURRENTLY ACTIVE
DSNV402I - ACTIVE THREADS -
NAME     ST A   REQ ID           AUTHID   PLAN     ASID   TOKEN
BATCH    T  *    20 TEPJOB       SYSADM   DSNTEP3  0012      12
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I - DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
```

When all updates are quiesced, the quiesce history record in the BSDS is updated with the date and time that the active log data sets were truncated, and with the last-written RBA in the current active log data sets. DB2 truncates the current active log data sets, switches to the next available active log data sets, and issues message DSNJ311E, stating that offload started.

If updates cannot be quiesced before the quiesce period expires, DB2 issues message DSNJ317I, and archive log processing terminates. The current active log data sets are not truncated and not switched to the next available log data sets, and offload is *not* started.

Whether the quiesce was successful or not, all suspended users and jobs are then resumed, and DB2 issues message DSNJ312I, stating that the quiesce is ended and update activity is resumed.

If ARCHIVE LOG is issued when the current active log is the last available active log data set, the command is not processed, and DB2 issues this message:

```
DSNJ319I - csect-name CURRENT ACTIVE LOG DATA SET IS THE LAST
           AVAILABLE ACTIVE LOG DATA SET.  ARCHIVE LOG PROCESSING WILL
           BE TERMINATED.
```

If ARCHIVE LOG is issued when another ARCHIVE LOG command is already in progress, the new command is not processed, and DB2 issues this message:

```
DSNJ318I - ARCHIVE LOG COMMAND ALREADY IN PROGRESS.
```

Canceling log offloads: It is possible for the offload of an active log to be suspended when something goes wrong with the offload process, such as a problem with allocation or tape mounting. If the active logs cannot be offloaded, DB2's active log data sets fill up and DB2 stops logging.

To avoid this problem, use the following command to cancel (and retry) an offload:

```
-ARCHIVE LOG CANCEL OFFLOAD
```

When you enter the command, DB2 restarts the offload again, beginning with the oldest active log data set and proceeding through all active log data sets that need offloading. If the offload fails again, you must fix the problem that is causing the failure before the command can work.

End of General-use Programming Interface

Dynamically changing the checkpoint frequency

Use the LOGLOAD or CHKTIME option of the SET LOG command to dynamically change the checkpoint frequency without recycling DB2. The LOGLOAD value specifies the number of log records that DB2 writes between checkpoints. The CHKTIME value specifies the number of minutes between checkpoints. Either value affects the restart time for DB2.

For example, during prime shift, your DB2 shop might have a low logging rate, but require that DB2 restart quickly if it terminates abnormally. To meet this restart requirement, you can decrease the LOGLOAD value to force a higher checkpoint frequency. In addition, during off-shift hours the logging rate might increase as batch updates are processed, but the restart time for DB2 might not be as critical. In that case, you can increase the LOGLOAD value which lowers the checkpoint frequency.

You can also use the LOGLOAD or CHKTIME option to initiate an immediate system checkpoint:

```
-SET LOG LOGLOAD(0)
or
-SET LOG CHKTIME(0)
```

The CHKFREQ value that is altered by the SET LOG command persists only while DB2 is active. On restart, DB2 uses the CHKFREQ value in the DB2 subsystem parameter load module. See Chapter 2 of *DB2 Command Reference* for detailed information about this command.

Monitoring the system checkpoint

DB2 schedules a system checkpoint every time it switches active log data sets, regardless of the currently defined checkpoint frequency. If DB2 switches active logs and finds that there has not been a system checkpoint since the last log switch, it issues the following message to notify the operator that the system checkpoint processor might not be functioning.

```
DSNJ016E - csect-name WARNING - SYSTEM CHECKPOINT PROCESSOR MAY
           HAVE STALLED. LAST CHECKPOINT WAS TAKEN date-time
```

DB2 continues processing. This situation can result in a very long restart if logging continues without a system checkpoint. If DB2 continues logging beyond the defined checkpoint frequency, you should quiesce activity and terminate DB2 to minimize the restart time.

You can issue the DISPLAY LOG command or run the Print Log Map utility (DSNJU004) to display the most recent checkpoint. For additional information, see "Displaying log information" on page 434.

Setting limits for archive log tape units

Use the DB2 command SET ARCHIVE to set the upper limit for the number of and the deallocation time of tape units for the archive log. This command overrules the values specified during installation or in a previous invocation of the SET

ARCHIVE command. The changes initiated by SET ARCHIVE are temporary; at restart, DB2 uses the values that were set during installation. See Chapter 2 of *DB2 Command Reference* for detailed information about this command.

Displaying log information

Use the DISPLAY LOG command to display the current checkpoint frequency (either the number of log records or the minutes between checkpoints). See Chapter 2 of *DB2 Command Reference* for more details about the DISPLAY LOG and SET LOG commands.

You can obtain additional information about log data sets and checkpoints from the Print Log Map utility (DSNJU004). See Part 3 of *DB2 Utility Guide and Reference* for more information about utility DSNJU004.

Resetting the log RBA

Each DB2 subsystem writes its own recovery logs. The log records are sequenced with a Relative Byte Address (RBA), which you need to reset before the RBA reaches its maximum value.

To determine when to reset the log RBA, use one of the following two methods:

- Apply the PTF for APAR PK27611. After you apply this APAR, message DSNJ032I is issued at the active log switch when the RBA threshold is reached. If the RBA exceeds x'F80000000000', the message is issued with the keyword **WARNING** and processing continues. If the RBA exceeds x'FFFF00000000', the message is issued with the keyword **CRITICAL**, and DB2 is stopped. To resolve any outstanding units of work, DB2 will restart automatically in restart-light mode. Then, DB2 will stop again. In this situation, you need to restart DB2 in ACCESS(MAINT) mode, and you must reset the log RBA value.
- Calculate how much space is left in the log. You can use the print log map (DSNJU004) utility to obtain the highest written RBA value in the log. Subtract this RBA from x'FFFFFFFFFFFF' to determine how much space is left in the log. If APAR PK27611 is applied, you need to use the RBA value of x'FFFF00000000' for this calculation.

  You can use the output for the print log map utility to determine how many archive logs are created on an average day. This number multiplied by the RBA range of the archive log data sets (ENDRBA minus STARTRBA) provides the average number of bytes that are logged per day. Divide this value into the space remaining in the log to determine approximately how much time is left before the end of the log RBA range is reached. If there is less than one year remaining before the end of the log RBA range is reached, start planning to reset the log RBA value. If less than three months remain before the end of the log RBA range is reached, you need to take immediate action to reset the log RBA value.

Log RBA range

The log RBA is an ever-increasing 6 byte hex value that starts as 0 (zero) when the DB2 subsystem is first installed and increases to a maximum value of x'FFFFFFFFFFFF' (2 to the 48th).

The rate at which the log RBA value increases through this range depends on the logging rate of the DB2 subsystem. In cases where a heavy logging rate is sustained over a period of years, the log RBA value can begin to approach the end of the range.

Before the DB2 subsystem reaches the end of the log RBA range, you need to reset the log RBA value. The process that you complete to reset the log RBA value depends on whether the DB2 subsystem is the member of a data sharing group or in a non-data sharing environment.

Resetting the log RBA value in a data sharing environment

Before the member of a data sharing group reaches the end of the log RBA range, you need to reset the log RBA value for that member.

To reset the log RBA value in a data sharing environment:

1.
2. Restart this member in ACCESS(MAINT) mode.
3. Issue the -DISPLAY THREAD command. Ensure that there are no INDOUBT or POSTPONED ABORT units of recovery.
4. Issue the -DISPLAY DATABASE(*) SPACENAM(*) RESTRICT command. Ensure that all restricted states are removed.
5. Quiesce the member again by issuing the -STOP DB2 command.
6. **Optional:** Start a new member to take over the work of the member that is quiesced. If this is an acceptable solution, you can leave the original member stopped indefinitely.
7. To bring the original member back into the data sharing group, you must cold start the member with a STARTRBA of 0 (zero). To cold start the member:
   a. Before restarting the member, apply the PTF for APAR PK22872.
   b. Make a full image copy of all data. To make a full image copy, the member might need to remain quiesced for a period of time. The length of time depends on the size of the databases. After all of the data has been image copied, you no longer need the member logs for recovery.
   c. Cold start this member back to the RBA value of 0 (zero). This step removes all log data from the BSDS, and you can use the member again. This step requires utility DSNJU003 with the following options:

      ```
      CRESTART CREATE,STARTRBA=0,ENDRBA=0
      ```

Resetting the log RBA value in a non-data sharing environment

Before the DB2 subsystem in a non-data sharing environment reaches the end of the log RBA range, you need to reset the log RBA value for that subsystem.

To reset the log RBA value in a non-data sharing environment:

1. Issue the STOP DB2 command to quiesce the subsystem that is approaching the end of the log RBA range.
2. Restart DB2 in ACCESS(MAINT) mode.
3. Issue the -DISPLAY THREAD command. Ensure that there are no INDOUBT or POSTPONED ABORT units of recovery.
4. Issue the -DISPLAY UTILITY command. Ensure there are no active or stopped utilities.
5. Issue the -DISPLAY DATABASE(*) SPACENAM(*) RESTRICT command. Ensure that all restricted states are removed.
6. Quiesce the DB2 subsystem again by issuing the -STOP DB2 command.
7. Use IDCAMS to delete and redefine the table spaces SYSUTILX, SYSCOPY, and SYSLGRNX and their corresponding indexes. Then, re-initialize these

page sets. For information about how to re-initialize these page sets, see member DSNTIJID in library SDSNSAMP.

8. Determine whether you can use the COPY utility, and complete the appropriate task as follows:
   - If the PTF for APAR PK28576 is applied, or can be applied, and you can use the COPY utility to make copies of all data, apply the PTF for APAR PK28576. This APAR enables the COPY utility to reset the RBA values in pages as they are copied. See the documentation for the PTF for information about enabling the COPY utility function that is provided by this APAR.
   - If you cannot use the COPY utility, do the following:
     a. Use DSN1COPY to copy every table space and partition to another data set. If indexes are not included in this process, you must rebuild the indexes after restarting DB2.
     b. Use DSN1COPY with the RESET parameter to copy each table space or partition to its original data set. This process resets the RBA values in all of the page sets. If indexes are not included in this process, you must rebuild the indexes after restarting DB2.
9. Cold start this subsystem back to the RBA value of 0 (zero) by using ACCESS(MAINT). This step removes all log data from the BSDS. This step requires utility DSNJU003 with the following options:

   ```
   CRESTART CREATE,STARTRBA=0,ENDRBA=0
   ```

10. If applicable, rebuild all indexes. Start by rebuilding the catalog and directory indexes, and then rebuild the user data.
11. Take new, full image copies of all data. If you applied the PTF for APAR PK28576, run the COPY utility with option SHRLEVEL REFERENCE to automatically reset the RBA values in all of the page sets. This step can be performed in parallel with the previous step after the catalog and directory indexes are rebuilt.
12. Stop DB2. If applicable, disable the reset RBA function in the COPY utility, and restart DB2 for normal access.

Managing the bootstrap data set (BSDS)

The BSDS is a VSAM key-sequenced data set that contains information about the log data sets and the records those data sets include. It also contains information about buffer pool attributes. The BSDS is defined with access method services when DB2 is installed and is allocated by a DD statement in the DB2 startup procedure. It is deallocated when DB2 terminates.

Normally, DB2 keeps duplicate copies of the BSDS. If an I/O error occurs, DB2 deallocates the failing copy and continues with a single BSDS. However, you can restore the dual mode as follows:

1. Use access method services to rename or delete the failing BSDS.
2. Define a new BSDS with the same name as the deleted BSDS.
3. Issue the DB2 command RECOVER BSDS to make a copy of the good BSDS in the newly allocated data set.

The active logs are first registered in the BSDS by job DSNTIJID, when DB2 is installed. They cannot be replaced, nor new ones added, without terminating and restarting DB2.

Archive log data sets are dynamically allocated. When one is allocated, the data set name is registered in the BSDS in separate entries for each volume on which the

archive log resides. The list of archive log data sets expands as archives are added, and wraps around when a user-determined number of entries has been reached. The maximum number of archive log data sets depends on whether the BSDS conversion utility (DSNJCNVB) has been run. The maximum number of data sets is 10000 if the BSDS has been converted (20000 for dual logging); otherwise the limits are 1000 for single archive logging and 2000 for dual logging. For more information about the BSDS conversion utility, see Part 3 of *DB2 Utility Guide and Reference*.

The inventory of archive log data sets can be managed by use of the change log inventory utility (DSNJU003). For further information, see "Changing the BSDS log inventory" on page 438.

A wide variety of tape management systems exist, along with the opportunity for external manual overrides of retention periods. Because of that, DB2 does not have an automated method to delete the archive log data sets from the BSDS inventory of archive log data sets. Thus, the information about an archive log data set can be in the BSDS long after the archive log data set has been scratched by a tape management system following the expiration of the data set's retention period.

Conversely, the maximum number of archive log data sets could have been exceeded, and the data from the BSDS dropped long before the data set has reached its expiration date. For additional information, refer to "Deleting archive logs automatically" on page 438.

If you specified at installation that archive log data sets are cataloged when allocated, the BSDS points to the integrated catalog facility catalog for the information needed for later allocations. Otherwise, the BSDS entries for each volume register the volume serial number and unit information that is needed for later allocation.

BSDS copies with archive log data sets

Each time a new archive log data set is created, a copy of the BSDS is also created. If the archive log is on tape, the BSDS is the first file on the first output volume. If the archive log is on disk, the BSDS copy is a separate file which could reside on a separate volume.

For better offload performance and space utilization, it is recommended that you use the default archive block size of 28672. If required, this value can be changed in the BLOCK SIZE field on installation panel DSNTIPA. The PRIMARY QUANTITY and SECONDARY QUANTITY fields should also be adjusted to reflect any changes in block size.

The data set names of the BSDS copy and the archive log are the same, except that the first character of the last data set name qualifier in the BSDS name is B instead of A, as in the following example:

Archive log name
DSNCAT.ARCHLOG1.**A**0000001

BSDS copy name
DSNCAT.ARCHLOG1.**B**0000001

If a read error while copying the BSDS, the copy is not created. Message DSNJ125I is issued, and the offload to the new archive log data set continues without the BSDS copy.

The utility DSNJU004, print log map, lists the information that is stored in the BSDS. For instructions on using it, see Part 3 of *DB2 Utility Guide and Reference*.

Changing the BSDS log inventory

You do not have to take special steps to keep the BSDS updated with records of logging events: DB2 does that automatically. However, you might want to change the BSDS if you:

- Add more active log data sets.
- Copy active log data sets to newly allocated data sets, as when providing larger active log allocations.
- Move log data sets to other devices.
- Recover a damaged BSDS.
- Discard outdated archive log data sets.
- Create or cancel control records for conditional restart.
- Add to or change the DDF communication record.

You can change the BSDS by running the DB2 batch change log inventory (DSNJU003) utility. This utility should not be run when DB2 is active. If it is run when DB2 is active, inconsistent results can be obtained. For instructions on how to use the change log inventory utility, see Part 3 of *DB2 Utility Guide and Reference*.

You can copy an active log data set using the access method services IDCAMS REPRO statement. The copy can only be performed when DB2 is down, because DB2 allocates the active log data sets as exclusive (DISP=OLD) at DB2 startup. For more information about the REPRO statement, see *DFSMS/MVS: Access Method Services for the Integrated Catalog* and *z/OS DFSMS Access Method Services for Catalogs*.

Discarding archive log records

You must keep enough log records to recover units of work and databases.

To recover units of recovery, you need log records at least until all current actions are completed. If DB2 abends, restart requires all log records since the previous checkpoint or the beginning of the oldest UR that was active at the abend, whichever is first on the log.

To tell whether all units of recovery are complete, read the status counts in the DB2 restart messages (shown in “Starting DB2” on page 324). If all counts are zero, no unit of recovery actions are pending. If there are indoubt units of recovery remaining, identify and recover them by the methods described in Chapter 17, “Monitoring and controlling DB2 and its connections,” on page 363.

To recover databases, you need log records and image copies of table spaces. How long you keep log records depends, then, on how often you make those image copies. Chapter 21, “Backing up and recovering databases,” on page 471 gives suggestions about recovery cycles; the following sections assume that you know what records you want to keep and describe only how to delete the records you do not want.

Deleting archive logs automatically

You can use a disk or tape management system to delete archive log data sets or tapes automatically. The length of the retention period (in days), which is passed to

the management system in the JCL parameter RETPD, is determined by the RETENTION PERIOD field on the DSNTIPA installation panel, discussed further in Part 2 of *DB2 Installation Guide*.

The default for the retention period keeps archive logs forever. If you use any other retention period, it must be long enough to contain as many recovery cycles as you plan for. For example, if your operating procedures call for a full image copy every sixty days of the least-frequently-copied table space, and you want to keep two complete image copy cycles on hand at all times, then you need an archive log retention period of at least 120 days. For more than two cycles, you need a correspondingly longer retention period.

If archive log data sets or tapes are deleted automatically, the operation does not update the archive log data set inventory in the BSDS. If you wish, you can update the BSDS with the change log inventory utility, as described in "Changing the BSDS log inventory" on page 438. The update is not really necessary; it wastes space in the BSDS to record old archive logs, but does no other harm as the archive log data set inventory *wraps* and automatically deletes the oldest entries. See "Managing the bootstrap data set (BSDS)" on page 436 for more details.

Locating archive log data sets

You must keep all the logs since the most recent checkpoint of DB2, so that DB2 can restart. You also must keep all the logs for two or more complete image copy cycles of your least-frequently-copied table space. What, then, can you discard?

You need an answer in terms of the log RBA ranges of the archive data sets. The earliest log record you want is identified by a log RBA. You can discard any archive log data sets that contain only records with log RBAs less than that.

The procedure that follows locates those data sets:

Step 1: Resolve indoubt units of recovery: If DB2 is running with TSO, continue with "Find the startup log RBA" on page 440. If DB2 is running with IMS, CICS, or distributed data, the following procedure applies:

1. The period between one startup and the next must be free of any indoubt units of recovery. Ensure that no DB2 activity is going on until you finish this procedure. (You might plan this procedure for a non-prime shift, for minimum impact on users.) To find out whether indoubt units exist, issue the DB2 command DISPLAY THREAD TYPE(INDOUBT). If there are none, skip to "Find the startup log RBA" on page 440.
2. If there are one or more indoubt units of recovery, take one of the following actions:
   - Start IMS or CICS, causing that subsystem to resolve the indoubt units of recovery. If the thread is a distributed indoubt unit of recovery, restart the distributed data facility (DDF) to resolve the unit of work. If DDF does not start or cannot resolve the unit of work, use the command RECOVER INDOUBT to resolve the unit of work.
   - Issue the DB2 command RECOVER INDOUBT.

     These topics, including making the proper commit or abort decision, are covered in greater detail in "Resolving indoubt units of recovery" on page 459.
3. Re-issue the command DISPLAY THREAD TYPE(INDOUBT) to ensure that the indoubt units have been recovered. When none remain, continue with "Find the startup log RBA" on page 440.

Step 2: Find the startup log RBA: Keep at least all log records with log RBAs greater than the one given in this message, issued at restart:

```
DSNR003I RESTART...PRIOR CHECKPOINT RBA=xxxxxxxxxxxx
```

If you suspended DB2 activity while performing step 1, you can restart it now.

Step 3: Find the minimum log RBA needed: Suppose that you have determined to keep some number of complete image copy cycles of your least-frequently-copied table space. You now need to find the log RBA of the earliest full image copy you want to keep.

1. If you have any table spaces so recently created that no full image copies of them have ever been taken, take full image copies of them. If you do not take image copies of them, and you discard the archive logs that log their creation, DB2 can never recover them.

   General-use Programming Interface

   The following SQL statement lists table spaces that have no full image copy:

   ```
   SELECT X.DBNAME, X.NAME, X.CREATOR, X.NTABLES, X.PARTITIONS
     FROM SYSIBM.SYSTABLESPACE X
       WHERE NOT EXISTS (SELECT * FROM SYSIBM.SYSCOPY Y
                         WHERE X.NAME = Y.TSNAME
                         AND X.DBNAME = Y.DBNAME
                         AND Y.ICTYPE = 'F')
     ORDER BY 1, 3, 2;
   ```

   End of General-use Programming Interface

2. Issue the following SQL statement to find START_RBA values:

   General-use Programming Interface

   ```
   SELECT DBNAME, TSNAME, DSNUM, ICTYPE, ICDATE, HEX(START_RBA)
     FROM SYSIBM.SYSCOPY
     ORDER BY DBNAME, TSNAME, DSNUM, ICDATE;
   ```

   End of General-use Programming Interface

   The statement lists all databases and table spaces within them, in ascending order by date.

   Find the START_RBA for the earliest full image copy (ICTYPE=F) that you intend to keep. If your least-frequently-copied table space is partitioned, and you take full image copies by partition, use the earliest date for all the partitions.

   If you are going to discard records from SYSIBM.SYSCOPY and SYSIBM.SYSLGRNX, note the date of the earliest image copy you want to keep.

Step 4: Copy catalog and directory tables: Take full image copies of the DB2 table spaces listed in Table 96 to ensure that copies of these table spaces are included in the range of log records you will keep.

Table 96. Catalog and directory tables to copy

Database name	Table space names	
DSNDB01	DBD01 SCT02 SPT01	SYSUTILX SYSLGRNX

Table 96. Catalog and directory tables to copy (continued)

Database name	Table space names	
DSNDB06	SYSCOPY	SYSPLAN
	SYSDBASE	SYSSTATS
	SYSDBAUT	SYSSTR
	SYSGPAUT	SYSUSER
	SYSGROUP	SYSVIEWS
	SYSPKAGE	

Step 5: Locate and discard archive log volumes: Now that you know the minimum LOGRBA, from step 3, suppose that you want to find archive log volumes that contain only log records earlier than that. Proceed as follows:

1. Execute the print log map utility to print the contents of the BSDS. For an example of the output, see the description of print log map (DSNJU004) in Part 3 of *DB2 Utility Guide and Reference*.
2. Find the sections of the output titled "ARCHIVE LOG COPY n DATA SETS". (If you use dual logging, there are two sections.) The columns labelled STARTRBA and ENDRBA show the range of log RBAs contained in each volume. Find the volumes (two, for dual logging) whose ranges include the minimum log RBA you found in step 3; these are the earliest volumes you need to keep.

   If no volumes have an appropriate range, one of these cases applies:

   - The minimum LOGRBA has not yet been archived, and you can discard all archive log volumes.
   - The list of archive log volumes in the BSDS wrapped around when the number of volumes exceeded the number allowed by the RECORDING MAX field of installation panel DSNTIPA. If the BSDS does not register an archive log volume, it can never be used for recovery. Therefore, you should consider adding information about existing volumes to the BSDS. For instructions, see Part 3 of *DB2 Utility Guide and Reference*.

     You should also consider increasing the value of MAXARCH. For information, see information about installation panel DSNTIPA in Part 2 of *DB2 Installation Guide*.
3. Delete any archive log data set or volume (both copies, for dual logging) whose ENDRBA value is less than the STARTRBA value of the earliest volume you want to keep.

   Because BSDS entries wrap around, the first few entries in the BSDS archive log section might be more recent than the entries at the bottom. Look at the combination of date and time to compare age. Do not assume you can discard all entries above the entry for the archive log containing the minimum LOGRBA.

   Delete the data sets. If the archives are on tape, scratch the tapes; if they are on disks, run a z/OS utility to delete each data set. Then, if you want the BSDS to list only existing archive volumes, use the change log inventory utility to delete entries for the discarded volumes; for an example, see Part 3 of *DB2 Utility Guide and Reference*.

Chapter 19. Restarting DB2 after termination

This chapter tells what to expect when DB2 terminates normally or abnormally, and how to start it again. The concepts are important background for Chapter 20, "Maintaining consistency across multiple systems," on page 455 and Chapter 21, "Backing up and recovering databases," on page 471. This chapter includes the following topics:

"Termination"
"Normal restart and recovery" on page 444
"Deferring restart processing" on page 450
"Restarting with conditions" on page 451

Chapter 20, "Maintaining consistency across multiple systems," on page 455 describes additional considerations for a DB2 subsystem that must be kept consistent with some other system. The term *object*, used throughout this chapter, refers to any database, table space, or index space.

Restarting in a data sharing environment: In a data sharing environment, restart processing is expanded to handle the coordination of data recovery across more than one DB2 subsystem. When certain critical resources are lost, restart includes additional processing to recovery and rebuild those resources. This process is called *group restart*, which is described in Chapter 5 of *DB2 Data Sharing: Planning and Administration*.

Termination

DB2 terminates normally in response to the command STOP DB2. If DB2 stops for any other reason, the termination is considered abnormal.

Normal termination

In a normal termination, DB2 stops all activity in an orderly way. You can use either STOP DB2 MODE (QUIESCE) or STOP DB2 MODE (FORCE). The effects are given in Table 97.

Table 97. Termination using QUIESCE and FORCE

Thread type	QUIESCE	FORCE
Active threads	Run to completion	Roll back
New threads	Permitted	Not permitted
New connections	Not permitted	Not permitted

You can use either command to prevent new applications from connecting to DB2.

When you issue the command STOP DB2 MODE(QUIESCE), current threads can run to completion, and new threads can be allocated to an application that is running.

With IMS and CICS, STOP DB2 MODE(QUIESCE) allows a current thread to run only to the end of the unit of recovery, unless either of the following conditions are true:

- There are open, held cursors.

© Copyright IBM Corp. 1982, 2008

- Special registers are not in their original state.

Before DB2 can come down, all held cursors must be closed and all special registers must be in their original state, or the transaction must complete.

With CICS, QUIESCE mode brings down the CICS attachment facility, so an active task will not necessarily run to completion.

For example, assume that a CICS transaction opens no cursors declared WITH HOLD and modifies no special registers as follows:

```
EXEC SQL
:          ← -STOP DB2 MODE(QUIESCE) issued here
:
SYNCPOINT
:
EXEC SQL    ←  This receives an AETA abend
```

The thread is allowed to run only through the first SYNCPOINT.

When you issue the command STOP DB2 MODE(FORCE), no new threads are allocated, and work on existing threads is rolled back.

During shutdown, use the command DISPLAY THREAD to check its progress. If shutdown is taking too long, you can issue STOP DB2 MODE (FORCE), but rolling back work can take as much or more time as the completion of QUIESCE.

When stopping in either mode, the following steps occur:
1. Connections end.
2. DB2 ceases to accept commands.
3. DB2 disconnects from the IRLM.
4. The shutdown checkpoint is taken and the BSDS is updated.

A data object could be left in an inconsistent state, even after a shutdown with mode QUIESCE, if it was made unavailable by the command STOP DATABASE, or if DB2 recognized a problem with the object. MODE (QUIESCE) does not wait for asynchronous tasks that are not associated with any thread to complete, before it stops DB2. This can result in data commands such as STOP DATABASE and START DATABASE having outstanding units of recovery when DB2 stops. These will become inflight units of recovery when DB2 is restarted, then be returned to their original states.

Abends

An abend can leave data in an inconsistent state for any of the following reasons:
- Units of recovery might be interrupted before reaching a point of consistency.
- Committed data might not be written to external media.
- Uncommitted data might be written to external media.

Normal restart and recovery

DB2 uses its recovery log and the bootstrap data set (BSDS) to determine what to recover when restarting. The BSDS identifies the active and archive log data sets, the location of the most recent DB2 checkpoint on the log, and the high-level qualifier of the integrated catalog facility catalog name.

After DB2 is initialized, the restart process goes through four phases, which are described in the following sections:

"Phase 1: Log initialization"
"Phase 2: Current status rebuild" on page 446
"Phase 3: Forward log recovery" on page 447
"Phase 4: Backward log recovery" on page 448

In the descriptions that follow, the terms *inflight*, *indoubt*, *in-commit*, and *in-abort* refer to statuses of a unit of work that is coordinated between DB2 and another system, such as CICS, IMS, or a remote DBMS. For definitions of those terms, see "Maintaining consistency after termination or failure" on page 457.

At the end of the fourth phase recovery, a checkpoint is taken, and committed changes are reflected in the data.

Application programs that do not commit often enough cause long running units of recovery (URs). These long running URs might be inflight after a DB2 failure. Inflight URs can extend DB2 restart time. You can restart DB2 more quickly by postponing the backout of long running URs. Use installation options LIMIT BACKOUT and BACKOUT DURATION to establish what work to delay during restart.

If your DB2 subsystem has the UR checkpoint count option enabled, DB2 generates console message DSNR035I and trace records for IFCID 0313 to inform you about long running URs. The UR checkpoint count option is enabled at installation time, through field UR CHECK FREQ on panel DSNTIPL. See Part 2 of *DB2 Installation Guide* for more information about enabling this option.

If your DB2 subsystem has the UR log threshold option enabled, DB2 generates
console message DSNB260I when an inflight UR writes more than the
installation-defined number of log records. DB2 also generates trace records for
IFCID 0313 to inform you about these long running URs. You can enable the UR
log threshold option at installation time, through field UR LOG WRITE CHECK on
panel DSNTIPL. See Part 2 of *DB2 Installation Guide* for more information about
enabling this option.

You can restart a large object (LOB) table space like other table spaces. LOB table spaces defined with LOG NO do not log LOB data, but they log enough control information (and follow a force-at-commit policy) so that they can restart without loss of data integrity.

Phase 1: Log initialization

During phase 1, DB2 attempts to locate the last log RBA written before termination. Logging continues at the next log RBA after that. In phase 1, DB2:

1. Compares the high-level qualifier of the integrated catalog facility catalog name, in the BSDS, with the corresponding qualifier of the name in the current subsystem parameter module (DSNZP*xxx*).
   - If they are equal, processing continues with step 2 on page 446.
   - If they are not equal, DB2 terminates with this message:

     ```
     DSNJ130I ICF CATALOG NAME IN BSDS
              DOES NOT AGREE WITH DSNZPARM.
              BSDS CATALOG NAME=aaaaa,
              DSNZPARM CATALOG NAME=bbbbb
     ```

     Without the check, the next DB2 session could conceivably update an entirely different catalog and set of table spaces. If the check fails, you presumably have the wrong parameter module. Start DB2 with the command START DB2 PARM(*module-name*), and name the correct module.

2. Checks the consistency of the timestamps in the BSDS.
   - If both copies of the BSDS are current, DB2 tests whether the two timestamps are equal.
     - If they are equal, processing continues with step 3.
     - If they are not equal, DB2 issues message DSNJ120I and terminates. That can happen when the two copies of the BSDS are maintained on separate disk volumes (as recommended) and one of the volumes is restored while DB2 is stopped. DB2 detects the situation at restart.

       To recover, copy the BSDS with the latest timestamp to the BSDS on the restored volume. Also recover any active log data sets on the restored volume, by copying the dual copy of the active log data sets onto the restored volume. For more detailed instructions, see "BSDS failure recovery" on page 538.
   - If one copy of the BSDS was deallocated, and logging continued with a single BSDS, a problem could arise. If both copies of the BSDS are maintained on a single volume, and the volume was restored, or if both BSDS copies were restored separately, DB2 might not detect the restoration. In that case, log records not noted in the BSDS would be unknown to the system.
3. Finds in the BSDS the log RBA of the last log record written before termination.

   The highest RBA field (as shown in the output of the print log map utility) is updated only when the following events occur:
   - When DB2 is stopped normally (STOP DB2).
   - When active log writing is switched from one data set to another.
   - When DB2 has reached the end of the log output buffer. The size of this buffer is determined by the OUTPUT BUFFER field of installation panel DSNTIPL described in Part 2 of *DB2 Installation Guide*.
4. Scans the log forward, beginning at the log RBA of the most recent log record, up to the last control interval (CI) written before termination.
5. Prepares to continue writing log records at the next CI on the log.
6. Issues message DSNJ099I, which identifies the log RBA at which logging continues for the current DB2 session. That message signals the end of the log initialization phase of restart.

Phase 2: Current status rebuild

During phase 2, DB2 determines the statuses of objects at the time of termination. By the end of the phase, DB2 has determined whether any units of recovery were interrupted by the termination. In phase 2, DB2:

1. Checks the BSDS to find the log RBA of the last complete checkpoint before termination.
2. Processes the RESTART or DEFER option of the parameter module of the START DB2 command if any exist. The default is always RESTART ALL.
3. Reads every log record from that checkpoint up to the end of the log (which was located during phase 1), and identifies:
   - All exception conditions existing for each database and all image copy information related to the DSNDB01.SYSUTILX, DSNDB01.DBD01, and DSNDB06.SYSCOPY table spaces.
   - All objects open at the time of termination, and how far back in the log to go to reconstruct data pages that were not written to disk.

The number of log records written between one checkpoint and the next is set when DB2 is installed; see the field CHECKPOINT FREQ of installation panel DSNTIPL, described in Part 2 of *DB2 Installation Guide*.

You can temporarily modify the checkpoint frequency by using the command SET LOG. The value you specify persists while DB2 is active; on restart, DB2 uses the value that is specified in the CHECKPOINT FREQ field of installation panel DSNTIPL. See Chapter 2 of *DB2 Command Reference* for detailed information about this command.

4. Issues message DSNR004I, which summarizes the activity required at restart for outstanding units of recovery.
5. Issues message DSNR007I if any outstanding units of recovery are discovered. The message includes, for each outstanding unit of recovery, its connection type, connection ID, correlation ID, authorization ID, plan name, status, log RBA of the beginning of the unit of recovery (URID), and the date and time of its creation.

During phase 2, no database changes are made, nor are any units of recovery completed. DB2 determines what processing is required by phase 3 forward log recovery before access to databases is allowed.

Phase 3: Forward log recovery

During phase 3, DB2 completes the processing for all committed changes and database write operations. This includes:

- Making all database changes for each indoubt unit of recovery and locking the data to prevent access to it after restart
- Making database changes for inflight and in-abort units of recovery, and in the case of group restarts in which locks have been lost, acquiring locks to prevent access to data in use by those units of recovery

DB2 can use the fast log apply process to enhance performance of the forward recovery phase. See the Log Apply Storage field on panel DSNTIPL in *DB2 Installation Guide*, for details on defining storage for the sort used in fast log apply processing. DB2 executes these steps:

1. Detects whether a page set being recovered is at the same level ID as it was when the page set was last closed. If it is not, DB2 issues message DSNB232I and places the pages for that object on the logical page list (LPL). DB2 does not restart that object. In this case, you must recover from the down level page set using one of the methods described in "Recovery from down-level page sets" on page 543.
2. Scans the log forward, beginning at the lowest (earliest) log RBA that is either required for completion of database writes or is associated with the "Begin Unit of Recovery" of units of recovery that require locks.

   That log RBA is determined during phase 2. REDO log records for all units of recovery are processed in this phase.
3. Uses the log RBA of the earliest potential redo log record for each object (determined during phase 2). All earlier changes to the object have been written to disk; therefore, DB2 ignores their log records.
4. Reads the data or index page for each remaining redo log record. The page header registers the log RBA of the record of the last change to the page.
   - If the log RBA of the page header is greater than or equal to that of the current log record, the logged change has already been made and written to disk, and the log record is ignored.

- If the log RBA in the page header is less than that of the current log record, the change has not been made; DB2 makes the change to the page in the buffer pool.

5. Writes pages to disk as the need for buffers demands it.
6. Marks the completion of each unit of recovery processed. If restart processing terminates later, those units of recovery do not reappear in status lists.
7. Stops scanning at the current end of the log.
8. Writes to disk all modified buffers not yet written.
9. Issues message DSNR005I, which summarizes the number of remaining in-commit or indoubt units of recovery. There should not be any in-commit units of recovery, because all processing for these should have completed. The number of indoubt units of recovery should be equal to the number specified in the previous DSNR004I restart message.
10. Issues message DSNR007I (described in "Phase 2: Current status rebuild" on page 446), which identifies any outstanding unit of recovery that still must be processed.

If DB2 encounters a problem while applying log records to an object during phase 3, the affected pages are placed in the logical page list. Message DSNI001I is issued once per page set or partition, and message DSNB250E is issued once per page. Restart processing continues.

DB2 issues status message DSNR031I periodically during this phase.

Phase 4: Backward log recovery

During phase 4, DB2 changes performed for inflight or in-abort units of recovery are reversed. In phase 4, DB2:

1. Scans the log backward, starting at the current end. The scan continues until the earliest "Begin Unit of Recovery" record for any outstanding inflight or in-abort unit of recovery.

   If you specified limited backout processing, the scan might stop prematurely (however, DB2 guarantees that all catalog and directory changes are completely backed out). In this case, the scan completes after DB2 receives the RECOVER POSTPONED command. You can have DB2 automatically issue this command after restart by specifying the AUTO option for limited backout processing, or you can issue this command manually.
2. Reads the data or index page for each remaining undo log record. The page header registers the log RBA of the record of the last change to the page.
   - If the log RBA of the page header is greater than or equal to that of the current log record, the logged change has already been made and written to disk, therefore DB2 reverses it.
   - If the log RBA in the page header is less than that of the current log record, the change has not been made; DB2 ignores it.
3. Writes redo compensation information in the log for each undo log record, as it does when backing out a unit of recovery. The redo records describe the reversal of the change and facilitate media recovery. They are written under all circumstances, even when:
   - The disk version of the data did not need to be reversed.
   - The page set has pages on the LPL.
   - An I/O error occurred on the disk version of the data.
   - The disk version of the data could not be allocated or opened.
4. Writes pages to disk as the need for buffers demands it.

5. Finally, writes to disk all modified buffers that have not yet been written.
6. Issues message DSNR006I, which summarizes the number of remaining inflight, in-abort, and postponed-abort units of recovery. The number of inflight and in-abort units of recovery should be zero; the number of postponed-abort units of recovery might not be zero.
7. Marks the completion of each completed unit of recovery in the log so that, if restart processing terminates, the unit of recovery is not processed again at the next restart.
8. If necessary, reacquires write claims for the objects on behalf of the indoubt and postponed-abort units of recovery.
9. Takes a checkpoint, after all database writes have been completed.

If DB2 encounters a problem while applying a log record to an object during phase 4, the affected pages are placed in the logical page list. Message DSNI001I is issued once per page set or partition, and message DSNB250E is issued once per page. Restart processing continues.

DB2 issues status message DSNR031I periodically during this phase.

Restarting automatically

If you are running DB2 in a Sysplex, you can have the automatic restart function of z/OS automatically restart DB2 or IRLM after a failure.

When DB2 or IRLM stops abnormally, z/OS determines whether z/OS failed too, and where DB2 or IRLM should be restarted. It then restarts DB2 or IRLM.

You must have DB2 installed with a command prefix scope of S to take advantage of automatic restart. See Part 2 of *DB2 Installation Guide* for instruction on specifying command scope.

Using an automatic restart policy: You control how automatic restart works by using automatic restart policies. When the automatic restart function is active, the default action is to restart the subsystems when they fail. If this default action is not what you want, then you must create a policy defining the action you want taken.

To create a policy, you need the element names of the DB2 and IRLM subsystems:

- For a non-data-sharing DB2, the element name is 'DB2$' concatenated by the subsystem name (DB2$DB2A, for example). To specify that a DB2 subsystem is not to be restarted after a failure, include RESTART_ATTEMPTS(0) in the policy for that DB2 element.
- For local mode IRLM, the element name is a concatenation of the IRLM subsystem name and the IRLM ID. For global mode IRLM, the element name is a concatenation of the IRLM data sharing group name, IRLM subsystem name, and the IRLM ID.

For instructions on defining automatic restart policies, see *z/OS MVS Setting Up a Sysplex*.

Deferring restart processing

Usually, restarting DB2 activates restart processing for objects that were available when DB2 terminated (in other words, not stopped with the command STOP DATABASE). Restart processing applies or backs out log records for objects that have unresolved work.

Restart processing is controlled by what you choose on installation panel DSNTIPS, and the default is RESTART ALL.

If some specific object is causing problems, you should consider deferring its restart processing by starting DB2 without allowing that object to go through restart processing. When you defer restart of an object, DB2 puts pages necessary for the object's restart in the logical page list (LPL). Only those pages are inaccessible; the rest of the object can still be accessed after restart.

There are restrictions to DB2's activation of restart processing for available objects. When you say DEFER ALL at a site that is designated as RECOVERYSITE in DSNZP*xxx*, all pages are placed in the LPL (as a page range, not as a large list of individual pages). All pages for the index are also placed in the LPL. The following conditions apply:

• If DB2 cannot open and read the DBD01 table space it will not put DB2 into ACCESS(MAINT), and DSNX204I is not be issued. Instead either DSNT500I or DSNT501I 'resource unavailable' are issued.
• For a deferring restart scenario that needs to recover all DB2 objects after DB2 is up, it is recommend that you set the ZPARM DEFER ALL and start DB2 with the ACCESS(MAINT) option.
• If DEFER ALL is specified, DSNX204I is not issued.
• With DEFER ALL, DB2 will not open any data sets, including SYSLGRNX and DSNRTSTS, during any phase of restart, and will not attempt to apply any log records.

DB2 can also defer restart processing for particular objects. DB2 puts pages in the LPL for any object (or specific pages of an object) with certain problems, such as an open or I/O error during restart. Again, only pages that are affected by the error are placed on the LPL.

You can defer an object's restart processing with any of the following actions:

VARY the device (or volume) on which the objects reside OFFLINE. If the data sets containing an object are not available, and the object requires recovery during restart, DB2 flags it as stopped and requiring deferred restart. DB2 then restarts without it.

Delay the backout of a long running UR. On installation panel DSNTIPL, you can use the following options:

- LIMIT BACKOUT defined as YES or AUTO indicates that some backout processing will be postponed when restarting DB2. Users must issue the RECOVER POSTPONED command to complete the backout processing when the YES option is selected. DB2 does the backout work automatically after DB2 is running and receiving new work when the AUTO option is selected.
- BACKOUT DURATION indicates the number of log records, specified as a multiplier, to be read during restart's backward log scan.

The amount of backout processing to be postponed is determined by:

- The frequency of checkpoints
- The BACKOUT DURATION installation option
- The characteristics of the inflight and in-abort activity when the system went down

Selecting a limited backout affects log processing during restart. The backward processing of the log proceeds until the oldest inflight or in-abort UR with activity against the catalog or directory is backed out, and the requested number of log records have been processed.

Name the object with DEFER when installing DB2. On installation panel DSNTIPS, you can use the following options:

- DEFER ALL defers restart log apply processing for all objects, including DB2 catalog and directory objects.
- DEFER *list_of_objects* defers restart processing only for objects in the list.

Alternatively, you can specify RESTART *list_of_objects*, which limits restart processing to the list of objects in the list.

DEFER does not affect processing of the log during restart. Therefore, even if you specify DEFER ALL, DB2 still processes the full range of the log for both the forward and backward log recovery phases of restart. However, logged operations are not applied to the data set.

Restarting with conditions

If you want to skip some portion of the log processing during DB2 restart, you can use a conditional restart. However, if a conditional restart skips any database change log records, data in the associated objects becomes inconsistent and any attempt to process them for normal operations might cause unpredictable results. The only operations that can safely be performed on the objects are recovery to a prior point of consistency, total replacement, or dropping.

In unusual cases, you might choose to make inconsistent objects available for use without recovering them. For example, the only inconsistent object might be a table space that is dropped as soon as DB2 is restarted, or the DB2 subsystem might be used only for testing application programs still under development. In cases like those, where data consistency is not critical, normal recovery operations can be partially or fully bypassed by using conditional restart control records in the BSDS. The procedure is:

1. While DB2 is stopped, run the change log inventory utility using the CRESTART control statement to create a new conditional restart control record.
2. Restart DB2. The type of recovery operations that take place is governed by the current conditional restart control record.

When considering a conditional restart, it is often useful to run the DSN1LOGP utility and review a summary report of the information contained in the log.

For an example of the messages that are written to the DB2 console during restart processing, see “Messages at start” on page 324.

In a data sharing environment, you can use the new LIGHT(YES) parameter on the START DB2 command to quickly recover retained locks on a DB2 member. For more information, see *DB2 Data Sharing: Planning and Administration*.

Restart considerations for identity columns: Cold starts and conditional restarts that skip forward recovery can cause additional data inconsistency within identity columns and sequence objects. After such restarts, DB2 might assign duplicate identity column values and create gaps in identity column sequences. For information about how to correct this data inconsistency, see "Recovering catalog and directory tables" on page 498.

This section gives an overview of the available options for conditional restart. For more detail, see information about the change log inventory utility (DSNJU003) in Part 3 of *DB2 Utility Guide and Reference*. For information about data sharing considerations, see Chapter 5 of *DB2 Data Sharing: Planning and Administration*.

Resolving postponed units of recovery

You can postpone some of the backout work that is associated with long running units of work during system restart by using the LIMIT BACKOUT installation option. By delaying such backout work, the DB2 subsystem can be restarted more quickly. If you specify LIMIT BACKOUT = YES, then you must use the RECOVER POSTPONED command to resolve postponed units of recovery. See Part 2 of *DB2 Installation Guide* for more information about installation options.

Use the RECOVER POSTPONED command to complete postponed backout processing on *all* units of recovery; you cannot specify a single unit of work for resolution. This command might take several hours to complete depending on the content of the long-running job. In some circumstances, you can elect to use the CANCEL option of the RECOVER POSTPONED command. This option leaves the objects in an inconsistent state (REFP) that you must resolve before using the objects. However, you might choose the CANCEL option for the following reasons:

- You determine that the complete recovery of the postponed units of recovery will take more time to complete than you have available. You also determine it is faster to either recover the objects to a prior point in time or run the LOAD utility with the REPLACE option.
- You want to replace the existing data in the object with new data.
- You decide to drop the object. To drop the object successfully, complete the following steps:
  1. Issue the RECOVER POSTPONED command with the CANCEL option.
  2. Issue the DROP TABLESPACE statement.
- You do not have the DB2 logs to successfully recover the postponed units of recovery.

Errors encountered during RECOVER POSTPONED processing

If a required page cannot be accessed during RECOVER POSTPONED processing, or if any other error is encountered while attempting to apply a log record, the page set or partition is deferred and processing continues. DB2 writes a compensation log record to reflect those deferrals and places the page in the logical page list. Some errors encountered while recovering indexes cause the entire page set to be placed in the logical page list. Some errors halt the construction of the compensation log and mark the page set as RECP.

When an error prevents logging of a compensation log record, DB2 abends. If DB2 abends:

1. Fix the error.
2. Restart DB2.
3. Re-issue the RECOVER POSTPONED command if automatic backout processing has not been specified.

Output from RECOVER POSTPONED processing

Output from the RECOVER POSTPONED command consists of informational messages. In Figure 47, backout processing was performed against two table space partitions and two index partitions:

```
DSNV435I ! RESOLUTION OF POSTPONED ABORT URS HAS BEEN SCHEDULED
DSN9022I ! DSNVRP 'RECOVER POSTPONED' NORMAL COMPLETION
DSNR047I ! DSNRBMON POSTPONED ABORT BACKOUT
 PROCESSING LOG RECORD AT RBA 000002055000 TO RBA 000001E6A20E
DSNR047I ! DSNRBMON POSTPONED ABORT BACKOUT
 PROCESSING LOG RECORD AT RBA 000002049000 TO RBA 000001E6A20E
DSNI024I ! DSNIARPL BACKOUT PROCESSING HAS COMPLETED
      FOR PAGESET DSNDB04 .I PART 00000004.
DSNI024I ! DSNIARPL BACKOUT PROCESSING HAS COMPLETED
      FOR PAGESET DSNDB04 .PT PART 00000004.
DSNI024I ! DSNIARPL BACKOUT PROCESSING HAS COMPLETED
      FOR PAGESET DSNDB04 .I PART 00000002.
DSNI024I ! DSNIARPL BACKOUT PROCESSING HAS COMPLETED
      FOR PAGESET DSNDB04 .PT PART 00000002.
```

Figure 47. Example of output from RECOVER POSTPONED processing

If the RECOVER POSTPONED processing lasts for an extended period, the output includes DSNR047I messages, as shown in Figure 47, to help you monitor backout processing. These messages show the current RBA that is being processed and the target RBA.

Choosing recovery operations for conditional restart

The recovery operations that take place during restart are controlled by the currently active conditional restart control record. An active control record is created or deactivated by running the change log inventory utility with the CRESTART control statement. You can choose:

- To retain a specific portion of the log for future DB2 processing
- To read the log forward to recover indoubt and uncommitted units of recovery
- To read the log backward to back out uncommitted and in-abort units of recovery
- To do a cold start, not processing any log records.

A conditional restart record that specifies left truncation of the log causes any postponed abort units of recovery that began earlier than the truncation RBA to end without resolution. The combination of unresolved postponed abort units of recovery can cause more records than requested by the BACKODUR system parameter to be processed. The left truncation RBA takes precedence over BACKODUR in this case.

Be careful about doing a conditional restart that discards log records. If the discarded log records contain information from an image copy of the DB2 directory, a future execution of the RECOVER utility on the directory will fail. For more information, see “Recovering the catalog and directory” on page 494.

Conditional restart records

In addition to information describing the active and archive logs, the BSDS contains two queues of records associated with conditional restart.

- A wrap-around queue of conditional restart control records. Each element in the queue records the choices you made when you created the record and the progress of the restart operation it controls. When the operation is complete, the use count is set at 1 for the record and it is not used again.
- A queue of checkpoint descriptions. Because a conditional restart can specify use of a particular log record range, the recovery process cannot automatically use the most recent checkpoint. The recovery process must find the latest checkpoint within the specified range, and uses that checkpoint queue for that purpose.

Use the utility DSN1LOGP to read information about checkpoints and conditional restart control records. See Part 3 of *DB2 Utility Guide and Reference* for information about that utility.

Chapter 20. Maintaining consistency across multiple systems

This chapter explains data consistency issues which arise when DB2 acts in conjunction with other systems, either IMS, CICS, or remote DBMSs.

The following sections are presented:

"Multiple system consistency"
"Resolving indoubt units of recovery" on page 459
"Resolution of remote DBMS indoubt units of recovery" on page 463
"Consistency across more than two systems" on page 467
"Resolution of RRS indoubt units of recovery" on page 466

Multiple system consistency

DB2 can work with other database management systems, including IMS, CICS, and other types of remote DBMSs through DDF. DB2 can also work with other DB2 subsystems through the distributed data facility (DDF).

If data in more than one subsystem is to be consistent, then all update operations at all subsystems for a single logical unit of work must either be committed or backed out.

The two-phase commit process

In a distributed system, the actions of a logical unit of work might occur at more than one system. When these actions update recoverable resources, the commit process insures that either all the effects of the logical unit of work persist or that none of the effects persist, despite component, system, or communications failures.

DB2 uses a two-phase commit process in communicating between subsystems. That process is under the control of one of the subsystems, called the *coordinator*. The other systems involved are the *participants*. IMS, CICS, or WebSphere Application Server are always the coordinator when they interact with DB2. In transactions that include IMS, CICS, or WebSphere Application Server, DB2 is always a participant. DB2 is always the coordinator when it interacts with TSO and DB2 completely controls the commit process in these interactions. In interactions with other DBMSs, including other DB2 subsystems, your local DB2 can be either the coordinator or a participant.

© Copyright IBM Corp. 1982, 2008

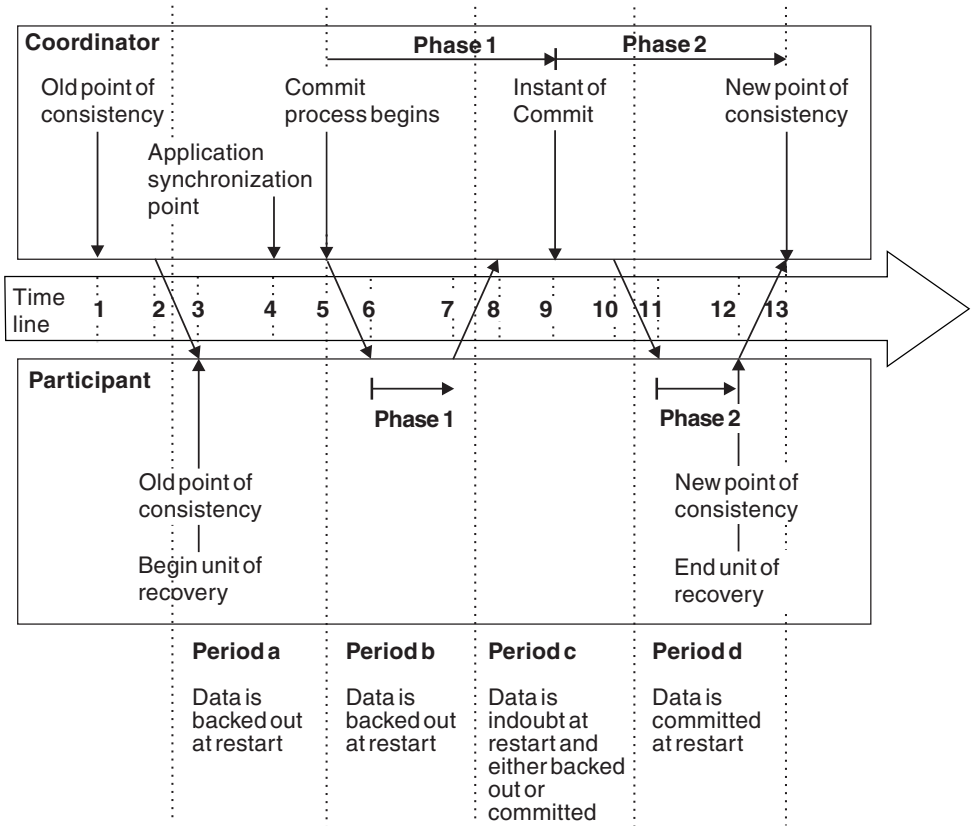


Figure 48. Time line illustrating a commit that is coordinated with another subsystem

Illustration of two-phase commit

Figure 48 illustrates the two-phase commit process. Events in the coordinator (IMS, CICS, or DB2) are shown on the upper line, events in the participant on the lower line. The numbers in the following discussion are keyed to those in the figure. The resultant state of the update operations at the participant are shown between the two lines.

1. The data in the coordinator is at a point of consistency.
2. An application program in the coordinator calls the participant to update some data, by executing an SQL statement.
3. This starts a unit of recovery in the participant.
4. Processing continues in the coordinator until an application synchronization point is reached.
5. The coordinator then starts commit processing. IMS can do that by using a DL/I CHKP call, a fast path SYNC call, a GET UNIQUE call to the I/O PCB, or a normal application termination. CICS uses a SYNCPOINT command or a normal application termination. A DB2 application starts commit processing by an SQL COMMIT statement or by normal termination. Phase 1 of commit processing begins.
6. The coordinator informs the participant that it is to prepare for commit. The participant begins phase 1 processing.
7. The participant successfully completes phase 1, writes this fact in its log, and notifies the coordinator.
8. The coordinator receives the notification.

9. The coordinator successfully completes its phase 1 processing. Now both subsystems agree to commit the data changes, because both have completed phase 1 and could recover from any failure. The coordinator records on its log the instant of commit—the irrevocable decision of the two subsystems to make the changes.

   The coordinator now begins phase 2 of the processing—the actual commitment.
10. It notifies the participant to begin its phase 2.
11. The participant logs the start of phase 2.
12. Phase 2 is successfully completed, which establishes a new point of consistency for the participant. The participant then notifies the coordinator that it is finished with phase 2.
13. The coordinator finishes its phase 2 processing. The data controlled by both subsystems is now consistent and available to other applications.

There are occasions when the coordinator invokes the participant when no participant resource has been altered since the completion of the last commit process. This can happen, for example, when SYNCPOINT is issued after performance of a series of SELECT statements or when end-of-task is reached immediately after SYNCPOINT has been issued. When this occurs, the participant performs both phases of the two-phase commit during the first commit phase and records that the user or job is read-only at the participant.

Maintaining consistency after termination or failure

If DB2 fails while acting as a coordinator, it has the appropriate information to determine commit or roll back decisions after restart. However, if DB2 fails while it acts as the participant, it must determine after restart whether to commit or to roll back units of recovery that were active at the time of the failure. For certain units of recovery, DB2 has enough information to make the decision. For others, it does not, and DB2 must get information from the coordinator when the connection is reestablished.

The status of a unit of recovery after a termination or failure depends upon the moment at which the incident occurred. Figure 48 on page 456 helps to illustrate the following possible statuses:

Status Description and Processing

Inflight
: The participant or coordinator failed before finishing phase 1 (period *a* or *b*); during restart, both systems back out the updates.

Indoubt
: The participant failed after finishing phase 1 and before starting phase 2 (period *c*); only the coordinator knows whether the failure happened before or after the commit (point *9*). If it happened before, the participant must back out its changes; if it happened afterward, it must make its changes and commit them. After restart, the participant waits for information from the coordinator before processing this unit of recovery.

In-commit
: The participant failed after it began its own phase 2 processing (period *d*); it makes committed changes.

In-abort
: The participant or coordinator failed after a unit of recovery began to be

rolled back but before the process was complete (not shown in the figure). The operational system rolls back the changes; the failed system continues to back out the changes after restart.

Postponed abort
If LIMIT BACKOUT installation option is set to YES or AUTO, any backout not completed during restart is postponed. The status of the incomplete URs is changed from inflight or in-abort to postponed abort.

Termination for multiple systems

Termination for multiple systems is like termination for single systems, but with the following additional considerations:

- Using STOP DB2 MODE(FORCE) could create indoubt units of recovery for threads that are between commit processing phases. These indoubt threads are resolved after you reconnect to the coordinator.
- Data updated by an indoubt unit of recovery is locked and unavailable for use by others. The unit could be indoubt when DB2 was stopped, or could be indoubt from an earlier termination and not yet resolved.
- A DB2 system failure can leave a unit of recovery in an indoubt state if the failure occurs between phase 1 and phase 2 of the commit process.

Normal restart and recovery for multiple systems

When DB2 acts together with another system, the recovery log contains information about units of recovery that are inflight, indoubt, in-abort, postponed abort, or in-commit. The phases of restart and recovery deal with that information as follows:

Phase 1: Log initialization

This phase proceeds as described in “Phase 1: Log initialization” on page 445.

Phase 2: Current status rebuild

While reading the log, DB2 identifies:

- The coordinator and all participants for every unit of recovery.
- All units of recovery that are outstanding and their statuses (indoubt, in-commit, in-abort, or inflight, as described under “Maintaining consistency after termination or failure” on page 457).

Phase 3: Forward log recovery

DB2 makes all database changes for each indoubt unit of recovery, and locks the data to prevent access to it after restart. Later, when an indoubt unit of recovery is resolved, processing is completed in one of these ways:

- For the ABORT option of the RECOVER INDOUBT command, DB2 reads and processes the log, reversing all changes.
- For the COMMIT option of the RECOVER INDOUBT command, DB2 reads the log but does not process the records because all changes have been made.

At the end of this phase, indoubt activity is reflected in the database as though the decision was made to commit the activity, but the activity has not yet been committed. The data is locked and cannot be used until DB2 recognizes and acts upon the indoubt decision. (For a description of indoubt units of recovery, see “Resolving indoubt units of recovery” on page 459.)

Phase 4: Backward log recovery

As described in "Phase 4: Backward log recovery" on page 448, this phase reverses changes performed for inflight or in-abort units of recovery. At the end of this phase, interrupted inflight and in-abort changes have been removed from the database (the data is consistent and can be used) or removal of the changes has been postponed (the data is inconsistent and unavailable).

If removal of the changes has been postponed, the units of recovery become known as *postponed abort* units of recovery. The data with pending backout work is in a restrictive state (restart pending) which makes the data unavailable. The data becomes available upon completion of backout work or upon cold or conditional restart of DB2.

If the LIMIT BACKOUT system parameter is AUTO, completion of the backout work begins automatically by DB2 when the system accepts new work. If the LIMIT BACKOUT system parameter is YES, completion of the backout work begins when you use the RECOVER POSTPONED command.

Restarting multiple systems with conditions

If conditional restart is performed when DB2 is acting together with other systems, the following actions occur:

1. All information about another coordinator and other participants known to DB2 is displayed by messages DSNL438I and DSNL439I.
2. This information is purged. Therefore the RECOVER INDOUBT command must be used at the local DB2 when the local location is a participant, and at another DB2 when the local location is the coordinator.
3. Indoubt database access threads continue to appear as indoubt and no resynchronization with either a coordinator or a participant is allowed.

Methods for resolving inconsistencies caused by conditional restart and implications in a distributed environment are described in "Resolving inconsistencies resulting from a conditional restart" on page 616.

Resolving indoubt units of recovery

If DB2 loses its connection to another system, it normally attempts to recover all inconsistent objects after restart. The information needed to resolve indoubt units of recovery must come from the coordinating system. This section describes the process of resolution for different types of other systems.

Check the console for message DSNR036I for unresolved units of recovery encountered during a checkpoint. This message might occur to remind operators of existing indoubt threads. See Part 2 of *DB2 Installation Guide* for details.

Important

If the TCP/IP address that is associated with a DRDA server is subject to change, the domain name of each DRDA server must be defined in the CDB. This allows DB2 to recover from situations where the server's IP address changes prior to successful resynchronization.

Resolution of IMS indoubt units of recovery

The resolution of indoubt units in IMS has no effect on DL/I resources. Because IMS is in control of recovery coordination, DL/I resources are never indoubt. When IMS restarts, it automatically commits or backs out incomplete DL/I work, based on whether or not the commit decision was recorded on the IMS log. The existence of indoubt units does not imply that DL/I records are locked until DB2 connects.

During the current status rebuild phase of DB2 restart, the DB2 participant makes a list of indoubt units of recovery. IMS builds its own list of residual recovery entries (RREs). The RREs are logged at IMS checkpoints until all entries are resolved.

When indoubt units are recovered, the following steps occur:

1. IMS either passes an RRE to the IMS attachment facility to resolve the entry or informs the attachment facility of a cold start. The attachment facility passes the required information to DB2.
2. If DB2 recognizes that an entry has been marked by DB2 for commit and by IMS for roll back, it issues message DSNM005I. DB2 issues this message for inconsistencies of this type between DB2 and IMS.
3. The IMS attachment facility passes a return code to IMS, indicating that it should either destroy the RRE (if it was resolved) or keep it (if it was not resolved). The procedure is repeated for each RRE.
4. Finally, if DB2 has any remaining indoubt units, the attachment facility issues message DSNM004I.

The IMS attachment facility writes all the records involved in indoubt processing to the IMS log tape as type X'5501FE'.

For all resolved units, DB2 updates databases as necessary and releases the corresponding locks. For threads that access offline databases, the resolution is logged and acted on when the database is started.

DB2 maintains locks on indoubt work that was not resolved. This can create a backlog for the system if important locks are being held. Use the DISPLAY DATABASE LOCKS command to find out which tables and table spaces are locked by indoubt units of recovery. The connection remains active so you can clean up the IMS RREs. Recover the indoubt threads by the methods described in “Controlling IMS connections” on page 388.

All indoubt work should be resolved unless there are software or operating problems, such as with an IMS cold start. Resolution of indoubt units of recovery from IMS can cause delays in SQL processing. Indoubt resolution by the IMS control region takes place at two times:

- At the start of the connection to DB2, during which resolution is done synchronously
- When a program fails, during which the resolution is done asynchronously.

In the first case, SQL processing is prevented in all dependent regions until the indoubt resolution is completed. IMS does not allow connections between IMS dependent regions and DB2 before the indoubt units are resolved.

Resolution of CICS indoubt units of recovery

The resolution of indoubt units has no effect on CICS resources. CICS is in control of recovery coordination and, when it restarts, CICS automatically commits or backs out each unit, depending on whether there was an end of unit of work log record. The existence of indoubt work does not lock CICS resources until DB2 connects.

A process to resolve indoubt units of recovery is initiated during start up of the attachment facility. During this process:

- The attachment facility receives a list of indoubt units of recovery for this connection ID from the DB2 participant and passes them to CICS for resolution.
- CICS compares entries from this list with entries in its own. CICS determines from its own list what action it took for the indoubt unit of recovery.
- For each entry in the list, CICS creates a task that drives the attachment facility, specifying the final commit or abort direction for the unit of recovery.
- If DB2 does not have any indoubt unit of recovery, a dummy list is passed. CICS then purges any unresolved units of recovery from previous connections, if any.

If the units of recovery cannot be resolved because of conditions described in messages DSNC001I, DSNC034I, DSNC035I, or DSNC036I, CICS enables the connection to DB2. For other conditions, it sends message DSNC016I and terminates the connection.

For all resolved units, DB2 updates databases as necessary and releases the corresponding locks. For threads that access offline databases, the resolution is logged and acted on when the database is started. Unresolved units of work can remain after restart; resolve them by the methods described in "Manually recovering CICS indoubt units of recovery" on page 527.

Resolution of WebSphere Application Server indoubt units of recovery

A *global transaction* is a unit of work that involves operations on multiple resource managers, such as DB2. All of the operations that comprise a global transaction are managed by a transaction manager, such as WebSphere Application Server. When the transaction manager receives transactionally demarcated requests from an application, it translates the requests into a series of transaction control commands, which it directs to the participating resource managers.

Example: An application requests updates to multiple databases. The transaction manager translates these update requests into transaction control commands that are passed to several resource managers. Each resource manager then performs its own set of operations on a database. When the application issues a COMMIT, the transaction manager coordinates the commit of the distributed transaction across all participating resource managers by using the two-phase commit protocol. If any resource manager is unable to complete its portion of the global transaction, the transaction manager directs all participating resource managers to roll back the operations that they performed on behalf of the global transaction.

If a communication failure occurs between the first phase (prepare) and the second phase (commit decision) of a commit, an indoubt transaction is created on the resource manager that experienced the failure. When an indoubt transaction is created, a message like this is displayed on the console of the resource manager:

```
00- 17.24.36 STC00051  DSNL405I  = THREAD
  - G91E1E35.GFA7.00F962CC4611.0001=217
  - PLACED IN INDOUBT STATE BECAUSE OF
  - COMMUNICATION FAILURE WITH COORDINATOR 9.30.30.53.
  - INFORMATION RECORDED IN TRACE RECORD WITH IFCID=209
  - AND IFCID SEQUENCE NUMBER=00000001
```

After a failure, WebSphere Application Server is responsible for resolving indoubt transactions and for driving any failure recovery. To perform these functions, the server must be restarted and the recovery process initiated by an operator. You can also manually resolve indoubt transactions with the RECOVER INDOUBT command.

Recommendation: Let WebSphere Application Server resolve the indoubt transactions. Manually recover indoubt transactions only as a last resort to get DB2 up and running, and to release locks.

To manually resolve indoubt transactions:

1. Display indoubt threads from the resource manager console:

   ```
   -DISPLAY THD(*) T(I) DETAIL
   ```

   This command produces output like this:

   ```
   00- 17.01.03          =dis thd(*) t(i) detail
     - 17.01.04 STC00051  DSNV401I  = DISPLAY THREAD REPORT FOLLOWS -
     - 17.01.04 STC00051  DSNV406I  = INDOUBT THREADS -
     - COORDINATOR               STATUS      RESET URID         AUTHID
     - 9.30.30.53:4007           INDOUBT     0000111F049A       ADMF002
     -  V437-WORKSTATION=jaijeetsvl, USERID=admf002,
     -       APPLICATION NAME=db2jccmain
     -  V440-XID=7C7146CE 00000014 00000021 F6EF6F8B
     -           F36873BE A37AC6BC 256F295D 04BE7EE0
     -           8DFEF680 D1A6EFD5 8C0E6343 67679239
     -           CC15A350 65BFB8EA 670CEBF4 E85938E0
     -           06
     -  V467-HAS LUWID G91E1E35.GFA7.00F962CC4611.0001=217
     -  V466-THREAD HAS BEEN INDOUBT FOR    00:00:17
     - DISPLAY INDOUBT REPORT COMPLETE
   ```

   Note that in this example, only one indoubt thread exists.

   A transaction is identified by a transaction identifier, called an XID. The first four bytes of the XID (in this case, 7C7146CE) identify the transaction manager. Each XID is associated with a logical unit of work ID (LUWID) at the DB2 server. Note the LUWID that is associated with each transaction, for use in the recovery step.
2. Query the transaction manager to determine whether a commit or abort decision was made for each transaction.
3. Based on the decision recorded by the transaction manager, recover each indoubt thread from the resource manager console by either committing or aborting the transaction. Specify the LUWID from the DISPLAY THREAD command.

   ```
   -RECOVER INDOUBT ACTION(COMMIT) LUWID(217)

   or:

   -RECOVER INDOUBT ACTION(ABORT) LUWID(217)
   ```

   This command produces output like this:

```
00- 17.30.21           =RECOVER INDOUBT ACTION(COMMIT) LUWID(217)
  - 17.30.22 STC00051  DSNV414I  = THREAD
  -  LUWID=G91E1E35.GFA7.00F962CC4611.0001=217 COMMIT
  - SCHEDULED
  - 17.30.22 STC00051  DSN9022I  = DSNVRI '-RECOVER INDOUBT' NORMAL COMPLETION
```

4. Display indoubt threads again from the resource manager console:

   ```
   -DISPLAY THD(*) T(I) DETAIL
   ```

   This command produces output like this:

   ```
   00- 17.03.37           =dis thd(*) t(i) detail
     - 17.03.37 STC00051  DSNV401I  = DISPLAY THREAD REPORT FOLLOWS -
     - 17.03.37 STC00051  DSNV406I  = INDOUBT THREADS -
     - COORDINATOR              STATUS      RESET URID         AUTHID
     - 9.30.30.53:4007          COMMITTED-H       0000111F049A ADMF002
     -  V437-WORKSTATION=jaijeetsvl, USERID=admf002,
     -       APPLICATION NAME=db2jccmain
     -  V440-XID=7C7146CE 00000014 00000021 F6EF6F8B
     -           F36873BE A37AC6BC 256F295D 04BE7EE0
     -           8DFEF680 D1A6EFD5 8C0E6343 67679239
     -           CC15A350 65BFB8EA 670CEBF4 E85938E0
     -           06
     -  V467-HAS LUWID G91E1E35.GFA7.00F962CC4611.0001=217
     - DISPLAY INDOUBT REPORT COMPLETE
   ```

   Notice that the transaction now appears as a heuristically committed transaction.

5. If the transaction manager does not recover the indoubt transactions in a timely manner, reset the transactions from the resource manager console to purge the indoubt thread information. Specify the IP address and port from the DISPLAY THREAD command.

   ```
   -RESET INDOUBT IPADDR(9.30.30.53:4007)FORCE
   ```

   This command produces output like this:

   ```
    00- 17.37.13           =RESET INDOUBT IPADDR(9.30.30.53:4393)FORCE
     - 17.37.13 STC00051  DSNL455I  = DB2 HAS NO INFORMATION RELATED TO
     - IPADDR 9.30.30.53:4007
     - 17.37.13 STC00051  DSNL454I  = QUALIFYING INDOUBT INFORMATION FOR
     - IPADDR 9.30.30.53:4007 HAS BEEN PURGED
   ```

Resolution of remote DBMS indoubt units of recovery

When communicating with a remote DBMS, indoubt units of recovery can result from failure with either the participant or coordinator or with the communication link between them even if the systems themselves have not failed.

Normally, if your subsystem fails while communicating with a remote system, you should wait until both systems and their communication link become operational. Your system then automatically recovers its indoubt units of recovery and continues normal operation. When DB2 restarts while any unit of recovery is indoubt, the data required for that unit remains locked until the unit of recovery is resolved.

If automatic recovery is not possible, DB2 alerts you to any indoubt units of recovery that you need to resolve. If it is imperative that you release locked resources and bypass the normal recovery process, you can resolve indoubt situations manually.

Important

In a manual recovery situation, you must determine whether the coordinator decided to commit or abort and ensure that the same decision is made at the participant. In the recovery process, DB2 attempts to automatically resynchronize with its participants. If you decide incorrectly what the coordinator's recovery action is, data is inconsistent at the coordinator and participant.

If you choose to resolve units of recovery manually, you must:

- Commit changes made by logical units of work that were committed by the other system
- Roll back changes made by logical units of work that were rolled back by the other system

Making heuristic decisions

From the DB2 point of view, a decision to commit or roll back an indoubt unit of recovery by any means but the normal resynchronization process is a *heuristic decision*. If you commit or roll back a unit of work and your decision is different from the other system's decision, data inconsistency occurs. This type of damage is called *heuristic damage*. If this situation should occur, and your system then updates any data involved with the previous unit of work, your data is corrupted and is extremely difficult to correct.

In order to make a correct decision, you must be absolutely sure that the action you take on indoubt units of recovery is the same as the action taken at the coordinator. Validate your decision with the administrator of the other systems involved with the logical unit of work.

Methods for determining the coordinator's commit or abort decision

The first step is to communicate with the other system administrator. There are several ways to ascertain the status of indoubt units at other systems:

- Use a NetView program. Write a program that analyzes NetView alerts for each involved system, and returns the results through the NetView system.
- Use an automated z/OS console to ascertain the status of the indoubt threads at the other involved systems.
- Use the command DISPLAY THREAD TYPE(INDOUBT) LUWID(*luwid*).

  If the coordinator DB2 system is started and no DB2 cold start was performed, then DISPLAY THREAD TYPE(INDOUBT) can be used. If the decision was to commit, the display thread indoubt report includes the LUWID of the indoubt thread. If the decision was to abort, the thread is not displayed.
- Read the recovery log using DSN1LOGP.

  If the coordinator DB2 cannot be started, then DSN1LOGP can determine the commit decision. If the coordinator DB2 performed a cold start (or any type of conditional restart), then the system log should contain messages DSNL438I or DSNL439I, which describe the status of the unit of recovery (LUWID).

Displaying information about indoubt threads

Use DISPLAY THREAD TYPE(INDOUBT) to find information about allied and database access indoubt threads. The command provides information about threads where DB2 is a participant, a coordinator, or both. The TYPE(INDOUBT) option

tells you which systems still need indoubt resolution and provides the LUWIDs you need to recover indoubt threads. A thread that has completed phase 1 of commit and still has a connection with its coordinator is in the *prepared* state and is displayed as part of the display thread active report. If a prepared thread loses its connection with its coordinator, it enters the *indoubt* state and terminates its connections to any participants at that time. Any threads in the prepared or indoubt state when DB2 terminates are indoubt after DB2 restart. However, if the participant system is waiting for a commit or roll back decision from the coordinator, and the connection is still active, DB2 considers the thread active.

If a thread is indoubt at a participant, you can determine whether a commit or abort decision was made at the coordinator by issuing the DISPLAY THREAD command at the coordinator as described previously. If an indoubt thread appears at one system and does not appear at the other system, then the latter system backed out the thread, and the first system must therefore do the same. See "Monitoring threads" on page 379 for examples of output from the DISPLAY THREAD command.

Recovering indoubt threads

After you determine whether you need to commit or roll back an indoubt thread, recover it with the RECOVER INDOUBT command. This command does not erase the thread's indoubt status. It still appears as an indoubt thread until the systems go through the normal resynchronization process. An indoubt thread can be identified by its LUWID, LUNAME or IP address. You can also use the LUWID's token with the command.

Committing or rolling back: Use the ACTION(ABORT|COMMIT) option of RECOVER INDOUBT to commit or roll back a logical unit of work. If your system is the coordinator of one or more other systems involved with the logical unit of work, your action propagates to the other system associated with the LUW.

For example, to recover two indoubt threads, the first with LUWID=DB2NET.LUNSITE0.A11A7D7B2057.0002 and the second with a token of 442, and commit the LUWs, enter:

```
-RECOVER INDOUBT ACTION(COMMIT) LUWID(DB2NET.LUNSITE0.A11A7D7B2057.0002,442)
```

Detailed scenarios describing indoubt thread resolution can be found in "Resolving indoubt threads" on page 582.

Resetting the status of an indoubt thread

After manual recovery of an indoubt thread, allow the systems to resynchronize automatically; this resets the status of the indoubt thread. However, if heuristic damage or a protocol error occurs, you must use the RESET INDOUBT command to delete indoubt thread information for a thread whose *reset* status is *yes*. DB2 maintains this information until normal automatic recovery. You can purge information about threads where DB2 is either the coordinator or participant. If the thread is an allied thread connected with IMS or CICS, the command only applies to coordinator information about downstream participants. Information that is purged does not appear in the next display thread report and is erased from DB2's logs.

For example, to purge information on two indoubt threads, the first with an LUWID=DB2NET.LUNSITE0.A11A7D7B2057.0002 and a resync port number of 123; and the second with a token of 442, enter:

```
-RESET INDOUBT LUWID(DB2NET.LUNSITE0.A11A7D7B2057.0002:123,442)
```

You can also use a LUNAME or IP address with the RESET INDOUBT command. A new keyword (IPADDR) can be used in place of LUNAME or LUW keywords, when the partner uses TCP/IP instead of SNA. The partner's resync port number is required when using the IP address. The DISPLAY THREAD output lists the resync port number. This allows you to specify a location, instead of a particular thread. You can reset all the threads associated with that location with the (*) option.

Resolution of RRS indoubt units of recovery

Sometimes a DB2 unit of recovery (for a thread that uses RRSAF) or an RRS unit of recovery (for a stored procedure) enters the INDOUBT state. This is a state where a failure occurs when the participant (DB2 for a thread that uses RRSAF or RRS for a stored procedure) has completed phase 1 of commit processing and is waiting for the decision from the commit coordinator. This failure could be a DB2 abnormal termination, an RRS abnormal termination, or both.

Normally, automatic resolution of indoubt units of recovery occurs when DB2 and RRS reestablish communication with each other. If something prevents this, then you can manually resolve an indoubt unit of recovery. This process is not recommended because it might lead to inconsistencies in recoverable resources.

The following errors make manual recovery necessary:

- An RRS cold start where the RRS log is lost.

  If DB2 is a participant and has one or more indoubt threads, then these indoubt threads must be manually resolved in order to commit or abort the database changes and to release database locks. If DB2 is a coordinator for an RRS unit of recovery, then DB2 knows the commit/abort decision but cannot communicate this information to the RRS compliant resource manager that has an indoubt unit of recovery.
- If DB2 performs a conditional restart and loses information from its log, then there might be inconsistent DB2 managed data.
- In a Sysplex, if DB2 is restarted on an z/OS system where RRS is not installed, then DB2 might have indoubt threads.

  This is a user error because RRS must be started on all processors in a Sysplex on which RRS work is to be performed.

Both DB2 and RRS can display information about indoubt units of recovery. Both also provide techniques for manually resolving these indoubt units of recovery.

In DB2, the DISPLAY THREAD command provides information about indoubt DB2 thread. The display output includes RRS unit of recovery IDs for those DB2 threads that have RRS either as a coordinator or as a participant. If DB2 is a participant, the RRS unit of recovery ID displayed can be used to determine the outcome of the RRS unit of recovery. If DB2 is the coordinator, you can determine the outcome of the unit of recovery from the DISPLAY THREAD output.

In DB2, the RECOVER INDOUBT command lets you manually resolve a DB2 indoubt thread. You can use RECOVER INDOUBT to commit or roll back a unit of recovery after you determine what the correct decision is.

RRS provides an ISPF interface that provides a similar capability.

Consistency across more than two systems

The principles and methods for maintaining consistency across more than two systems are similar to those used to ensure consistency across two systems. The main difference involves a system's role as coordinator or participant when a unit of work spans multiple systems.

Commit coordinator and multiple participants

The coordinator of a unit of work that involves two or more other DBMSs must ensure that all systems remain consistent. After the first phase of the two-phase commit process, the DB2 coordinator waits for the other participants to indicate that they can commit the unit of work. If all systems are able, the DB2 coordinator sends the commit decision and each system commits the unit of work.

If even one system indicates that it cannot commit, then the DB2 coordinator sends out the decision to roll back the unit of work at all systems. This process ensures that data among multiple DBMSs remains consistent. When DB2 is the participant, it follows the decision of the coordinator, whether the coordinator is another DB2 or another DBMS.

DB2 is always the participant when interacting with IMS or CICS systems, or WebSphere Application Server. However, DB2 can also serve as the coordinator for other DBMSs or DB2 subsystems in the same unit of work. For example, if DB2 receives a request from a coordinating system that also requires data manipulation on another system, DB2 propagates the unit of work to the other system and serves as the coordinator for that system.

In Figure 49, DB2A is the participant for an IMS transaction, but becomes the coordinator for the two database servers (AS1 and AS2), DB2B, and its respective DB2 servers (DB2C, DB2D, and DB2E).

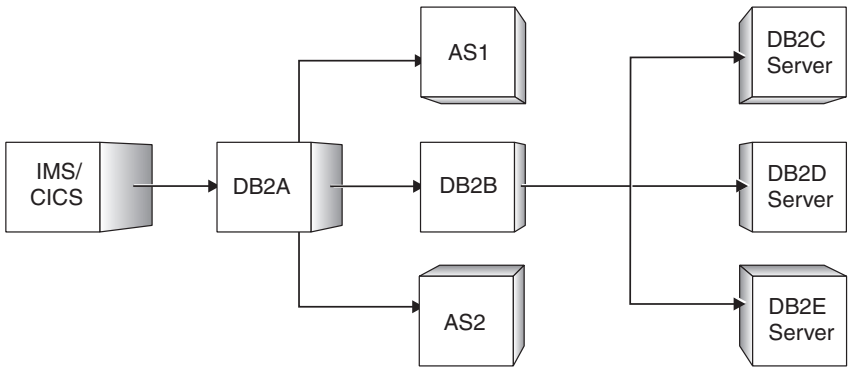


Figure 49. Illustration of multi-site unit of work

If the connection goes down between DB2A and the coordinating IMS system, the connection becomes an indoubt thread. However, DB2A's connections to the other systems are still waiting and are not considered indoubt. Wait for automatic recovery to occur to resolve the indoubt thread. When the thread is recovered, the unit of work commits or rolls back and this action is propagated to the other systems involved in the unit of work.

Illustration of multi-site update

Figure 50 illustrates a multi-site update involving one coordinator and two participants.

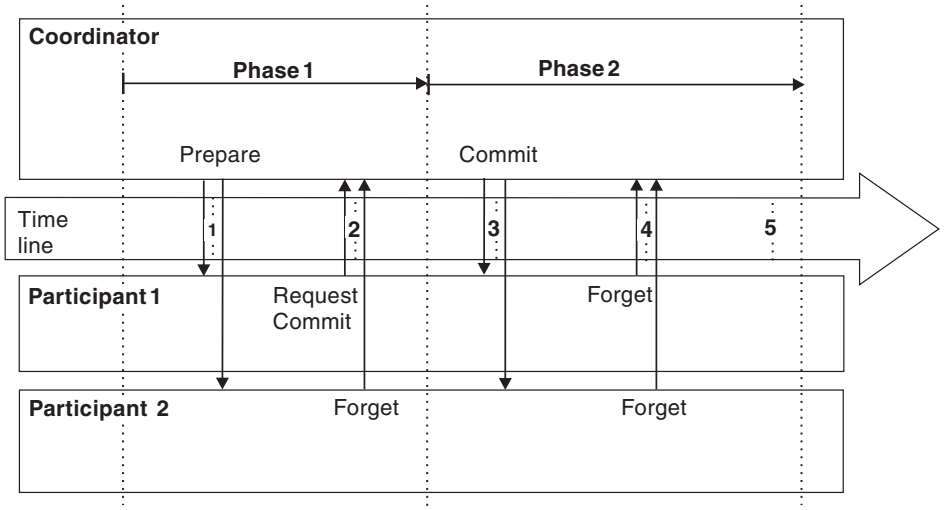


Figure 50. Illustration of multi-site update. C is the coordinator; P1 and P2 are the participants.

The following process describes each action that Figure 50 illustrates.

Phase 1:

1. When an application commits a logical unit of work, it signals the DB2 coordinator. The coordinator starts the commit process by sending messages to the participants to determine whether they can commit.
2. A participant (*Participant 1*) that is willing to let the logical unit of work be committed, and which has updated recoverable resources, writes a log record. It then sends a request commit message to the coordinator and waits for the final decision (commit or roll back) from the coordinator. The logical unit of work at the participant is now in the prepared state.

   If a participant (*Participant 2*) has not updated recoverable resources, it sends a forget message to the coordinator, releases its locks and forgets about the logical unit of work. A read-only participant writes no log records. As far as this participant is concerned, it does not matter whether the logical unit of work ultimately gets rolled back or committed.

   If a participant wants to have the logical unit of work rolled back, it writes a log record and sends a message to the coordinator. Because a message to roll back acts like a veto, the participant in this case knows that the logical unit of work will be rolled back by the coordinator. The participant does not need any more information from the coordinator and therefore rolls back the logical unit of work, releases its locks, and forgets about the logical unit of work. (This case is not illustrated in the figure.)

Phase 2:

3. After the coordinator receives request commit or forget messages from all its participants, it starts the second phase of the commit process. If at least one of the responses is request commit, the coordinator writes a log record and sends committed messages to all the participants who responded to the prepare

message with request commit. If neither the participants nor the coordinator have updated any recoverable resources, there is no second phase and no log records are written by the coordinator.

4. Each participant, after receiving a committed message, writes a log record, sends a response to the coordinator, and then commits the logical unit of work.

   If any participant responds with a roll back message, the coordinator writes a log record and sends a roll back message to all participants. Each participant, after receiving a roll back message writes a log record, sends an acknowledgment to the coordinator, and then rolls back the logical unit of work. (This case is not illustrated in the figure.)

5. The coordinator, after receiving the responses from all the participants that were sent a message in the second phase, writes an 'end' record and forgets the logical unit of work.

Important: If you try to resolve any indoubt threads manually, you need to know whether the participants committed or rolled back their units of work. With this information, you can make an appropriate decision regarding processing at your site.

Chapter 21. Backing up and recovering databases

DB2 provides means for recovering data to its current state or to an earlier state. The units of data that can be recovered are table spaces, indexes, index spaces, partitions, and data sets.

This chapter explains the following topics:

"Planning for backup and recovery"
"Copying page sets and data sets" on page 489
"Recovering page sets and data sets" on page 491
"Recovering the catalog and directory" on page 494
"Recovering data to a prior point of consistency" on page 495
"Recovery of dropped objects" on page 505
"Discarding SYSCOPY and SYSLGRNX records" on page 510
"System-level point in time recovery" on page 511

For all commands and utility statements, the complete syntax and parameter descriptions can be found in *DB2 Command Reference* and *DB2 Utility Guide and Reference*.

Planning for backup and recovery

Development at your site of backup and recovery procedures is critical in order to avoid costly and time-consuming losses of data. You should develop procedures to:

- Create a point of consistency
- Restore system and data objects to a point of consistency
- Back up the DB2 catalog and directory and your data
- Recover the DB2 catalog and directory and your data
- Recover from out-of-space conditions
- Recover from a hardware or power failure
- Recover from an z/OS component failure

In addition, you should consider a procedure for offsite recovery in case of a disaster.

To improve recovery capability in the event of a disk failure, it is advisable to use dual active logging and to place the copies of the active log data sets and the bootstrap data sets on different disk volumes. These concepts are described in "Establishing the logging environment" on page 425.

The principal tools for recovery are the utilities QUIESCE, REPORT, COPY, RECOVER, and MERGECOPY. This section also gives an overview of these utilities to help you with your backup and recovery planning.

This section covers the following topics, which you should consider when you plan for backup and recovery:

- "Considerations for recovering distributed data" on page 472
- "Considerations for recovering indexes" on page 473
- "Preparing for recovery" on page 473
- "Maximizing data availability during backup and recovery" on page 477
- "Events that occur during recovery" on page 474
- "How to find recovery information" on page 480
- "Preparing to recover to a prior point of consistency" on page 481

© Copyright IBM Corp. 1982, 2008

- "Preparing to recover the entire DB2 subsystem to a prior point in time" on page 482
- "Preparing for disaster recovery" on page 483
- "Ensuring more effective recovery from inconsistency" on page 486
- "Running the RECOVER utility in parallel" on page 488
- "Reading the log without RECOVER" on page 489

Considerations for recovering distributed data

Using distributed data, no matter where a unit of work originates, the unit is processed as a whole at the target systems. At a DB2 server, the entire unit is either committed or rolled back. It is not processed if it violates referential constraints defined within the target system. Whatever changes it makes to data are logged. A set of related table spaces can be quiesced at the same point in the log, and image copies can be made of them simultaneously. If that is done, and if the log is intact, any data can be recovered after a failure and be internally consistent.

However, DB2 provides no special means to coordinate recovery between different subsystems even if an application accesses data in several systems. To guarantee consistency of data between systems, write your applications, as usual, to do all related updates within one unit of work.

Point in time recovery (to the last image copy or to an RBA) presents other problems. You cannot control a utility in one subsystem from another subsystem. In practice, you cannot quiesce two sets of table spaces, or make image copies of them, in two different subsystems at exactly the same instant. Neither can you recover them to exactly the same instant, because there are two different logs, and a relative byte address (RBA) does not mean the same thing for both of them.

In planning, then, the best approach is to consider carefully what the QUIESCE, COPY, and RECOVER utilities do for you and then plan not to place data that must be closely coordinated on separate subsystems. After that, recovery planning is a matter of agreement among database administrators at separate locations.

Because DB2 is responsible for recovering DB2 data only, it does not recover non-DB2 data. Non-DB2 systems do not always provide equivalent recovery capabilities.

Extended recovery facility (XRF) toleration

DB2 can be used in an XRF recovery environment with CICS or IMS. All DB2-owned data sets (executable code, the DB2 catalog, user databases) must be on disk shared between the primary and alternate XRF processors. In the event of an XRF recovery, DB2 must be stopped on the primary processor and started on the alternate. For CICS that can be done automatically, by using the facilities provided by CICS, or manually, by the system operator. For IMS, that is a manual operation and must be done after the coordinating IMS system has completed the processor switch. In that way, any work that includes SQL can be moved to the alternate processor with the remaining non-SQL work. Other DB2 work (interactive or batch SQL and DB2 utilities) must be completed or terminated before DB2 can be switched to the alternate processor. Consider the effect of this potential interruption carefully when planning your XRF recovery scenarios.

Care must be taken to prevent DB2 from being started on the alternate processor until the DB2 system on the active, failing processor terminates. A premature start can cause severe integrity problems in data, the catalog, and the log. The use of global resource serialization (GRS) helps avoid the integrity problems by

preventing simultaneous use of DB2 on the two systems. The bootstrap data set (BSDS) must be included as a protected resource, and the primary and alternate XRF processors must be included in the GRS ring.

Considerations for recovering indexes

To recover DB2 indexes to currency, you can use the REBUILD INDEX utility or the RECOVER utility. The REBUILD INDEX utility reconstructs the indexes by reading the appropriate rows in the table space, extracting the index keys, sorting the keys, and then loading the index keys into the index. The RECOVER utility recovers indexes by restoring an image copy of the index and then applying the log.

You can use the REBUILD INDEX utility to recover any index and you not need to prepare image copies of those indexes.

To use the RECOVER utility to recover indexes, you must include the following actions in your normal database operation:

- Create or alter indexes using the SQL statement ALTER INDEX with the option COPY YES before you can copy and recover them.
- Create image copies of all indexes that you plan to recover using the RECOVER utility. The COPY utility makes full image copies or concurrent copies of indexes. Incremental copies of indexes is not supported. If full image copies of the index are taken at timely intervals, recovering a large index might be faster than rebuilding the index.

Tip: You can recover indexes and table spaces in a single list when you use the RECOVER utility. If you use a single list of objects in one RECOVER utility control statement, the logs for all of the indexes and table spaces are processed in one pass.

Preparing for recovery

The RECOVER utility supports the recovery of table spaces or index spaces. In the discussions about recovery in this chapter, the term page set can be a table space, index space, or any combination of these.

DB2 can recover a page set by using a backup copy or the recovery log or both. The DB2 recovery log contains a record of all changes made to the page set. If DB2 fails, it can recover the page set by restoring the backup copy and applying the log changes to it from the point of the backup copy.

The DB2 catalog and directory page sets must be copied at least as frequently as the most critical user page sets. Moreover, it is your responsibility to periodically copy the tables in the communications database (CDB), the application registration table, the object registration table, and the resource limit facility (governor), or to maintain the information necessary to re-create them. Plan your backup strategy accordingly.

A recovery preparation scenario: The following backup scenario suggests how DB2 utilities might be used:

Imagine that you are the database administrator for DBASE1. Table space TSPACE1 in DBASE1 has been available all week. On Friday, a disk write operation for TSPACE1 fails. You need to recover the table space to the last consistent point

before the failure occurred. You can do that because you have regularly followed a cycle of preparations for recovery. The most recent cycle began on Monday morning.

Monday morning: You start the DBASE1 database and make a *full image copy* of TSPACE1 and all indexes immediately. That gives you a starting point from which to recover. Use the COPY utility with the SHRLEVEL CHANGE option to improve availability. See Part 2 of *DB2 Utility Guide and Reference* for more information about the COPY utility.

Tuesday morning: You run COPY again. This time you make an incremental image copy to record only the changes made since the last full image copy that you took on Monday. You also make a full index copy.

TSPACE1 can be accessed and updated while the image copy is being made. For maximum efficiency, however, you schedule the image copies when online use is minimal.

Wednesday morning: You make another incremental image copy, and then create a full image copy by using the MERGECOPY utility to merge the incremental image copy with the full image copy.

Thursday and Friday mornings: You make another incremental image copy and a full index copy each morning.

Friday afternoon: An unsuccessful write operation occurs and you need to recover the table space. Run the RECOVER utility, as described in Part 2 of *DB2 Utility Guide and Reference*. The utility restores the table space from the full image copy made by MERGECOPY on Wednesday and the incremental image copies made on Thursday and Friday, and includes all changes made to the recovery log since Friday morning.

Later Friday afternoon: The RECOVER utility issues a message announcing that it has successfully recovered TSPACE1 to current point in time.

This imaginary scenario is somewhat simplistic. You might not have taken daily incremental image copies on just the table space that failed. You might not ordinarily recover an entire table space. However, it illustrates this important point: with proper preparation, recovery from a failure is greatly simplified.

Events that occur during recovery

Figure 51 on page 475 shows an overview of the recovery process. To recover a page set, the RECOVER utility typically uses these items:

- A full image copy; that is, a complete copy of the page set
- For table spaces only, any later incremental image copies; each summarizes all the changes made to the table space from the time the previous image copy was made
- All log records for the page set that were created since the most recent image copy.

The RECOVER utility uses information in the SYSIBM.SYSCOPY catalog table to:

- Restore the page set with the data in the most recent full image copy (appearing, in Figure 51 on page 475, at X'40000').
- For table spaces only, apply all the changes summarized in any later incremental image copies. (There are two in Figure 51: X'80020' and X'C0040'.)

- Apply all changes to the page set that are registered in the log, beginning at the log RBA of the most recent image copy. (In Figure 51, X'C0040', that address is also stored in the SYSIBM.SYSCOPY catalog table.)

If the log has been damaged or discarded, or if data has been changed erroneously and then committed, you can recover to a particular point in time by limiting the range of log records to be applied by the RECOVER utility.

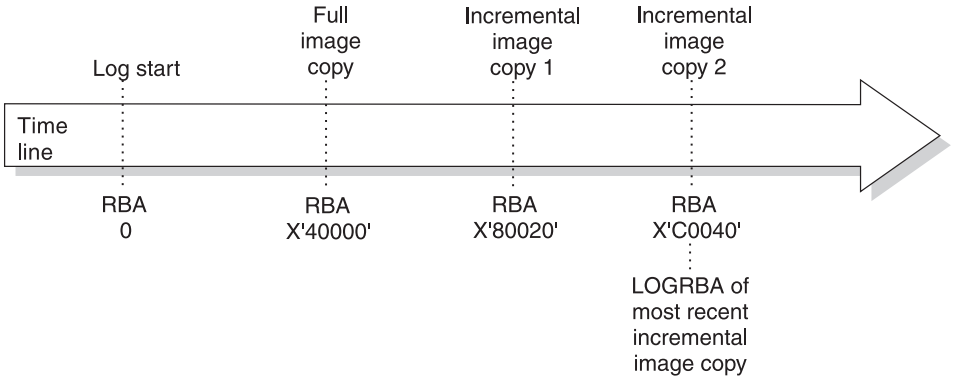


Figure 51. Overview of DB2 recovery. The figure shows one complete cycle of image copies; the SYSIBM.SYSCOPY catalog table can record many complete cycles.

Complete recovery cycles

In planning for recovery, you determine how often to take image copies and how many complete cycles to keep. Those values tell how long you must keep log records and image copies for database recovery.

In deciding how often to take image copies, consider the time needed to recover a table space. It is determined by all of the following factors:

- The amount of log to traverse
- The time it takes an operator to mount and remove archive tape volumes
- The time it takes to read the part of the log needed for recovery
- The time needed to reprocess changed pages

In general, the more often you make image copies, the less time recovery takes; but, of course, the more time is spent making copies. If you use LOG NO without the COPYDDN keyword when you run the LOAD or REORG utilities, DB2 places the table space in copy pending status. You must remove the copy pending status of the table space by making an image copy before making further changes to the data. However, if you run REORG or LOAD REPLACE with the COPYDDN keyword, DB2 creates a full image copy of a table space during execution of the utility, so DB2 does not place the table space in copy pending status. Inline copies of indexes during LOAD and REORG are not supported.

If you use LOG YES and log all updates for table spaces, then an image copy of the table space is not required for data integrity. However, taking an image copy makes the recovery process more efficient. The process is even more efficient if you use MERGECOPY to merge incremental image copies with the latest full image copy. You can schedule the MERGECOPY operation at your own convenience, whereas the need for a recovery can come upon you unexpectedly. The MERGECOPY operation does not apply to indexes.

Recommendation: Copy your indexes after the associated utility has run. Indexes are placed in informational copy pending (ICOPY) status after running LOAD TABLESPACE, REORG TABLESPACE, REBUILD INDEX, or REORG INDEX

utilities. Only structural modifications of the index are logged when these utilities are run, so there is not enough information in the log to recover the index.

Use the CHANGELIMIT option of the COPY utility to let DB2 determine when an image copy should be performed on a table space and whether a full or incremental copy should be taken. Use the CHANGELIMIT and REPORTONLY options together to let DB2 recommend what types of image copies to make. When you specify both CHANGELIMIT and REPORTONLY, DB2 makes no image copies. The CHANGELIMIT option does not apply to indexes.

In determining how many complete copy and log cycles to keep, you are guarding against damage to a volume containing an important image copy or a log data set. A retention period of at least two full cycles is recommended. For further security, keep records for three or more copy cycles.

A recovery cycle example

Table 98 suggests how often a user group with 10 locally defined table spaces (one table per table space) might take image copies, based on frequency of updating. Their least-frequently copied table is EMPSALS, containing employee salary data. If they choose to keep two complete image copy cycles on hand, then each time they copy EMPSALS they can delete records prior to its previous copy or copies, made two months ago. They will always have on hand between two months and four months of log records.

In the example, the user's most critical tables are copied daily. Hence, the DB2 catalog and directory are also copied daily.

Table 98. DB2 log management example

Table space name	Content	Update activity	Full image copy period
ORDERINF	Invoice line: part and quantity ordered	Heavy	Daily
SALESINF	Invoice description	Heavy	Daily
SALESQTA	Quota information for each sales person	Moderate	Weekly
SALESDSC	Customer descriptions	Moderate	Weekly
PARTSINV	Parts inventory	Moderate	Weekly
PARTSINF	Parts suppliers	Light	Monthly
PARTS	Parts descriptions	Light	Monthly
SALESCOM	Commission rates	Light	Monthly
EMPLOYEE	Employee descriptive data	Light	Monthly
EMPSALS	Employee salaries	Light	Bimonthly

If you do a full recovery, you do not need to recover the indexes unless they are damaged. If you recover to a prior point in time, you do need to recover the indexes. See "Considerations for recovering indexes" on page 473 for information about indexes.

How DFSMShsm affects your recovery environment

The Data Facility Hierarchical Storage Manager (DFSMShsm) can automatically manage space and data availability among storage devices in your system. If you use it, you need to know that it automatically moves data to and from the DB2 databases.

DFSMShsm manages your disk space efficiently by moving data sets that have not been used recently to less expensive storage. It also makes your data available for recovery by automatically copying new or changed data sets to tape or disk. It can delete data sets, or move them to another device. Its operations occur daily, at a specified time, and allow for keeping a data set for a predetermined period before deleting or moving it.

All DFSMShsm operations can also be performed manually. *z/OS DFSMShsm Managing Your Own Data* tells how to use the DFSMShsm commands.

DFSMShsm:
- Uses cataloged data sets
- Operates on user tables, image copies, and logs
- Supports VSAM data sets

If a volume has a DB2 storage group specified, the volume should only be recalled to like devices of the same VOLSER defined by CREATE or ALTER STOGROUP.

DB2 can recall user page sets that have been migrated. Whether DFSMShsm recall occurs automatically is determined by the values of the RECALL DATABASE and RECALL DELAY fields of installation panel DSNTIPO. If the value of the RECALL DATABASE field is NO, automatic recall is not performed and the page set is considered an unavailable resource. It must be recalled explicitly before it can be used by DB2. If the value of the RECALL DATABASE field is YES, DFSMShsm is invoked to recall the page sets automatically. The program waits for the recall for the amount of time specified by the RECALL DELAY parameter. If the recall is not completed within that time, the program receives an error message indicating the page set is unavailable but that recall was initiated.

The deletion of DFSMShsm migrated data sets and the DB2 log retention period must be coordinated with use of the MODIFY utility. If not, you could need recovery image copies or logs that have been deleted. See “Discarding archive log records” on page 438 for suggestions.

Maximizing data availability during backup and recovery

You need to develop a plan for backup and recovery, and you need to become familiar enough with that plan so that when an outage occurs, you can get back in operation as quickly as possible. This topic contains some factors to consider when you develop and implement your plan.

Decide on the level of availability you need: To do this, start by determining the primary types of outages you are likely to experience. Then, for each of those types of outages, decide on the maximum amount of time that you can spend on recovery. Consider the trade-off between cost and availability. Recovery plans for continuous availability are very costly, so you need to think about what percentage of the time your systems really need to be available.

Practice for recovery: You cannot know whether a backup and recovery plan is workable unless you practice it. In addition, the pressure of a recovery situation

can cause mistakes. The best way to minimize mistakes is to practice your recovery scenario until you know it well. The best time to practice is outside of regular working hours, when fewer key applications are running.

Minimize preventable outages: One aspect of your backup and recovery plan should be eliminating the need to recover whenever possible. One way to do that is to prevent outages caused by errors in DB2. Be sure to check available maintenance often and apply fixes for problems that are likely to cause outages.

Determine the required backup frequency: Use your recovery criteria to decide how often to make copies of your databases. For example, if the maximum acceptable recovery time after you lose a volume of data is two hours, your volumes typically hold about 4 GB of data, and you can read about 2 GB of data per hour, then you should make copies after every 4 GB of data written. You can use the COPY option SHRLEVEL CHANGE or DFSMSdss concurrent copy to make copies while transactions and batch jobs are running. You should also make a copy after running jobs that make large numbers of changes. In addition to copying your table spaces, you should also consider copying your indexes.

You can make additional backup image copies from a primary image copy by using the COPYTOCOPY utility. This capability is especially useful when the backup image is copied to a remote site that is to be used as a disaster recovery site for the local site. Applications can run concurrently with the COPYTOCOPY utility. Only utilities that write to the SYSCOPY catalog table cannot run concurrently with COPYTOCOPY.

Minimize the elapsed time of RECOVER jobs: The RECOVER utility supports the recovery of a list of objects in parallel. For those objects in the list that can be processed independently, multiple subtasks are created to restore the image copies for the objects. The parallel function can be used for either disk or tape.

Minimize the elapsed time for copy jobs: You can use the COPY utility to make image copies of a list of objects in parallel. Image copies can be made to either disk or tape.

Determine the right characteristics for your logs:

- If you have enough disk space, use more and larger active logs. Recovery from active logs is quicker than from archive logs.
- To speed recovery from archive logs, consider archiving to disk.
- If you archive to tape, be sure you have enough tape drives that DB2 does not have to wait for an available drive on which to mount an archive tape during recovery.
- Make the buffer pools and the log buffers large enough to be efficient.

Minimize DB2 restart time: Many recovery processes involve restart of DB2. You need to minimize the time that DB2 shutdown and startup take.

For non-data-sharing systems, you can limit the backout activity during DB2 system restart. You can postpone the backout of long running URs until after the DB2 system is operational. See “Deferring restart processing” on page 450 for an explanation of how to use the installation options LIMIT BACKOUT and BACKOUT DURATION to determine what backout work will be delayed during restart processing.

These are some major factors that influence the speed of DB2 shutdown:

- Number of open DB2 data sets

  During shutdown, DB2 must close and deallocate all data sets it uses if the fast shutdown feature has been disabled. The default is to use the fast shutdown feature. Contact your IBM software support representative for information about enabling and disabling the fast shutdown feature. The maximum number of concurrently open data sets is determined by the DB2 subsystem parameter DSMAX. Closing and deallocation of data sets generally takes .1 to .3 seconds per data set. See Part 5 (Volume 2) of *DB2 Administration Guide* for information about how to choose an appropriate value for DSMAX.

  Be aware that z/OS global resource serialization (GRS) can increase the time to close DB2 data sets. If your DB2 data sets are not shared among more than one z/OS system, set the GRS RESMIL parameter value to OFF or place the DB2 data sets in the SYSTEMS exclusion RNL. See *z/OS MVS Planning: Global Resource Serialization* for details.

- Active threads

  DB2 cannot shut down until all threads have terminated. Issue the DB2 command DISPLAY THREAD to determine if there are any active threads while DB2 is shutting down. If possible, cancel those threads.

- Processing of SMF data

  At DB2 shutdown, z/OS does SMF processing for all DB2 data sets opened since DB2 startup. You can reduce the time that this processing takes by setting the z/OS parameter DDCONS(NO).

These major factors influence the speed of DB2 startup:

- DB2 checkpoint interval

  The DB2 checkpoint interval creates a number of log records that DB2 writes between successive checkpoints. This value is controlled by the DB2 subsystem parameter CHKFREQ. The default of 50000 results in the fastest DB2 startup time in most cases.

  You can use the LOGLOAD or CHKTIME option of the SET LOG command to modify the CHKFREQ value dynamically without recycling DB2. The value you specify depends on your restart requirements. See "Dynamically changing the checkpoint frequency" on page 433 for examples of how you might use these command options. See Chapter 2 of *DB2 Command Reference* for detailed information about the SET LOG command.

- Long running units of work

  DB2 rolls back uncommitted work during startup. The amount of time for this activity is roughly double the time that the unit of work was running before DB2 shut down. For example, if a unit of work runs for two hours before a DB2 abend, it will take at least four hours to restart DB2. Decide how long you can afford for startup, and avoid units of work that run for more than half that long.

  You can use accounting traces to detect long running units of work. For tasks that modify tables, divide the elapsed time by the number of commit operations to get the average time between commit operations. Add commit operations to applications for which this time is unacceptable.

  Recommendation: To detect long running units of recovery, enable the UR CHECK FREQ option of installation panel DSNTIPL. If long running units of recovery are unavoidable, consider enabling the LIMIT BACKOUT option on installation panel DSNTIPL.

- Size of active logs

  If you archive to tape, you can avoid unnecessary startup delays by making each active log big enough to hold the log records for a typical unit of work. This

lessens the probability that DB2 will have to wait for tape mounts during startup. See Part 5 (Volume 2) of *DB2 Administration Guide* for more information about choosing the size of the active logs.

How to find recovery information

This section contains guidance on locating and reporting information needed for recovery.

Where recovery information resides

Information needed for recovery is contained in these locations:

- SYSIBM.SYSCOPY, a catalog table, contains information about full and incremental image copies. If concurrent updates were allowed when making the copy, the log RBA corresponds to the image copy start time; otherwise, it corresponds to the end time. The RECOVER utility uses the log RBA to look for log information after restoring the image copy. The SYSCOPY catalog table also contains information recorded by the COPYTOCOPY utility.

  SYSCOPY also contains entries with the same kinds of log RBAs recorded by the utilities QUIESCE, REORG, LOAD, REBUILD INDEX, RECOVER TOCOPY, and RECOVER TOLOGPOINT. For a summary of the information contained in the DB2 catalog tables, see Appendix F of *DB2 SQL Reference*.

  When the REORG utility is used, the time at which DB2 writes the log RBA to SYSIBM.SYSCOPY depends on the value of the SHRLEVEL parameter:

  – For SHRLEVEL NONE, the log RBA is written at the end of the reload phase.

    If a failure occurs before the end of the reload phase, the RBA is not written to SYSCOPY.

    If a failure occurs after the reload phase is complete (and thus, after the log RBA is written to SYSCOPY), the RBA is not backed out of SYSCOPY.

  – For SHRLEVEL REFERENCE and SHRLEVEL CHANGE, the log RBA is written at the end of the switch phase.

    If a failure occurs before the end of the switch phase, the RBA is not written to SYSCOPY.

    If a failure occurs after the switch phase is complete (and thus, after the log RBA is written to SYSCOPY), the RBA is not backed out of SYSCOPY.

  The log RBA is put in SYSCOPY whether the LOG option is YES or NO, or whether the UNLOAD PAUSE option is indicated.

  When DSNDB01.DBD01, DSNDB01.SYSUTILX, and DSNDB06.SYSCOPY are quiesced or copied, a SYSCOPY record is created for each table space and any associated index that has the COPY YES attribute. For recovery reasons, the SYSCOPY records for these three objects are placed in the log.

- SYSIBM.SYSLGRNX, a directory table, contains records of the log RBA ranges used during each period of time that any recoverable page set is open for update. Those records speed recovery by limiting the scan of the log for changes that must be applied.

  If you discard obsolete image copies, you should consider removing their records from SYSIBM.SYSCOPY and the obsolete log ranges from SYSIBM.SYSLGRNX. “Discarding SYSCOPY and SYSLGRNX records” on page 510 describes the process.

Reporting recovery information

You can use the REPORT utility in planning for recovery. REPORT provides information necessary for recovering a page set. REPORT displays:

- Recovery information from the SYSIBM.SYSCOPY catalog table

- Log ranges of the table space from the SYSIBM.SYSLGRNX directory
- Archive log data sets from the bootstrap data set
- The names of all members of a table space set

You can also use REPORT to obtain recovery information about the catalog and directory.

Details about the REPORT utility and examples showing the results obtained when using the RECOVERY option are contained in Part 2 of *DB2 Utility Guide and Reference*.

Preparing to recover to a prior point of consistency

The major steps in preparing to recover to a particular point in time are:
1. Release the data from any exception status.
2. Copy the data, taking appropriate precautions about concurrent activity.
3. Immediately after, establish a point when the data is consistent and no unit of work is changing it.

With that preparation, recovery to the point of consistency is as quick and simple as possible. DB2 begins recovery with the copy you made and reads the log only up to the point of consistency. At that point, there are no indoubt units of recovery to hinder restarting.

Step 1: Resetting exception status

You can use the DISPLAY DATABASE RESTRICT command to determine whether the data is in an exception status. See Appendix C of *DB2 Utility Guide and Reference* for instructions on resetting those states.

For indexes that have been set to REBUILD-pending status at any time before the point of consistency, see “Recovering indexes on altered tables” on page 497.

Step 2: Copying the data

You can copy the data and also establish a point of consistency for a list of objects, in one operation, by using the COPY utility with the option SHRLEVEL REFERENCE. That operation allows only read access to the data while it is copied. The data is consistent at the moment when copying starts and remains consistent until copying ends. The advantage of the method is that the data can be restarted at a point of consistency by restoring the copy only, with no need to read log records. The disadvantage is that updates cannot be made throughout the entire time that the data is being copied.

You can use the CONCURRENT option of the COPY utility to make a backup, with DFSMSdss concurrent copy, that is recorded in the DB2 catalog. For more information about using this option, see *DB2 Utility Guide and Reference*.

Copying data while updates occur is not recommended. However, to allow updates while the data is being copied, you can:

- Use the COPY utility with the option SHRLEVEL CHANGE.
- Use an offline program to copy the data, such as DSN1COPY, DFSMShsm, or disk dump.

If you allow updates while copying, then step 3 is essential. With concurrent updates, the copy can include uncommitted changes. Those might be backed out after copying ends. Thus, the copy is not necessarily consistent data, and recovery

cannot rely on the copy only. Recovery requires reading the log up to a point of consistency, so you want to establish such a point as soon as possible.

Step 3: Establishing a point of consistency

Use the QUIESCE utility also to establish a single point of consistency (a *quiesce point*) for one or more page sets. Typically, you name all the table spaces in a table space set that you want recovered to the same point in time to avoid referential integrity violations. Or you can use the QUIESCE utility with the TABLESPACESET keyword for RI-related tables. The following statement quiesces two table spaces in database DSN8D81A:

```
QUIESCE TABLESPACE DSN8D81A.DSN8S81E
        TABLESPACE DSN8D81A.DSN8S81D
```

QUIESCE writes changed pages from the page set to disk. The catalog table SYSIBM.SYSCOPY records the current RBA and the timestamp of the quiesce point. At that point, neither page set contains any uncommitted data. A row with ICTYPE Q is inserted into SYSCOPY for each table space quiesced. Page sets DSNDB06.SYSCOPY, DSNDB01.DBD01, and DSNDB01.SYSUTILX, are an exception: their information is written to the log. Indexes are quiesced automatically when you specify WRITE(YES) on the QUIESCE statement. A SYSIBM.SYSCOPY row with ICTYPE Q is inserted for indexes that have the COPY YES attribute.

QUIESCE allows concurrency with many other utilities; however, it does not allow concurrent updates until it has quiesced all specified page sets. Depending upon the amount of activity, that can take considerable time. Try to run QUIESCE when system activity is low.

Also, consider using the MODE(QUIESCE) option of the ARCHIVE LOG command when planning for offsite recovery. It creates a system-wide point of consistency, which can minimize the number of data inconsistencies when the archive log is used with the most current image copy during recovery. See “Archiving the log” on page 430 for more information about using the MODE(QUIESCE) option of the ARCHIVE LOG command.

Preparing to recover the entire DB2 subsystem to a prior point in time

Under certain circumstances, you might want to reset the entire DB2 subsystem to a point of consistency. You can prepare a point of consistency by using the following procedure:

1. Display and resolve any indoubt units of recovery.
2. Use COPY to make image copies of all data, both user data and DB2 catalog and directory table spaces and optionally indexes. Copy SYSLGRNX and SYSCOPY last. Install job DSNTIJIC creates image copies of the DB2 catalog and directory table spaces. If you decide to copy your directory and catalog indexes, modify job DSNTIJIC to include those indexes. See Part 2 of *DB2 Installation Guide* for a description of job DSNTIJIC.

   Alternatively, you can use an offline method to copy the data. In that case, stop DB2 first; that is, do step 3 before step 2. If you do not stop DB2 before copying, you might have trouble restarting after restoring the system. If you do a volume restore, verify that the restored data is cataloged in the integrated catalog facility catalog. Use the access method services LISTCAT command to get a listing of the integrated catalog.

3. Stop DB2 with the command STOP DB2 MODE (QUIESCE). DB2 does not actually stop until all currently executing programs have completed processing. Be sure to use MODE (QUIESCE); otherwise, I/O errors can occur when the steps listed in "Performing fall back to a prior shutdown point" on page 611 are used to restart DB2.
4. When DB2 has stopped, use access method services EXPORT to copy all BSDS and active log data sets. If you have dual BSDSs or dual active log data sets, export both copies of the BSDS and the logs.
5. Save all the data that has been copied or dumped, and protect it and the archive log data sets from damage.

Preparing for disaster recovery

In the case of a total loss of a DB2 computing center, you can recover on another DB2 system at a recovery site. To do this, you must regularly back up the data sets and the log for recovery. As with all data recovery operations, the objectives of disaster recovery are to lose as little data, workload processing (updates), and time as possible.

You can provide shorter restart times after system failures by using the installation options LIMIT BACKOUT and BACKOUT DURATION. These options postpone the backout processing of long running URs during DB2 restart. See Part 2 of *DB2 Installation Guide* for the details on how to use these parameters.

Data sharing

In a data sharing environment, you can use the LIGHT(YES) parameter to quickly bring up a DB2 member to recover retained locks. Restart light is not recommended for a restart in place and is intended only for a cross-system restart for a system that does not have adequate capacity to sustain the DB2 IRLM pair. Restart light can be used for normal restart and recovery. See Chapter 5 of *DB2 Data Sharing: Planning and Administration* for more details.

For data sharing, you need to consider whether you want the DB2 group to use light mode at the recovery site. A light start might be desirable if you have configured only minimal resources at the remote site. If this is the case, you might run a subset of the members *permanently* at the remote site. The other members are restarted and then directly shutdown. The procedure for a light start at the remote site is:

1. Start the members that run *permanently* with the LIGHT(NO) option. This is the default.
2. Start other members with LIGHT(YES). The members started with LIGHT(YES) use a smaller storage footprint. After their restart processing completes, they automatically shutdown. If ARM is in use, ARM does not automatically restart the members with LIGHT(YES) again.
3. Members started with LIGHT(NO) remain active and are available to run new work.

Several levels of preparation for disaster recovery exist:

- Prepare the recovery site to recover to a fixed point in time.

  For example, you could copy everything weekly with a DFSMSdss volume dump (logical) and manually send it to the recovery site, then restore the data there.

- For recovery through the last archive, copy and send the following objects to the recovery site as you produce them:
  - Image copies of all catalog, directory, and user page sets
  - Archive logs
  - Integrated catalog facility catalog EXPORT and list
  - BSDS lists

    With this approach you can determine how often you want to make copies of essential recovery elements and send them to the recovery site.

  After you establish your copy procedure and have it operating, you must prepare to recover your data at the recovery site. See "Remote site recovery from a disaster at the local site" on page 558 for step-by-step instructions on the disaster recovery process.
- Use the log capture exit to capture log data in real time and send it to the recovery site. See "Reading log records with the log capture exit routine" on page 1134 and "Log capture routines" on page 1101.

System-wide points of consistency

In any disaster recovery scenario, system-wide points of consistency are necessary for maintaining data integrity and preventing a loss of data. A direct relationship exists between the frequency at which you make and send copies to the recovery site and the amount of data that you can potentially lose.

Figure 52 is an overview of the process of preparing to bring DB2 up at a recovery site.

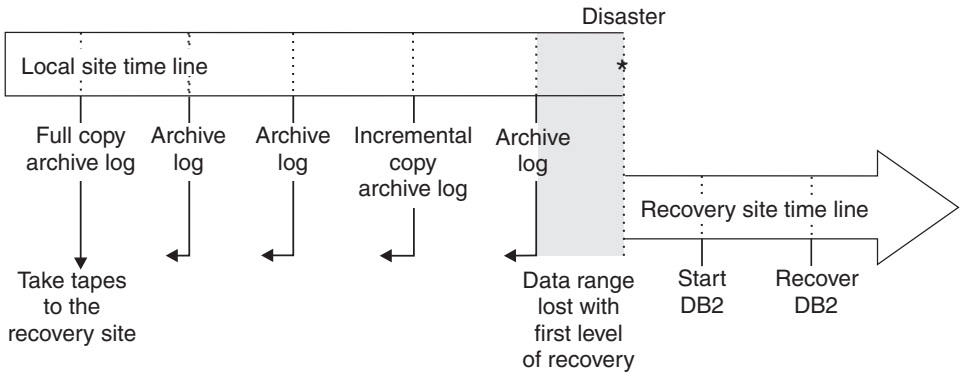


Figure 52. Preparing for disaster recovery. The information you need to recover is contained in the copies of data (including the DB2 catalog and directory) and the archive log data sets.

Essential disaster recovery elements

A list of essential disaster recovery elements and the steps you need to take to create them follows. You must determine how often to make copies and send them to the recovery site.

- Image copies
  1. Make copies of your data sets and DB2 catalogs and directories.

     Use the COPY utility to make copies for the local subsystem and additional copies for disaster recovery. You can also use the COPYTOCOPY utility to make additional image copies from the primary image copy made by the COPY utility. Install your local subsystem with the LOCALSITE option of the SITE TYPE field on installation panel DSNTIPO. Use the RECOVERYDDN

option when you run COPY to make additional copies for disaster recovery. You can use those copies on any DB2 subsystem that you have installed using the RECOVERYSITE option.[6]

For information about making multiple image copies, see COPY and COPYTOCOPY in Part 2 of *DB2 Utility Guide and Reference*. Do not produce the copies by invoking COPY twice.

2. Catalog the image copies if you want to track them.
3. Create a QMF report or use SPUFI to issue a SELECT statement to list the contents of SYSCOPY.
4. Send the image copies and report to the recovery site.
5. Record this activity at the recovery site when the image copies and the report are received.

   All table spaces should have valid image copies. Indexes can have valid image copies or they can be rebuilt from the table spaces.

- Archive logs
  1. Make copies of the archive logs for the recovery site.
     a. Use the ARCHIVE LOG command to archive all current DB2 active log data sets. For more ARCHIVE LOG command information see "Archiving the log" on page 430.

        Recommendation: When using dual logging, keep both copies of the archive log at the local site in case the first copy becomes unreadable. If the first copy is unreadable, DB2 requests the second copy. If the second copy is not available, the read fails.

        However, if you take precautions when using dual logging, such as making another copy of the first archive log, you can send the second copy to the recovery site. If recovery is necessary at the recovery site, specify YES for the READ COPY2 ARCHIVE field on installation panel DSNTIPO. Using this option causes DB2 to request the second archive log first.

     b. Catalog the archive logs if you want to track them.

        You will probably need some way to track the volume serial numbers and data set names. One way of doing this is to catalog the archive logs to create a record of the necessary information. You could also create your own tracking method and do it manually.
  2. Use the print log map utility to create a BSDS report.
  3. Send the archive copy, the BSDS report, and any additional information about the archive log to the recovery site.
  4. Record this activity at the recovery site when the archive copy and the report are received.
- Consistent system time
  1. Choose a consistent system time for all DB2 subsystems.

     DB2 utilities, including the RECOVER utility, require system clocks that are consistent across all members of a DB2 data-sharing group. To prevent inconsistencies during data recovery, ensure that all system times are consistent with the system time at the failing site.
  2. Ensure that the system clock remains consistent.

6. You can also use these copies on a subsystem installed with the LOCALSITE option if you run RECOVER with the RECOVERYSITE option. Or you can use copies prepared for the local site on a recovery site, if you run RECOVER with the option LOCALSITE.

Once you establish a consistent system time, do not alter the system clock. Any manual change in the system time (forward or backward) can affect how DB2 writes and processes image copies and log records.

- Integrated catalog facility catalog backups
  1. Back up all DB2-related integrated catalog facility catalogs with the VSAM EXPORT command on a daily basis.
  2. Synchronize the backups with the cataloging of image copies and archives.
  3. Use the VSAM LISTCAT command to create a list of the DB2 entries.
  4. Send the EXPORT backup and list to the recovery site.
  5. Record this activity at the recovery site when the EXPORT backup and list are received.
- DB2 libraries
  1. Back up DB2 libraries to tape when they are changed. Include the SMP/E, load, distribution, and target libraries, as well as the most recent user applications and DBRMs.
  2. Back up the DSNTIJUZ job that builds the ZPARM and DECP modules.
  3. Back up the data set allocations for the BSDS, logs, directory, and catalogs.
  4. Document your backups.
  5. Send backups and corresponding documentation to the recovery site.
  6. Record activity at the recovery site when the library backup and documentation are received.

For disaster recovery to be successful, all copies and reports must be updated and sent to the recovery site regularly. Data will be up to date through the last archive sent. For disaster recovery start up procedures, see "Remote site recovery from a disaster at the local site" on page 558.

Ensuring more effective recovery from inconsistency

The DB2 RECOVER utility is often the quickest and easiest method of resolving data inconsistency problems. However, these problems can involve data that the RECOVER utility needs to use, such as the recovery log or image copy data sets. If the data needed by the RECOVER utility is damaged or unavailable, you might have to resolve the problem manually.

Actions to take

To aid in successful recovery of inconsistent data:

- **During the installation of, or migration to, Version 8, make a full image copy of the DB2 directory and catalog using installation job DSNTIJIC.**

  See Part 2 of *DB2 Installation Guide* for DSNTIJIC information. If you did not do this during installation or migration, use the COPY utility, described in Part 2 of *DB2 Utility Guide and Reference*, to make a full image copy of the DB2 catalog and directory. If you do not do this and you subsequently have a problem with inconsistent data in the DB2 catalog or directory, you will not be able to use the RECOVER utility to resolve the problem.
- **Periodically make an image copy of the catalog, directory, and user databases.**

  This minimizes the time the RECOVER utility requires to perform recovery. In addition, this increases the probability that the necessary archive log data sets will still be available. You should keep *two* copies of each level of image copy data set. This reduces the risk involved if one image copy data set is lost or damaged. See Part 2 of *DB2 Utility Guide and Reference*for more information about using the COPY utility.

- **Use dual logging for your active log, archive log, and bootstrap data sets.**

  This increases the probability that you can recover from unexpected problems. It is especially useful in resolving data inconsistency problems. See "Establishing the logging environment" on page 425 for related dual logging information.

- **Before using RECOVER, rename your data sets.**

  If the image copy or log data sets are damaged, you can compound your problem by using the RECOVER utility. Therefore, before using RECOVER, rename your data sets by using one of the following methods:
  - rename the data sets that contain the page sets you want to recover, or
  - copy your data sets using DSN1COPY, or
  - for user-defined data sets, use access method services to define a new data set with the original name.

  The RECOVER utility applies log records to the new data set with the old name. Then, if a problem occurs during RECOVER utility processing, you will have a copy (under a different name) of the data set you want to recover.

- **Keep back-level image copy data sets.**

  If you make an image copy of a page set containing inconsistent data, the RECOVER utility cannot resolve the data inconsistency problem. However, you can use RECOVER TOCOPY or TOLOGPOINT to resolve the inconsistency if you have an older image copy of the page set that was taken before the problem occurred. You can also resolve the inconsistency problem by using a point in time recovery to avoid using the most recent image copy.

- **Maintain consistency between related objects.**

  A referential structure is a set of tables including indexes and their relationships. It must include at least one table, and for every table in the set, include all of the relationships in which the table participates, as well as all the tables to which it is related. To help maintain referential consistency, keep the number of table spaces in a table space set to a minimum, and avoid tables of different referential structures in the same table space. The TABLESPACESET option of the REPORT utility reports all members of a table space set defined by referential constraints.

  A referential structure must be kept consistent with respect to point in time recovery. Use the QUIESCE utility to establish a point of consistency for a table space set, to which the table space set can later be recovered without introducing referential constraint violations.

  A base table space must be kept consistent with its associated LOB table spaces with respect to point in time recovery. Use the TABLESPACESET option of the REPORT utility to find all LOB table spaces associated with a base table space. Use the QUIESCE utility to establish a point of consistency, for a table space set, to which the table space set can later be recovered.

Actions to avoid

- **Do not discard archive logs you might need.**

  The RECOVER utility might need an archive log to recover from an inconsistent data problem. If you have discarded it, you cannot use the RECOVER utility and must resolve the problem manually. For information about determining when you can discard archive logs, see "Discarding archive log records" on page 438.

- **Do not make an image copy of a page set that contains inconsistent data.**

  If you use the COPY utility to make an image copy of a page set containing inconsistent data, the RECOVER utility cannot recover a problem involving that page set unless you have an older image copy of that page set taken before the problem occurred. You can run DSN1COPY with the CHECK option to determine whether intra-page data inconsistency problems exist on page sets

before making image copies of them. If you are taking a copy of a catalog or directory page set, you can run DSN1CHKR which verifies the integrity of the links, and the CHECK DATA utility which checks the DB2 catalog (DSNDB06). For information, see *DB2 Utility Guide and Reference*.

- **Do not use the TERM UTILITY command on utility jobs you want to restart.**

  If an error occurs while a utility is running, the data on which the utility was operating might continue to be written beyond the commit point. If the utility is restarted later, processing resumes at the commit point or at the beginning of the current phase, depending on the restart parameter that was specified. If the utility stops while it has exclusive access to data, other applications cannot access that data. In this case, you might want to issue the TERM UTILITY command to terminate the utility and make the data available to other applications. However, use the TERM UTILITY command only if you cannot restart or do not need to restart the utility job.

  When you issue the TERM UTILITY command, two different situations can occur:
  – If the utility is active, it terminates at its next commit point.
  – If the utility is stopped, it terminates immediately.

  If you use the TERM UTILITY command to terminate a utility, the objects on which the utility was operating are left in an indeterminate state. Often, the same utility job cannot be rerun. The specific considerations vary for each utility, depending on the phase in process when you issue the command. For details, see Part 2 of *DB2 Utility Guide and Reference*.

Running the RECOVER utility in parallel

You can schedule jobs with the RECOVER utility in two ways:

- Use the PARALLEL keyword on the RECOVER utility to support the recovery of a list of objects in parallel. For those objects in the list that can be processed independently, multiple subtasks are created to restore the image copies for the objects. The parallel function can be used for either disk or tape.

  If an image copy on tape was taken at the tablespace level and not on partition or dataset level, the PARALLEL keyword cannot enable RECOVER utility on different parts (*RECOVER TABLESPACE name DSNUM 1 TABLESPACE name DSNUM 2 etc.*) to restore the parts in parallel. RECOVER must read the appropriate part of the dataset for every DSNUM specification. This means that for a tablespace level copy on tape, the tape is always read from the beginning. You can also refer to the PARALLEL keyword description in Utility Guide and Reference. It is recommended for image copies to tape, that a partitioned tablespace should be copied at the partition level and if practical, to different tape stacks. You can use LISTDEF with PARTLEVEL to simplify your work.

  When you use one utility statement to recover indexes and table spaces, the logs for all indexes and tables spaces are processed in one pass. This approach results in a significant performance advantage, especially when the required archive log data is on tape or the fast log apply function is enabled, or both of these conditions occur.

  This parallel log processing for fast log apply is not dependent whether you specify *RECOVER TABLESPACE name DSNUM 1 TABLESPACE name DSNUM 2 etc.* or only RECOVER TABLESPACE name, because the log apply is always done at the partition level. You should also note that if you have copies at the partition level, you cannot specify *RECOVER TABLESPACE dbname.tsname*, you must specify *RECOVER TABLESPACE dbname.tsname DSNUM 1 TABLESPACE*

dbname.tsname DSNUM 2 etc. (refer also to message DSNU512I). You can simplify this specification by using LISTDEF with PARTLEVEL if all parts must be recovered.

- Schedule concurrent RECOVER jobs that process different partitions. The degree of parallelism in this case is limited by contention for both the image copies and the required log data.

  Log data is read by concurrent jobs as follows:

  - Active logs and archive logs are read entirely in parallel.
  - A data set controlled by DFSMShsm is first recalled. It then resides on disk and is read in parallel.

Using fast log apply during RECOVER

The RECOVER utility automatically uses the fast log apply process during the LOGAPPLY phase if fast log apply has been enabled on the DB2 subsystem. For detailed information about defining storage for the sort used in fast log apply processing, see the Log Apply Storage field on panel DSNTIPL in *DB2 Installation Guide*.

Reading the log without RECOVER

The DATA CAPTURE(CHANGES) clause of the SQL statements CREATE TABLE and ALTER TABLE captures all SQL data changes made to the table on the DB2 log. The captured data can be propagated to an IMS subsystem or remain in the DB2 log. This allows the creation of duplicate data for recovery purposes. Although SQL changes to tables defined for data capture are supported from any subsystem, propagation is permitted only to an IMS subsystem. For further information see Appendix C, “Reading log records,” on page 1111.

Data written to the log for propagation to IMS uses an expanded format that is much longer than the DB2 internal format. Using DATA CAPTURE(CHANGES) can greatly increase the size of your log.

Copying page sets and data sets

You can use the COPY utility to copy data from a page set to an z/OS sequential data set on disk or tape. It makes a full or incremental image copy, as you choose, and it can be used to make copies that will be used for local or offsite recovery operations. Use the COPYTOCOPY utility to make additional image copies from a primary image copy that you made with the COPY utility.

A full image copy is required for indexes. For information about copying indexes, see “Considerations for recovering indexes” on page 473.

You can use the CONCURRENT option of the COPY utility to make a copy, with DFSMSdss concurrent copy, that is recorded in the DB2 catalog. For more information about using this option, see *DB2 Utility Guide and Reference*.

Use the MERGECOPY utility to merge several image copies. MERGECOPY does not apply to indexes.

The CHANGELIMIT option of the COPY utility causes DB2 to make an image copy automatically when a table space has changed past a default limit or a limit you specify. DB2 determines whether to make a full or incremental image copy. DB2 makes an incremental image copy if the percent of changed pages is greater than the low CHANGELIMIT value and less than the high CHANGELIMIT value.

DB2 makes a full image copy if the percent of changed pages is greater than or equal to the high CHANGELIMIT value. The CHANGELIMIT option does not apply to indexes.

If you want DB2 to recommend what image copies should be made but not make the image copies, use the CHANGELIMIT and REPORTONLY options of the COPY utility.

If you specify the parameter DSNUM ALL with CHANGELIMIT and REPORTONLY, DB2 reports information for each partition of a partitioned table space or each piece of a nonpartitioned table space.

You can add conditional code to your jobs so that an incremental or full image copy, or some other step is performed depending on how much the table space has changed. When you use the COPY utility with the CHANGELIMIT option to display image copy statistics, the COPY utility uses the following return codes to indicate the degree that a table space or list of table spaces has changed:

Code	Meaning
1	Successful and no CHANGELIMIT value is met. No image copy is recommended or taken.
2	Successful and the percent of changed pages is greater than the low CHANGELIMIT value and less than the high CHANGELIMIT value. An incremental image copy is recommended or taken.
3	Successful and the percent of changed pages is greater than or equal to the high CHANGELIMIT value. A full image copy is recommended or taken.

When you use generation data groups (GDGs) and need to make an incremental image copy, there are new steps you can take to prevent an empty image copy output data set from being created if no pages have been changed. You can perform one of the following:

1. Make a copy of your image copy step, but add the REPORTONLY and CHANGELIMIT options to the new COPY utility statement. The REPORTONLY keyword specifies that you only want image copy information displayed. Change the SYSCOPY DD card to DD DUMMY so that no output data set is allocated. Run this step to visually determine the change status of your table space.
2. Add step 1 before your existing image copy step, and add a JCL conditional statement to examine the return code and execute the image copy step if the table space changes meet either of the CHANGELIMIT values.

You can also use the COPY utility with the CHANGELIMIT option to determine whether any space map pages are broken, or to identify any other problems that might prevent an image copy from being taken, such as the object being in recover pending status. You need to correct these problems before you run the image copy job.

You can also make a full image copy when you run the LOAD or REORG utility. This technique is better than running the COPY utility after the LOAD or REORG utility because it decreases the time that your table spaces are unavailable. However, only the COPY utility makes image copies of indexes.

Related information: For guidance in using COPY and MERGECOPY and making image copies during LOAD and REORG, see Part 2 of *DB2 Utility Guide and Reference*.

Backing up with DFSMS

The concurrent copy function of Data Facility Storage Management Subsystem (DFSMS) can copy a data set concurrently with access by other processes, without significant impact on application performance. The function requires the 3990 Model 3 controller with the extended platform.

There are two ways to use the concurrent copy function of Data Facility Storage Management Subsystem (DFSMS):

- Run the COPY utility with the CONCURRENT option. DB2 records the resulting image copies in SYSIBM.SYSCOPY. To recover with these DFSMS copies, you can run the RECOVER utility to restore those image copies and apply the necessary log records to them to complete recovery.
- Make copies using DFSMS outside of DB2's control. To recover with these copies, you must manually restore the data sets, and then run RECOVER with the LOGONLY option to apply the necessary log records.

Backing up with RVA storage control or Enterprise Storage Server

IBM's RAMAC® Virtual Array (RVA) storage control with the peer-to-peer remote copy (PPRC) function or Enterprise Storage Server® provides a faster method of recovering DB2 subsystems at a remote site in the event of a disaster at the local site. You can use RVAs, PPRC, and the RVA fast copy function, SnapShot, to create entire DB2 subsystem backups to a point in time on a *hot stand-by* remote site without interruption of any application process. Another option is to use the Enterprise Storage Server FlashCopy® function to create point in time backups of entire DB2 subsystems.

To use RVA SnapShot or Enterprise Storage Server FlashCopy for a DB2 backup requires a method of suspending all update activity for a DB2 subsystem to make a remote copy of the entire subsystem without quiescing the update activity at the primary site. Use the SUSPEND option on the SET LOG command to suspend all logging activity at the primary site which also prevents any database updates.

After the remote copy has been created, use the RESUME option on the SET LOG command to return to normal logging activities. See the *DB2 Command Reference* for more details on using the SET LOG command.

For more information about RVA, see *IBM RAMAC Virtual Array*. For more information about using PPRC, see *RAMAC Virtual Array: Implementing Peer-to-Peer Remote Copy*. For more information about Enterprise Storage Server and the FlashCopy function, see *Enterprise Storage Server Introduction and Planning*.

Recovering page sets and data sets

You can recover objects in either of the following ways:

- If you made backup copies of table spaces using the COPY utility, the COPYTOCOPY utility, or the inline copy feature of the LOAD or REORG utility, use the RECOVER utility to restore the objects to the current or a previous state. Backup copies of indexes are made using the DB2 COPY utility.

- If you made backup copies using a method outside of DB2's control, such as with DSN1COPY or the DFSMSdss concurrent copy function, use the same method to restore the objects to a prior point in time. Then, if you wish to restore the objects to currency, run the RECOVER utility with the LOGONLY option.

The RECOVER utility performs these actions:
- Restores the most current full image copy
- Applies changes recorded in later incremental image copies of table spaces, if applicable, and applies later changes from the archive or active log

RECOVER can act on:
- A table space, or list of table spaces
- An index, or list of indexes
- A specific partition or data set within a table space
- A specific partition within an index space
- A mixed list of table spaces, indexes, partitions, and data sets
- A single page
- A page range within a table space that DB2 finds in error
- The catalog and directory

Typically, RECOVER restores an object to its current state by applying all image copies and log records. It can also restore to a prior state, which is one of the following points in time:
- A specified point on the log (use the TOLOGPOINT keyword)
- A particular image copy (use the TOCOPY, TOLASTCOPY, or TOLASTFULLCOPY keywords)

The RECOVER utility can use image copies for the local site or the recovery site, regardless of where you invoke the utility. The RECOVER utility locates all full and incremental image copies.

The RECOVER utility first attempts to use the primary image copy data set. If an error is encountered (allocation, open, or I/O), RECOVER attempts to use the backup image copy. If DB2 encounters an error in the backup image copy or no backup image copy exists, RECOVER falls back to an earlier full copy and attempts to apply incremental copies and log records. If an earlier full copy in not available, RECOVER attempts to apply log records only.

For guidance in using RECOVER and REBUILD INDEX, see Part 2 of *DB2 Utility Guide and Reference*.

Not every recovery operation requires RECOVER; see also
"Recovering error ranges for a work file table space" on page 493
"Recovering the work file database" on page 493
"Recovering data to a prior point of consistency" on page 495

Important: Be very careful when using disk dump and restore for recovering a data set. Disk dump and restore can make one data set inconsistent with DB2 subsystem tables in some other data set. Use disk dump and restore only to restore the entire subsystem to a previous point of consistency, and prepare that point as described in the alternative in step 2 under "Preparing to recover to a prior point of consistency" on page 481.

Recovering the work file database

You *cannot* use RECOVER with the work file database (called DSNDB07, except in a data sharing environment). That database is used for temporary space for certain SQL operations, such as join and ORDER BY. If DSNDB01.DBD01 is stopped or otherwise inaccessible when DB2 is started, then the descriptor for the work file database is not loaded into main storage and the work file database is not allocated. In order to recover from this condition after DSNDB01.DBD01 has been made available, it is necessary to stop and then start the work file database again.

Problems with the user-defined work file data set

If you have a problem on a volume of a user-defined data set for the work file database, then:

1. Issue the following DB2 command:

   ```
   -STOP DATABASE (DSNDB07)
   ```

2. Use the DELETE and DEFINE functions of access method services to redefine a user work file on a different volume and reconnect it to DB2.
3. Issue the following DB2 command:

   ```
   -START DATABASE (DSNDB07)
   ```

Problems with DB2-managed work file data sets

If you have a problem on a volume in a DB2 storage group for the work file database, such as a system I/O problem, then:

1. Enter the following SQL statement to remove the problem volume from the DB2 storage group:

   ```
   ALTER STOGROUP stogroup-name
    REMOVE VOLUMES (xxxxxx);
   ```

2. Issue the following DB2 command:

   ```
   -STOP DATABASE (DSNDB07)
   ```

3. Enter the following SQL statement to drop the table space with the problem:

   ```
   DROP TABLESPACE DSNDB07.tsname:
   ```

4. Re-create the table space. You can use the same storage group, because the problem volume has been removed, or you can use an alternate.

   ```
   CREATE TABLESPACE tsname
     IN DSNDB07
     USING STOGROUP stogroup-name;
   ```

5. Issue the following command:

   ```
   -START DATABASE (DSNDB07)
   ```

Recovering error ranges for a work file table space

Page error ranges operate for work file table spaces in the same way as for other DB2 table spaces, except for the process of recovering them. Error ranges in a work file table space cannot be reset by RECOVER ERROR RANGE. Instead, perform the following procedure:

1. Stop the work file table space.
2. Correct the disk error, using the ICKDSF service utility or access method services to delete and redefine the data set.
3. Start the work file table space. When the work file table space is started, DB2 automatically resets the error range.

Also, DB2 always resets any error ranges when the work file table space is initialized, regardless of whether the disk error has really been corrected. Work file table spaces are initialized when:

- The work file table space is stopped and then started
- The work file database is stopped and then started, and the work file table space was not previously stopped
- DB2 is started and the work file table space was not previously stopped

If the error range is reset while the disk error still exists, and if DB2 has an I/O error when using the work file table space again, then DB2 sets the error range again.

Recovering the catalog and directory

Catalog and directory objects must be recovered in a particular order. Because the recovery of some catalog and directory objects depends on information that is derived from other catalog and directory objects, you must recover these objects in separate RECOVER utility control statements. You can, however, use the same RECOVER control statement to recover a catalog or directory table space along with its corresponding IBM-defined indexes. After these logically dependent objects are restored to an undamaged state, you can recover the remaining catalog and directory objects in a single RECOVER utility control statement. These restrictions apply regardless of the type of recovery you perform on the catalog. See the description of RECOVER in Part 2 of *DB2 Utility Guide and Reference* for more information about the RECOVER utility and for details about the order in which you must recover objects in the catalog and directory.

You can use the REPORT utility to report on recovery information about the catalog and directory.

To avoid restart processing of any page sets before attempts are made to recover any of the members of the list of catalog and directory objects, use the DEFER option when installing DB2 followed by the option ALL. For more information about DEFER, see “Deferring restart processing” on page 450.

Point in time recovery: Recovering the DB2 catalog and directory to a prior point in time is strongly discouraged. For more information about recovering the catalog and directory to a prior point in time see “Considerations for recovering to a prior point of consistency” on page 495.

Recovery after a conditional restart of DB2: After a DB2 conditional restart in which a log record range is specified, such as with a cold start, a portion of the DB2 recovery log is no longer available. If the unavailable portion includes information that is needed for internal DB2 processing, an attempt to use the RECOVER utility to restore directory table spaces DSNDBD01 or SYSUTILX, or catalog table space SYSCOPY, will fail with abend 00E40119. Instead of using the RECOVER utility, use this procedure to recover those table spaces and their indexes:

1. Run DSN1COPY to recover the table spaces from an image copy.
2. Run the RECOVER utility with the LOGONLY option to apply updates from the log records to the recovered table spaces.
3. Rebuild the indexes.
4. Make a full image copy of the table spaces and optionally the indexes to establish a new recovery point.

Recovering data to a prior point of consistency

You can restore data to a state it was in at a prior point in time if you back up that data appropriately. To restore data to a prior point of consistency, use the methods described in the following sections:

- "Using RECOVER to restore data to a previous point in time" on page 500.
- "Restoring data by using DSN1COPY" on page 504
- "Backing up and restoring data with non-DB2 dump and restore" on page 504

You cannot recover to some points in time. See Part 2 of *DB2 Utility Guide and Reference* for more information about these restrictions.

The following terms are used throughout this discussion:

Term	Meaning
DBID	Database identifier
OBID	Data object identifier
PSID	Table space identifier

Considerations for recovering to a prior point of consistency

When you restore data in any of the following DB2 objects to a prior point of consistency, be aware of the considerations this section describes:

- Table spaces
- Tables with identity columns
- Indexes
- Catalog and directory tables

Recovering table spaces

When you recover table spaces to a prior point of consistency, you need to consider how partitioned table spaces, segmented table spaces, LOB tables spaces, and table space sets can restrict recovery.

Recovering partitioned table spaces: You cannot recover a table space to a point in time prior to rotating partitions. After you rotate a partition, you cannot recover the contents of that partition.

If you recover to a point in time prior to the addition of a partition, DB2 cannot roll back the definition of the partition. In such a recovery, DB2 clears all data from the partition, and the partition remains part of the database.

If you recover a table space partition to a point in time before the table space partitions were rebalanced, you must include all partitions that are affected by that rebalance in your recovery list.

Recovering segmented table spaces: When data is restored to a prior point in time on a segmented table space, information in the DBD for the table space might not match the restored table space.

If you use the DB2 RECOVER utility, the database descriptor (DBD) is updated dynamically to match the restored table space on the next non-index access of the table. The table space must be in write access mode.

If you use a method outside of DB2's control, such as DSN1COPY to restore a table space to a prior point in time, run the REPAIR utility with the LEVELID option to force DB2 to accept the down-level data. Then run the REORG utility on the table space to correct the DBD.

Recovering LOB table spaces: When you recover tables with LOB columns, recover the entire set of objects, including the base table space, the LOB table spaces, and index spaces for the auxiliary indexes.

If you use the RECOVER utility to recover a LOB table space to a prior point of consistency, RECOVER might place the table space in a pending state. For more details about the particular pending states that the RECOVER utility sets, see "Using RECOVER to restore data to a previous point in time" on page 500.

Recovering table space sets: If you restore a page set to a prior state, restore all related tables and indexes to the same point to avoid inconsistencies. The table spaces that contain referentially related tables are called a *table space set*. Similarly, a LOB table space and its associated base table space are also part of a table space set. For example, in the DB2 sample application, a column in the EMPLOYEE table identifies the department to which each employee belongs. The departments are described by records in the DEPARTMENT table, which is in a different table space. If only that table space is restored to a prior state, a row in the unrestored EMPLOYEE table might then identify a department that does not exist in the restored DEPARTMENT table.

You can use the REPORT TABLESPACESET utility to determine all the page sets that belong to a single table space set and then restore those page sets that are related. However, if page sets are logically related outside of DB2 in application programs, you are responsible for identifying all the page sets on your own.

To determine a valid quiesce point for the table space set, use the procedure for determining a RECOVER TOLOGPOINT value. See RECOVER in Part 2 of *DB2 Utility Guide and Reference* for more information.

Recovering tables that contain identity columns

When you recover a table that contains an identity column, consider the point in time to which you recover. Your recovery procedure can depend on whether that identity column existed, or was not yet defined at the point in time to which you recover. The following considerations apply for each of these two cases:

- If you recover to a point in time at which the identity column existed, you might create a gap in the sequence of identity column values. When you insert a row after this recovery, DB2 produces an identity value for the row as if all previously added rows still exist.

  Example: Assume that at time T1 an identity column that increments by 1 contains the identity column values 1 through 100. At T2, the same identity column contains the values 1 through 1000. Now, assume that the table space is recovered back to time T1. When you insert a row after the recovery, DB2 generates an identity value of 1001. This value leaves a gap from 101 to 1000 in the values of the identity column.

  To prevent a gap in identity column values, use the following ALTER TABLE statement to modify the attributes of the identity column before you insert rows after the recovery:

  ```
  ALTER TABLE table-name
  ALTER COLUMN identity-column-name
  RESTART WITH next-identity-value
  ```

Tip: To determine the last value in an identity column, issue the MAX column function for ascending sequences of identity column values, or the MIN column function for descending sequences of identity column values. This method works only if the identity column does not use CYCLE.

- If you recover to a point in time at which the identity column was not yet defined, that identity column remains part of the table. The resulting identity column no longer contains values.

  To regenerate missing identity column values, perform the following steps:

  1. Choose a starting value for the identity column with the following ALTER TABLE statement:

     ```
     ALTER TABLE table-name
     ALTER COLUMN identity-column-name
     RESTART WITH starting-identity-value
     ```

  2. Run the REORG utility to regenerate lost sequence values.

  If you do not choose a starting value, the REORG utility generates a sequence of identity column values that starts with the next value that DB2 would have assigned before the recovery.

  A table space that contains an identity column is set to REORG-pending (REORP) status if you recover the table space to a point in time before the identity column was defined. To access the recovered table, you need to remove this status. See "Ensuring consistency" on page 502 for information about how to remove REORP status from a table space.

Recovering indexes

When you recover indexes to a prior point of consistency, the following general rules apply:

- If an image copies exists for an indexes, use the RECOVER utility.
- If indexes do not have image copies, you must use REBUILD INDEX to re-create the indexes after the data has been recovered.

More specifically, you must consider how indexes on altered tables and indexes on tables in partitioned table spaces can restrict recovery.

Recovering indexes on altered tables: You cannot use the RECOVER utility to recover an index to a point in time that existed before you issued any of the following ALTER statements on that index. These statements place the index in REBUILD-pending (RBDP) status:

- ALTER INDEX PADDED
- ALTER INDEX NOT PADDED
- ALTER TABLE SET DATA TYPE on an indexed column for numeric data type changes
- ALTER TABLE ADD COLUMN and ALTER INDEX ADD COLUMN that are not issued in the same commit scope

When you recover a table space to prior point in time and the table space uses indexes that were set to RBDP at any time after the recovery point, you must use the REBUILD INDEX utility to rebuild these indexes.

For more information about the RECOVER and REBUILD INDEX utilities, see Part 2 of *DB2 Utility Guide and Reference*.

The output from the DISPLAY DATABASE RESTRICT command shows the restrictive states for index spaces. See *DB2 Command Reference* for descriptions of status codes displayed by the DISPLAY DATABASE command.

Recovering indexes on tables in partitioned table spaces: The partitioning of secondary indexes allows you to copy and recover at the entire index or individual partition level. However, if you use COPY at the partition level, you need to use RECOVER at the partition level also. If you use COPY at the partition level and then try to RECOVER the index, an error occurs. If COPY is performed at the index level, you can use RECOVER at either the index level or the partition level.

You cannot recover an index space to a point in time prior to rotating partitions. After you rotate a partition, you cannot recover the contents of that partition to a point in time before the rotation.

If you recover to a point in time prior to the addition of a partition, DB2 cannot roll back the addition of that partition. In such a recovery, DB2 clears all data from the partition, and it remains part of the database.

Recovering catalog and directory tables

Recovering catalog and directory tables to a prior point in time is strongly discouraged for the following reasons:

- You must recover all table spaces that are associated with the catalog tables that you recover to the same point in time. For example, if you recover any table space in the DB2 catalog (DSNDB06) and directory (DSNDB01), all table spaces (except SYSUTILX) must be recovered.

  The catalog and directory contain definitions of all databases. When databases DSNDB01 and DSNDB06 are restored to a prior point, information about later definitions, authorizations, binds, and recoveries is lost. If you restore the catalog and directory, you might have to restore user databases; if you restore user databases, you might have to restore the catalog and directory.
- You might create catalog definitions that are inconsistent with your data. These catalog and data inconsistencies are usually the result of one of the following actions:
  - A catalog table space was restored.
  - SYSSEQ and SYSSEQ2 were recovered to a prior point in time.
  - The definition of a table, table space, index, or index space was changed after the data was last backed up.
- You can cause problems for user table spaces or index spaces that have been reorganized with FASTSWITCH.
- You can cause a populated VSAM data set that was defined with DEFINE NO option to revert back to the undefined state. To avoid errors, you must delete the existing VSAM data sets before the table space or index can be accessed.

Avoiding point-in-time recovery of the catalog is easier than attempting to correct the inconsistencies that the recovery causes. If you choose to recover catalog and directory tables to a prior point in time, you need to first shut down the DB2 system cleanly and then restart in ACCESS(MAINT) mode before the recovery. If you recover catalog tables to a prior point in time, you must perform the following actions to make catalog definitions consistent with your data.

1. Run the DSN1PRNT utility with the PARM=(FORMAT, NODATA) option on all data sets that might contain user table spaces. The NODATA option suppresses all row data, which reduces the output volume that you receive. Data sets that contain user tables are of the following form, where *y* can be either I or J:

   `catname.DSNDBC.dbname.tsname.y0001.A00n`

2. Execute the following SELECT statements to find a list of table space and table definitions in the DB2 catalog:

Product-sensitive Programming Interface

```
SELECT NAME, DBID, PSID FROM SYSIBM.SYSTABLESPACE;
SELECT NAME, TSNAME, DBID, OBID FROM SYSIBM.SYSTABLES;
```

End of Product-sensitive Programming Interface

3. For each table space name in the catalog, look for a data set with a corresponding name. If a data set exists, take the following additional actions:
   a. Find the field HPGOBID in the header page section of the DSN1PRNT output. This field contains the DBID and PSID for the table space. See if the corresponding table space name in the DB2 catalog has the same DBID and PSID.
   b. If the DBID and PSID do not match, execute DROP TABLESPACE and CREATE TABLESPACE to replace the incorrect table space entry in the DB2 catalog with a new entry. Be sure to make the new table space definition exactly like the old one. If the table space is segmented, SEGSIZE must be identical for the old and new definitions.

      A LOB table space can be dropped only if it is empty (that is, it does not contain auxiliary tables). If a LOB table space is not empty, you must first drop the auxiliary table before you drop the LOB table space. To drop auxiliary tables, you can perform one of the following actions:
      - Drop the base table.
      - Delete all rows that reference LOBs from the base table, and then drop the auxiliary table.
   c. Find the PGSOBD fields in the data page sections of the DSN1PRNT output. These fields contain the OBIDs for the tables in the table space. For each OBID that you find in the DSN1PRNT output, search the DB2 catalog for a table definition with the same OBID.
   d. If any of the OBIDs in the table space do not have matching table definitions, examine the DSN1PRNT output to determine the structure of the tables that are associated with these OBIDs. If you find a table whose structure matches a definition in the catalog, but the OBIDs differ, proceed to the next step. The OBIDXLAT option of DSN1COPY corrects the mismatch. If you find a table for which no table definition exists in the catalog, re-create the table definition using the CREATE TABLE statement. To re-create a table definition for a table that has had columns added, first use the **original** CREATE TABLE statement, and then use ALTER TABLE to add columns, which makes the table definition match the current structure of the table.
   e. Use the utility DSN1COPY with the OBIDXLAT option to copy the existing data to the new tables in the table space, and translate the DBID, PSID, and OBIDs.

   If a table space name in the DB2 catalog does not have a data set with a corresponding name, one of the following events has probably occurred:
   - The table space was dropped after the point in time to which you recovered. In this case, you cannot recover the table space. Execute DROP TABLESPACE to delete the entry from the DB2 catalog.
   - The table space was defined with the DEFINE(NO) option. In this case, the data set will be allocated when you insert data into the table space.

4. For each data set in the DSN1PRNT output, look for a corresponding DB2 catalog entry. If no entry exists, follow the instructions in "Recovery of an accidentally dropped table space" on page 507 to re-create the entry in the DB2 catalog.
5. If you recover the catalog tables SYSSEQ and SYSSEQ2, identity columns and sequence objects are inconsistent. To avoid duplicate identity column values, recover all table spaces that contain tables that use identity columns to the point in time to which you recovered SYSSEQ and SYSSEQ2. To eliminate gaps between identity column values, use the ALTER TABLE statement. For sequence objects, use the ALTER SEQUENCE statement to eliminate these gaps. See "Recovering tables that contain identity columns" on page 496 for more information.
6. Ensure that the IPREFIX values of user table spaces and index spaces that were reorganized with the FASTSWITCH option match the IPREFIX value in the VSAM data set names that are associated with each table space or partition. If the IPREFIX recorded in the DB2 catalog and directory is different from the VSAM cluster names, you cannot access your data. To ensure that these IPREFIX values match, complete the following procedure:
   a. Query the SYSIBM.SYSTABLEPART and SYSIBM.SYSINDEXPART catalog tables to determine the IPREFIX value that is recorded in the catalog for objects that were reorganized with the FASTSWITCH option.
   b. Compare this IPREFIX value to the IPREFIX value in the VSAM data set name that is associated with the table space or index space.
   c. When IPREFIX values do not match for an object, rename the VSAM data set to specify the correct IPREFIX.

      Example: If the catalog specifies an IPREFIX of **J** for an object but the VSAM data set that corresponds to this object is `catname.DSNDBC.dbname.spname.I0001.A001`, you must rename this data set to `catname.DSNDBC.dbname.spname.J0001.A001`.
7. Delete the VSAM data sets that are associated with table spaces that were created with the DEFINE NO option and that reverted to an unallocated state. After you delete the VSAM data sets, you can insert or load rows into these unallocated table spaces to allocate new VSAM data sets. For more information about the DEFINE NO option of the CREATE TABLESPACE and CREATE INDEX SQL statements, see*DB2 SQL Reference*.

See Part 3 of *DB2 Utility Guide and Reference* for more information about DSN1COPY and DSN1PRNT.

Using RECOVER to restore data to a previous point in time

TOCOPY, TOLOGPOINT, TOLASTCOPY, and TOLASTFULLCOPY are options of the RECOVER utility. All of these options terminate recovery at a specified point. Because they recover data to a prior time, and not to the present, they are referred to as point in time recoveries.

Using the TOCOPY, TOLASTCOPY, and TOLASTFULLCOPY options

The TOCOPY, TOLASTCOPY, and TOLASTFULLCOPY options identify an image copy as the point in time to which to recover. With these options, the RECOVER utility restores objects to the value of a specified image copy and does not apply subsequent changes from the log. If the image copy that is specified in one of these options cannot be applied, the RECOVER utility uses an earlier full image copy and applies the changes in the log up to the point in time at which the specified image copy was taken.

If the image copy data set is cataloged when the image copy is made, the entry for that copy in SYSIBM.SYSCOPY does not record the volume serial numbers of the data set. You can identify that copy by its name by using TOCOPY *data set name*. If the image copy data set was not cataloged when it was created, you can identify the copy by its volume serial identifier by using TOVOLUME *volser*.

Using the TOLOGPOINT option

With TOLOGPOINT, the RECOVER utility restores the most recent full image copy and the most recent set of incremental copies that occur before the specified log point. The log and logged changes are applied up to, and including, the record that contains the log point. If no full image copy exists before the chosen log point, recovery is attempted entirely from the log. The log is applied from the log point at which the page set was created or the last LOAD or REORG TABLESPACE utility was run to a log point that you specify. You can apply the log only if you have **not** used the MODIFY RECOVERY utility to delete the SYSIBM.SYSLGRNX records for the log range your recovery requires.

You can use the TOLOGPOINT option in both data sharing and non-data-sharing environments. In a non-data-sharing environment, TOLOGPOINT and TORBA are interchangeable keywords that identify an RBA on the log at which recovery stops. TORBA can be used in a data sharing environment only if the TORBA value is before the point at which data sharing was enabled.

Recommendation: Use the TOLOGPOINT keyword instead of the TORBA keyword. Although DB2 still supports the TORBA option, the TOLOGPOINT option supports both data sharing and non-data-sharing environments and is used for both of these environments throughout this publication.

Planning for point in time recovery

TOCOPY and TOLOGPOINT are viable alternatives in many situations in which recovery to the current point in time is not possible or is not desirable. To make these options work best, take periodic quiesce points at points of consistency that are appropriate to your applications.

When making copies of a single object, use SHRLEVEL REFERENCE to establish consistent points for TOCOPY recovery. Copies that are made with SHRLEVEL CHANGE do not copy data at a single instant, because changes can occur as the copy is made. A subsequent RECOVER TOCOPY operation can produce inconsistent data.

When copying a list of objects, use SHRLEVEL REFERENCE. If a subsequent recovery to a point in time is necessary, you can use a single RECOVER utility statement to list all of the objects, along with TOLOGPOINT to identify the common RBA or LRSN value. If you use SHRLEVEL CHANGE to copy a list of objects, you should follow it with a QUIESCE of the objects.

An inline copy that is made during LOAD REPLACE can produce unpredictable results if that copy is used later in a RECOVER TOCOPY operation. DB2 makes the copy during the RELOAD phase of the LOAD operation. Therefore, the copy does not contain corrections for unique index violations and referential constraint violations because those corrections occur during the INDEXVAL and ENFORCE phases.

To improve the performance of the recovery, take a full image copy of the page sets, and then quiesce them using the QUIESCE utility. This allows RECOVER TOLOGPOINT to recover the page sets to the quiesce point with minimal use of the log.

If you are working with partitioned table spaces, image copies that were taken prior to resetting the REORG-pending status of any partition of a partitioned table space cannot be used for recovery to a current point in time. Avoid performing a point in time recovery for a partitioned table space to a point in time that is after the REORG-pending status was set, but before a rebalancing REORG was performed. See information about RECOVER in Part 2 of *DB2 Utility Guide and Reference* for details on determining an appropriate point in time and creating a new recovery point.

If you use the REORG TABLESPACE utility with the FASTSWITCH YES option on only some partitions of a table space, you must recover that table space at the partition level. When you take an image copy of such a table space, the COPY utility issues the informational message `DSNU429I`. For a complete description of `DSNU429I`, see *DB2 Messages*.

Authorization: Restrict the use of the TOCOPY, TOLOGPOINT, TOLASTCOPY, and TOLASTFULLCOPY options of the RECOVER utility to personnel with a thorough knowledge of the DB2 recovery environment.

Ensuring consistency

RECOVER TOLOGPOINT and RECOVER TOCOPY can be used on a single:

- Partition of a partitioned table space
- Partition of a partitioning index space
- Data set of a simple table space

All page sets must be restored to the same level; otherwise the data is inconsistent.

A table space and all of its indexes (or a table space set and all related indexes) should be recovered in a single RECOVER utility statement that specifies TOLOGPOINT. The log point should identify a quiesce point or a common SHRLEVEL REFERENCE copy point. This action avoids placing indexes in the CHECK-pending or RECOVER-pending status. If the log point is not a common quiesce point or SHRLEVEL REFERENCE copy point for all objects, use the following procedure:

1. RECOVER table spaces to the log point.
2. Use concurrent REBUILD INDEX jobs to rebuild the indexes for each table space.

This procedure ensures that the table spaces and indexes are synchronized and eliminates the need to run the CHECK INDEX utility.

Point in time recovery can cause table spaces to be placed in CHECK-pending status if they have table check constraints or referential constraints defined on them. When recovering tables that are involved in a referential constraint, you should recover all the table spaces that are involved in a constraint. This is the *table space set*. To avoid setting CHECK-pending status, you must perform both of the following tasks:

- Recover the table space set to a quiesce point.

  If you do not recover each table space of the table space set to the same quiesce point, and if any of the table spaces are part of a referential integrity structure:

  - All dependent table spaces that are recovered are placed in CHECK-pending status with the scope of the whole table space.
  - All table spaces that are dependent on the table spaces that are recovered are placed in CHECK-pending status with the scope of the specific dependent tables.
- Establish a quiesce point or take an image copy after you add check constraints or referential constraints to a table.

  If you recover each table space of a table space set to the same quiesce point, but referential constraints were defined after the quiesce point, the CHECK-pending status is set for the table space containing the table with the referential constraint.

For information about resetting the CHECK-pending status, see "Violation of referential constraint recovery" on page 552.

The RECOVER utility sets various states on table spaces. The following point in time recoveries set such states on table spaces:

- When the RECOVER utility finds an invalid column during the LOGAPPLY phase on a LOB table space, it sets the table space to auxiliary-warning (AUXW) status.
- When you recover a LOB table space to a point in time that is not a quiesce point or to an image copy that is produced with SHRLEVEL CHANGE, the LOB table space is placed in CHECK-pending (CHKP) status.
- When you recover only the LOB table space to any previous point in time, the base table space is placed in auxiliary CHECK-pending (ACHKP) status, and the index space containing an index on the auxiliary table is placed in REBUILD-pending (RBDP) status.
- When you recover only the base table space to a point in time, the base table space is placed in CHECK-pending (CHKP) status.
- When you recover only the index space containing an index on the auxiliary table to a point in time, the index space is placed in CHECK-pending (CHKP) status.
- When you recover partitioned table spaces with the RECOVER utility to a point in time that is prior to a partition rebalance, all partitions that were rebalanced are placed in REORG-pending (REORP) status.
- When you recover a table space to point in time prior to when an identity column was defined with the RECOVER utility, the table space is placed in REORG-pending (REORP) status.
- If you do not recover all objects that are members of a referential set to a prior point in time with the RECOVER utility, or if you recover a referential set to a point in time that is not a point of consistency with the RECOVER utility, all dependent table spaces are placed in CHECK-pending (CHKP) status.

See Part 2 of *DB2 Utility Guide and Reference* for detailed information about recovering a table space that contains LOB data.

To remove various pending states from a table space, run the following utilities in this order:

1. Use the REORG TABLESPACE utility to remove the REORP status.
2. If the table space status is auxiliary CHECK-pending status:
   - Use CHECK LOB for all associated LOB table spaces.
   - Use CHECK INDEX for all indexes on the LOB table spaces.
3. Use the CHECK DATA utility to remove the CHECK-pending status.

Compressed data: Use caution when recovering a single data set of a non-partitioned page set to a prior point in time. If the data set that is recovered was compressed with a different dictionary from the rest of the page set, you can no longer read the data. For important information about loading and compressing data, see the description of LOAD in Part 2 of *DB2 Utility Guide and Reference*.

The RECOVER utility does not reset the values that DB2 generates for identity columns.

Important: The RECOVER utility does not back out CREATE or ALTER statements. After a recovery to a previous point in time, all previous alterations to identity column attributes remain unchanged. Because these alterations are not backed out, a recovery to a point in time might put identity column tables out of sync with the SYSIBM.SYSSEQUENCES catalog table. You might need to modify identity column attributes after a recovery to resynchronize identity columns with the catalog table.

Restoring data by using DSN1COPY

You can use DSN1COPY to restore data that was previously backed up by DSN1COPY or by COPY. If you use DSN1COPY to restore data or move data, the data definitions for the target object must be exactly the same as when the copy was created. You cannot use DSN1COPY to restore data that was backed up with the DFSMSdss concurrent copy facility.

Be careful when creating backups with DSN1COPY. You must ensure that the data is consistent, or you might produce faulty backup copies. One advantage of using COPY to create backups is that it does not allow you to copy data that is in CHECK-pending or RECOVER-pending status. You can use COPY to prepare an up-to-date image copy of a table space, either by making a full image copy or by making an incremental image copy and merging that incremental copy with the most recent full image copy.

Keep access method services LISTCAT listings of table space data sets that correspond to each level of retained backup data.

For more information about using DSN1COPY, see Part 3 of *DB2 Utility Guide and Reference*.

Backing up and restoring data with non-DB2 dump and restore

You can use certain non-DB2 facilities to dump and restore data sets and volumes. But note carefully the limitations described in this section.

Even though DB2 data sets are defined as VSAM data sets, DB2 data cannot be read or written by VSAM record processing because it has a different internal format. The data can be accessed by VSAM control interval (CI) processing. If you manage your own data sets, you can define them as VSAM linear data sets (LDSs), and access them through services that support data sets of that type.

Access method services for CI and LDS processing are available in z/OS. IMPORT and EXPORT use CI processing; PRINT and REPRO do not, but they do support LDSs.

DFSMS Data Set Services (DFSMSdss) is available on z/OS and provides dump and restore services that can be used on DB2 data sets. Those services use VSAM CI processing.

Recovery of dropped objects

The procedures described in this section can be used if a table or table space is inadvertently dropped.

Avoiding accidentally dropping objects

To avoid the problem of accidentally dropping tables, you can create a table with the clause WITH RESTRICT ON DROP. No one can drop the table, nor the table space or database containing the table, until the restriction on the table is removed. The ALTER TABLE statement includes a clause to remove the restriction, as well as one to impose it.

Procedures for recovery

The following terms are used throughout this discussion:

Term	Meaning
DBID	Database identifier
OBID	Data object identifier
PSID	Table space identifier

Recommendation: To prepare for this procedure, **run regular catalog reports** that include a list of all OBIDs in the subsystem. In addition create catalog reports that list dependencies on the table (such as referential constraints, indexes, and so on). After a table is dropped, this information disappears from the catalog.

If an OBID has been reused by DB2, you must run DSN1COPY to translate the OBIDs of the objects in the data set. However, this event is unlikely; DB2 reuses OBIDs only when no image copies exist that contain data from that table.

Important: When you recover a dropped object, you essentially recover a table space to a point in time. If you want to use log records to perform forward recovery on the table space, you need the IBM DB2 UDB Log Analysis Tool for z/OS.

Recovery of an accidentally dropped table

Tables in a partitioned table space cannot be dropped without dropping the table space. If you accidentally drop a table space, see “Recovery of an accidentally dropped table space” on page 507.

To perform this procedure, you need a full image copy or a DSN1COPY file that contains the data from the dropped table.

For segmented table spaces, the image copy or DSN1COPY file must contain the table when it was active (that is, created). Because of the way space is reused for segmented table spaces, this procedure cannot be used if the table was not active when the image copy or DSN1COPY was made. For nonsegmented table spaces, the image copy or DSN1COPY file can contain the table when it was active or not active.

The following procedure recovers a dropped table:

1. If you know the DBID, the PSID, the original OBID of the dropped table, and the OBIDs of all other tables in the table space, go to step 2.

   If you do not know all of the preceding items, use the following steps to find them. For later use with DSN1COPY, record the DBID, the PSID, and the OBIDs of all the tables in the table space, not just the dropped table.

a. For the data set that contains the dropped table, run DSN1PRNT with the FORMAT and NODATA options. Record the HPGOBID field in the header page and the PGSOBD field from the data records in the data pages.

   For the auxiliary table of a LOB table space, record the HPGROID field in the header page instead of PGSOBD field in the data pages.

   - Field HPGOBID is 4 bytes long and contains the DBID in the first 2 bytes and the PSID in the last 2 bytes.
   - Field HPGROID (for LOB table spaces) contains the OBID of the table. A LOB table space can contain only one table.
   - Field PGSOBD (for non-LOB table spaces) is 2 bytes long and contains the OBID of the table. If your table space contains more than one table, check for all OBIDs. In other words, search for all different PGSOBD fields. You need to specify all OBIDs from the data set as input for the DSN1COPY utility.

b. Convert the hex values in the identifier fields to decimal so that they can be used as input for the DSN1COPY utility.

2. Use the SQL CREATE statement to re-create the table and any indexes on the table.
3. To allow DSN1COPY to access the DB2 data set, stop the table space using the following command:

   ```
   -STOP DATABASE(database-name) SPACENAM(tablespace-name)
   ```

   Stopping the table space is necessary to ensure that all changes are written out and that no data updates occur during this procedure.
4. Find the OBID for the table you created in step 2 by querying the SYSIBM.SYSTABLES catalog table. The following statement returns the object ID (OBID) for the table:

   Product-sensitive Programming Interface

   ```
   SELECT NAME, OBID FROM SYSIBM.SYSTABLES
     WHERE NAME='table_name'
       AND CREATOR='creator_name';
   ```

   End of Product-sensitive Programming Interface

   This value is returned in decimal format, which is the format you need for DSN1COPY.
5. Run DSN1COPY with the OBIDXLAT and RESET options to perform the OBID translation and to copy data from the dropped table into the original data set. You must specify a previous full image copy data set, inline copy data set, or DSN1COPY file as the input data set SYSUT1 in the control statement. Specify each of the input records in the following order in the SYSXLAT file to perform OBID translations:

   a. The DBID that you recorded in step 1 as both the translation source and the translation target
   b. The PSID that you recorded in step 1 as both the translation source and the translation target
   c. The original OBID that you recorded in step 1 for the dropped table as the translation source and the OBID that you recorded in step 4 as the translation target
   d. OBIDs of all other tables in the table space that you recorded in step 2 as both the translation sources and translation targets

Be sure that you have named the VSAM data sets correctly by checking messages DSN1998I and DSN1997I after DSN1COPY completes.

For more information about DSN1COPY, see Part 3 of *DB2 Utility Guide and Reference*.

6. Use DSN1COPY with the OBIDXLAT and RESET options to apply any incremental image copies. You must apply these incremental copies in sequence, and specify the same SYSXLAT records that step 5 specifies.

   Important: After you complete this step, you have essentially recovered the table space to the point in time of the last image copy. If you want to use log records to perform forward recovery on the table space, you must use the IBM DB2 UDB Log Analysis Tool for z/OS at this point in the recovery procedure.

   For more information about point in time recovery, see "Recovering data to a prior point of consistency" on page 495.
7. Start the table space for normal use by using the following command:

   ```
   -START DATABASE(database-name) SPACENAM(tablespace-name)
   ```
8. Rebuild all indexes on the table space.
9. Execute SELECT statements on the previously dropped table to verify that you can access the table. Include all LOB columns in these queries.
10. Make a full image copy of the table space. See "Copying page sets and data sets" on page 489 for more information about the COPY utility.
11. Re-create the objects that are dependent on the recovered table.

    As explained in "Implications of dropping a table" on page 89, when a table is dropped, objects that are dependent on that table (synonyms, views, indexes, referential constraints, and so on) are dropped. (Aliases are not dropped.) Privileges that are granted for that table are also dropped. Catalog reports or a copy of the catalog taken prior to the DROP TABLE can make this task easier.

Recovery of an accidentally dropped table space

These procedures are for table spaces, including LOB table spaces, that were dropped accidentally. You might accidently drop a table space, for example, when all tables in an implicitly created table space are dropped, or if someone unintentionally executes a DROP TABLESPACE statement for a particular table space.

When a table space is dropped, DB2 loses all information about the image copies of that table space. Although the image copy data set is not lost, locating it might require examination of image copy job listings or manually recorded information about the image copies.

This section provides two separate procedures: one for user-managed data sets and one for DB2-managed data sets.

User-managed data sets: The following procedure recovers dropped table spaces that are not part of the catalog. In this procedure, you copy the data sets containing data from the dropped table space to redefined data sets using the OBID-translate function of DSN1COPY.

1. Find the DBID for the database, the PSID for the dropped table space, and the OBIDs for the tables contained in the dropped table space. For information about how to do this, see step 1 of "Recovery of an accidentally dropped table" on page 505.

2. Rename the data set containing the dropped table space by using the IDCAMS ALTER command. Rename both the CLUSTER and DATA portion of the data set with a name that begins with the integrated catalog facility catalog name or alias.
3. Redefine the original DB2 VSAM data sets.

   Use the access method services LISTCAT command to obtain a list of data set attributes. The data set attributes on the redefined data sets must be the same as they were on the original data sets.
4. Use SQL CREATE statements to re-create the table space, tables, and any indexes on the tables.
5. To allow DSN1COPY to access the DB2 data sets, stop the table space using the following command:

   ```
   -STOP DATABASE(database-name) SPACENAM(tablespace-name)
   ```

   This step is necessary to prevent updates to the table space during this procedure in the event that the table space has been left open.
6. Find the target identifiers of the objects you created in step 4 (which consist of a PSID for the table space and the OBIDs for the tables within that table space) by querying the SYSIBM.SYSTABLESPACE and SYSIBM.SYSTABLES catalog tables.

   Product-sensitive Programming Interface

   The following statement returns the object ID for a table space; this is the PSID.

   ```
   SELECT DBID, PSID FROM SYSIBM.SYSTABLESPACE
     WHERE NAME='tablespace_name' and DBNAME='database_name'
       AND CREATOR='creator_name';
   ```

   The following statement returns the object ID for a table:

   ```
   SELECT NAME, OBID FROM SYSIBM.SYSTABLES
     WHERE NAME='table_name'
       AND CREATOR='creator_name';
   ```

   End of Product-sensitive Programming Interface

   These values are returned in decimal format, which is the format that you need for DSN1COPY.
7. Run DSN1COPY with the OBIDXLAT and RESET options to perform the OBID translation and to copy the data from the renamed VSAM data set that contains the dropped table space to the newly defined VSAM data set. Specify the VSAM data set that contains data from the dropped table space as the input data set SYSUT1 in the control statement. Specify each of the input records in the following order in the SYSXLAT file to perform OBID translations:
   a. The DBID that you recorded in step 1 on page 507 as both the translation source and the translation target
   b. The PSID that you recorded in step 1 on page 507 as the translation source and the PSID that you recorded in step 6 as the translation target
   c. The original OBIDs that you recorded in step 1 on page 507 as the translation sources and the OBIDs that you recorded in step 6 as the translation targets

   Be sure that you have named the VSAM data sets correctly by checking messages DSN1998I and DSN1997I after DSN1COPY completes.

For more information about DSN1COPY, see Part 3 of *DB2 Utility Guide and Reference*.

8. Use DSN1COPY with the OBIDXLAT and RESET options to apply any incremental image copies to the recovered table space. You must apply these incremental copies in sequence, and specify the same SYSXLAT records that step 7 specifies.

   Important: After you complete this step, you have essentially recovered the table space to the point in time of the last image copy. If you want to use log records to perform forward recovery on the table space, you must now the IBM DB2 UDB Log Analysis Tool for z/OS.

   For more information about point in time recovery, see "Recovering data to a prior point of consistency" on page 495.

9. Start the table space for normal use by using the following command:

   ```
   -START DATABASE(database-name) SPACENAM(tablespace-name)
   ```

10. Rebuild all indexes on the table space.
11. Execute SELECT statements on each table in the recovered table space to verify the recovery. Include all LOB columns in these queries.
12. Make a full image copy of the table space.

    See "Copying page sets and data sets" on page 489 for more information about the COPY utility.

13. Re-create the objects that are dependent on the table.

    See step 11 of "Recovery of an accidentally dropped table" on page 505 for more information.

DB2-managed data sets: The following procedure recovers dropped table spaces that are part of the catalog. If a consistent full image copy or DSN1COPY file is available, you can use DSN1COPY to recover a dropped table space. To recover a dropped table space, complete the following procedure:

1. Find the original DBID for the database, the PSID for the table space, and the OBIDs of all tables contained in the dropped table space. For information about how to do this, see step 1 of "Recovery of an accidentally dropped table" on page 505.
2. Re-create the table space and all tables. This re-creation can be difficult when any of the following conditions are true:
   - A table definition is not available
   - A table is no longer required

   If you cannot re-create a table, you must use a dummy table to take its place. A *dummy table* is a table with an arbitrary structure of columns that you delete after you recover the dropped table space.

   Attention: When you use a dummy table, you lose all data from the dropped table that you do not re-create.

3. Re-create auxiliary tables and indexes if a LOB table space has been dropped.
4. To allow DSN1COPY to access the DB2 data set, stop the table space with the following command:

   ```
   -STOP DATABASE(database-name) SPACENAM(tablespace-name)
   ```

5. Find the new PSID, and OBIDs by querying the DB2 catalog as described in step 6 of "User-managed data sets" on page 507. (Find the OBID of the dummy table that you created in step 2 if you could not re-create a table.)
6. Run DSN1COPY with the OBIDXLAT and RESET options to translate the OBID and to copy data from a previous full image copy data set, inline copy data set, or DSN1COPY file. Use one of these copies as the input data set

SYSUT1 in the control statement. Specify each of the input records in the following order in the SYSXLAT file to perform OBID translations:

a. The DBID that you recorded in step 1 as both the translation source and the translation target
b. The PSID that you recorded in step 1 as the translation source and the PSID that you recorded in step 5 as the translation target
c. The OBIDs that you recorded in step 1 as the translation sources and the OBIDs that you recorded in step 5 as the translation targets

Be sure that you name the VSAM data sets correctly by checking messages DSN1998I and DSN1997I after DSN1COPY completes.

For more information about DSN1COPY, see Part 3 of *DB2 Utility Guide and Reference*.

7. Use DSN1COPY with the OBIDXLAT and RESET options to apply any incremental image copies to the recovered table space. You must apply these incremental copies in sequence, and specify the same SYSXLAT records that step 7 specifies.

   Important: After you complete this step, you have essentially recovered the table space to the point in time of the last image copy. If you want to use log records to perform forward recovery on the table space, you must use the IBM DB2 UDB Log Analysis Tool for z/OS at this point in the recovery procedure.

   For more information about point in time recovery, see "Recovering data to a prior point of consistency" on page 495.

8. Start the table space for normal by using the following command:

   ```
   -START DATABASE(database-name) SPACENAM(tablespace-name)
   ```

9. Drop all dummy tables. The row structure does not match the table definition. This mismatch makes the data in these tables unusable.
10. Reorganize the table space to remove all rows from dropped tables.
11. Rebuild all indexes on the table space.
12. Execute SELECT statements on each table in the recovered table space to verify the recovery. Include all LOB columns in these queries.
13. Make a full image copy of the table space.

    See "Copying page sets and data sets" on page 489 for more information about the COPY utility.

14. Re-create the objects that are dependent on the table.

    See step 11 on page 507 of "Recovery of an accidentally dropped table" on page 505 for more information.

Discarding SYSCOPY and SYSLGRNX records

Use the MODIFY utility to delete obsolete records from SYSIBM.SYSCOPY and SYSIBM.SYSLGRNX. To keep a table space and its indexes synchronized, the MODIFY utility deletes the SYSCOPY and SYSLGRNX records for the table space and its indexes that are defined with the COPY YES option.

1. Follow the following steps of the procedure that is presented in "Locating archive log data sets" on page 439:
   a. "Resolve indoubt units of recovery" on page 439.
   b. "Find the startup log RBA" on page 440.
   c. "Find the minimum log RBA needed" on page 440. In that step, note the date of the earliest image copy that you intend to keep.

   What copies to keep: The earliest image copies and log data sets that you need for recovery to the present date are not necessarily the earliest ones that you

want to keep. If you foresee resetting the DB2 subsystem to its status at any earlier date, you also need the image copies and log data sets that allow you to recover to that date.

If the most recent image copy of an object is damaged, the RECOVER utility seeks a backup copy. If no backup copy is available, or if the backup is lost or damaged, RECOVER uses a previous image copy. It continues searching until it finds an undamaged image copy or no more image copies exist. The process has important implications for keeping archive log data sets. At the very least, you need all log records since the most recent image copy; to protect against loss of data from damage to that copy, you need log records as far back as the earliest image copy that you keep.

2. Run the MODIFY utility for each table space whose old image copies that you want to discard, using the date of the earliest image copy that you will keep. For example, you could enter:

```
MODIFY RECOVERY TABLESPACE dbname.tsname
                 DELETE DATE date
```

The DELETE DATE option removes records that were written earlier than the given date. You can also use DELETE AGE to remove records that are older than a specified number of days.

You can delete SYSCOPY records for a single partition by naming it with the DSNUM keyword. That option does not delete SYSLGRNX records and does not delete SYSCOPY records that are later than the earliest point to which you can recover the entire table space. Thus, you can still recover by partition after that point.

You cannot run the MODIFY utility on a table space that is in RECOVER-pending status.

System-level point in time recovery

DB2 provides the following solutions for system-level recovery:

- "Recovery to an arbitrary point in time" on page 513. The point in time is either between two backup times or between the last backup time and the current time. In recovery to an arbitrary point in time, after the appropriate volume copies have been restored, the outstanding logs are applied to the databases to recover the data to the designated point in time.
- "Recovery to the point in time of a backup" on page 513. In recovery to the point in time of a backup, the recovery involves restoring the appropriate volume copies of the data and logs. The logs are used only to back out inflight transactions on restart.
- "Remote site recovery from a disaster at a local site" on page 514. In remote site recovery, the recovery is determined by what you keep at the remote site. For example, in addition to offsite tapes of backups, current logs might have been transmitted electronically. If so, the current logs can be applied to the databases after the data and logs are restored from the offsite tapes.

In this section, a DB2 system is either a single non-data sharing DB2 subsystem or a DB2 data sharing group.

System-level backup and restore

The DB2 solutions described in this section use the BACKUP SYSTEM and RESTORE SYSTEM online utilities. For system-level backup and recovery procedures that do not use these utilities, see "Backing up with RVA storage control or Enterprise Storage Server" on page 491 and "Remote site recovery from a disaster at the local site" on page 558.

The online utilities BACKUP SYSTEM and RESTORE SYSTEM are described in Part 2 of *DB2 Utility Guide and Reference*, and the stand-alone utility DSNJU003 is described in Part 3 of *DB2 Utility Guide and Reference*.

BACKUP SYSTEM online utility: This utility invokes z/OS Version 1 Release 5 DFSMShsm services to take volume copies of the data. All DB2 data sets that are to be copied (and then recovered) must be managed by SMS. This utility works in both data-sharing and non-data sharing DB2 systems.

The BACKUP SYSTEM utility requires z/OS Version 1 Release 5 data structures called copy pools. Because these data structures are implemented in z/OS, DB2 cannot generate copy pools automatically. Before you invoke the BACKUP SYSTEM utility, copy pools must be allocated in z/OS. For a more detailed description of how DB2 uses copy pools, see "Using DFSMShsm with the BACKUP SYSTEM utility" on page 37. For information about how to allocate a copy pool in z/OS, see *z/OS DFSMSdfp Storage Administration Reference*.

The BACKUP SYSTEM utility invokes the DFSMShsm fast replication function to take volume level backups using FlashCopy.

You can use the BACKUP SYSTEM utility to ease the task of managing data recovery. Choose either DATA ONLY or FULL (data and logs), depending on whether your objective is to minimize the amount of data loss or to minimize the time for the recovery.

Using data-only system backups: The BACKUP SYSTEM DATA ONLY utility control statement creates copies that contain only databases. The RESTORE SYSTEM utility uses these copies to restore databases to an arbitrary point in time. (The RESTORE SYSTEM DATA ONLY utility can also use full system backup copies as input, but the RESTORE SYSTEM utility will not restore log volumes, which the BACKUP SYSTEM FULL utility control statement creates.)

In this type of recovery, you lose only a few seconds of data (or none), but your recovery time could be an hour or more (because recovery takes longer than restart). For more information about recovering the DB2 system to an arbitrary point in time, see "Recovery to an arbitrary point in time" on page 513.

Using full system backups: The BACKUP SYSTEM FULL utility control statement creates copies that contain both logs and databases. With these copies, you can recover your DB2 system to the point in time of a backup using normal DB2 restart recovery, or to an arbitrary point in time by using the RECOVER SYSTEM utility.

When you recover your DB2 system to the point in time of a full system backup, you could lose a few hours of data, but your recovery time would be only a few minutes (because restart is fast). For more information about this type of recovery, see "Recovery to the point in time of a backup" on page 513.

RESTORE SYSTEM online utility: This utility invokes z/OS Version 1 Release 5 DFSMShsm services to recover a DB2 system to a prior point in time by restoring the databases in the volume copies that have been provided by the BACKUP SYSTEM utility. After restoring the data, this utility can then recover to an arbitrary point in time.

The SYSPITR option of DSNJU003 CRESTART allows you to create a conditional restart control record (CRCR) to truncate logs for system point in time recovery in preparation for running RESTORE SYSTEM.

If you restore the system data by some other means, use the RESTORE SYSTEM utility with the LOGONLY option to skip the restore phase, and use the CRCR to apply the logs to the restored databases.

Recovery to an arbitrary point in time

To minimize the amount of data that you lose when you recover, recover to an arbitrary point in time. To recover to an arbitrary point in time, you must perform the following backup and recovery procedures.

Backup: You must perform the following procedure before an event occurs that creates a need to recover your DB2 system:

1. Use BACKUP SYSTEM DATA ONLY to take the backup. You can also take a FULL backup, although it is not needed.

Recovery: If you have performed the appropriate backup procedures, you can use the following procedure to recover your DB2 system to an arbitrary point in time:

1. Stop the DB2 subsystem. If your system is a data sharing group, stop all members of the group.
2. Run DSNJU003 (change log inventory) with the CRESTART SYSPITR option specifying the log truncation point that corresponds to the point in time to which you want to recover the system. For data sharing systems, run DSNJU003 on all active members of the data-sharing group, and specify the same LRSN truncation point for each member. If the point in time that you specify for recovery is prior to the oldest system backup, you must manually restore the volume backup from tape.
3. For data sharing systems, delete all CF structures that the data-sharing group owns.
4. Start DB2. For data sharing systems, start all active members.
5. Submit the RESTORE SYSTEM utility job. If you manually restored the backup, use the LOGONLY option of RESTORE SYSTEM to apply the current logs.

Recovery to the point in time of a backup

Recovery to the point in time of a backup minimizes the amount of time that is required for recovery. You can recover using backups that either the BACKUP SYSTEM utility or FlashCopy create.

Using backups from BACKUP SYSTEM: To recover a DB2 system to the point in time of a back up that the BACKUP SYSTEM utility creates, you must perform the following backup and recovery procedures. For more information about the BACKUP SYSTEM utility, see “BACKUP SYSTEM online utility” on page 512.

Backup: You must perform the following procedure before an event occurs that creates a need to recover your DB2 system.

1. Use BACKUP SYSTEM FULL to take the system backup. DFSMShsm maintains up to 85 versions of system backups on disk at any given time.

Recovery: If you have performed the appropriate backup procedures, you can use the following procedure to recover your DB2 system to the point in time of a backup:

1. Stop the DB2 subsystem. For data sharing systems, stop all members of the group.
2. Use the DFSMShsm command FRRECOV * COPYPOOL(*cpname*) GENERATION(*gen*) to restore the database and log copy pools that the BACKUP SYSTEM utility creates. In this command, *cpname* specifies the name of the copy pool, and *gen* specifies which version of the copy pool is restored.
3. For data sharing systems, delete all CF structures that are owned by this group.
4. Start DB2. For data sharing systems, start all active members.
5. For data sharing systems, execute the GRECP and LPL recovery, which recovers the changed data that was stored in the coupling facility at the time of the backup.

Using backups from FlashCopy: To recover a DB2 system to the point in time of a backup that FlashCopy creates, you must perform the following backup and recovery procedures. For more information about the FlashCopy function, see *z/OS DFSMS Advanced Copy Services*.

Backup: You must perform the following procedure before an event occurs that creates a need to recover your DB2 system:

1. Issue the DB2 command SET LOG SUSPEND to suspend logging and update activity, and to quiesce 32K page writes and data set extensions. For data sharing systems, issue the command to each member of the group.
2. Use the FlashCopy function to copy all DB2 volumes. Include any ICF catalogs that are used by DB2, as well as active logs and BSDSs.
3. Issue the DB2 command SET LOG RESUME to resume normal DB2 update activity. To save disk space, you can use DFSMSdss to dump the disk copies you just created to a lower-cost medium, such as tape.

Recovery: If you have performed the appropriate backup procedures, you can use the following procedure to recover your DB2 system to the point in time of a backup:

1. Stop the DB2 subsystem. For data sharing systems, stop all members of the group.
2. Use DFSMSdss RESTORE to restore the FlashCopy data sets to disk. See *z/OS DFSMSdss Storage Administration Reference* for more information.
3. For data sharing systems, delete all CF structures that are owned by this group.
4. Start DB2. For data sharing systems, start all active members.
5. For data sharing systems, execute the GRECP and LPL recovery, which recovers the changed data that was stored in the coupling facility at the time of the backup.

Remote site recovery from a disaster at a local site

After a disaster at your local site, you can recover at a remote site by using the RESTORE SYSTEM utility. You can use RESTORE SYSTEM to recover DB2 from the backups that the BACKUP SYSTEM utility produces, or you can use RESTORE SYSTEM LOGONLY to recover from backups you produce in some other way. For DB2 remote-site recovery procedures that do not use the RESTORE SYSTEM utility, see "Remote site recovery from a disaster at the local site" on page 558.

Recovering with BACKUP SYSTEM: The following procedures use the BACKUP SYSTEM and RESTORE SYSTEM utilities to recover your DB2 system:

Preparation:

1. Use BACKUP SYSTEM FULL to take the system backup.
2. Transport the system backups to the remote site.

Recovery:

1. Run the DSNJU003 utility using the control statement `CRESTART CREATE, SYSPITR=log-truncation-point`, where *log-truncation-point* is the RBA or LRSN of the point to which you want to recover.
2. Start DB2.
3. Run the RESTORE SYSTEM utility using the control statement `RESTORE SYSTEM` to recover to the current time (or to the time of the last log transmission from the local site).

Recovering without BACKUP SYSTEM: Use the following procedures to recover your DB2 system if you **do not** use the BACKUP SYSTEM utility to produce backups:

Preparation:

1. Issue the DB2 command SET LOG SUSPEND to suspend logging and update activity, and to quiesce 32K page writes and data set extensions. For data sharing systems, issue the command to each member of the data sharing group.
2. Use the FlashCopy function to copy all DB2 volumes. Include any ICF catalogs that are used by DB2, as well as active logs and BSDSs.
3. Issue the DB2 command SET LOG RESUME to resume normal DB2 update activity.
4. Use DFSMSdss to dump the disk copies that you just created to tape, and then transport this tape to the remote site. You can also use other methods to transmit the copies you make to the remote site.

Recovery:

1. Use DFSMSdss to restore the FlashCopy data sets to disk.
2. Run the DSNJU003 utility using the control statement `CRESTART CREATE, SYSPITR=log-truncation-point`, where *log-truncation-point* is the RBA or LRSN of the point to which you want to recover.
3. Start DB2.
4. Run the RESTORE SYSTEM utility using the control statement `RESTORE SYSTEM LOGONLY` to recover to the current time (or to the time of the last log transmission from the local site).

Chapter 22. Recovery scenarios

This chapter contains problem scenarios and the recommended procedures for restarting and recovering DB2. The following situations are described:

"IRLM failure recovery"
"z/OS or power failure recovery" on page 518
"Disk failure recovery" on page 518
"Application error recovery" on page 520
"IMS-related failure recovery" on page 521
"CICS-related failure recovery" on page 525
"Subsystem termination recovery" on page 530
"Resource failure recovery" on page 531
- "Active log failure recovery" on page 531
- "Archive log failure recovery" on page 535
- "BSDS failure recovery" on page 538
- "Recovering the BSDS from a backup copy" on page 539

"DB2 database failures" on page 542
"Recovery from down-level page sets" on page 543
"Table space input/output error recovery" on page 545
"DB2 catalog or directory input/output errors" on page 546
"Integrated catalog facility failure recovery" on page 547
- "Recovery when the VSAM volume data set (VVDS) is destroyed" on page 547
- "Out of disk space or extent limit recovery" on page 548

"Violation of referential constraint recovery" on page 552
"Distributed data facility failure recovery" on page 553
"Remote site recovery from a disaster at the local site" on page 558
"Using a tracker site for disaster recovery" on page 568
"Resolving indoubt threads" on page 582
- "Communication failure recovery" on page 583
- "Making a heuristic decision" on page 584
- "Recovery from an IMS outage that results in an IMS cold start" on page 585
- "Recovery from a DB2 outage at a requester that results in a DB2 cold start" on page 586
- "Recovery from a DB2 outage at a server that results in a DB2 cold start" on page 589
- "Correcting a heuristic decision" on page 589

IRLM failure recovery

Problem: The IRLM fails in a wait, loop, or abend.

Symptom: The IRLM abends and the following message appears:

```
DXR122E irlmnm ABEND UNDER IRLM TCB/SRB IN MODULE xxxxxxxx
ABEND CODE zzzz
```

System action: If the IRLM abends, DB2 terminates. If the IRLM waits or loops, then terminate the IRLM, and DB2 terminates automatically.

System programmer action: None.

Operator action:

© Copyright IBM Corp. 1982, 2008

- Start the IRLM if you did not set it for automatic start when you installed DB2. (For instructions on starting the IRLM, see "Starting the IRLM" on page 377.)
- Start DB2. (For instructions, see "Starting DB2" on page 324.)
- Issue the command /START SUBSYS *ssid* to connect IMS to DB2.
- Issue the command DSNC STRT to connect CICS to DB2. (See "Connecting from CICS" on page 384.)

z/OS or power failure recovery

Problem: z/OS or processor power fails.

Symptom: No processing is occurring.

System action: None.

System programmer action: None.

Operator action:

1. IPL z/OS and initialize the job entry subsystem.
2. If you normally run VTAM with DB2, start VTAM at this point.
3. Start the IRLM if you did not set it for automatic start when you installed DB2. (See "Starting the IRLM" on page 377.)
4. Start DB2. (See "Starting DB2" on page 324.)
5. Use the RECOVER POSTPONED command if postponed-abort units of recovery were reported after restarting DB2, and the AUTO option of the LIMIT BACKOUT field on installation panel DSNTIPL was not specified.
6. Restart IMS or CICS.
   a. IMS automatically connects and resynchronizes when it is restarted. (See "Connecting to the IMS control region" on page 388.)
   b. CICS automatically connects to DB2 if the CICS PLT contains an entry for the attach module DSNCCOM0. Alternatively, use the command DSNC STRT to connect the CICS attachment facility to DB2. (See "Connecting from CICS" on page 384.)

If you know that a power failure is imminent, it is a good idea to issue STOP DB2 MODE(FORCE) to allow DB2 to come down cleanly before the power is interrupted. If DB2 is unable to stop completely before the power failure, the situation is no worse than if DB2 were still up.

Disk failure recovery

Problem: A disk hardware failure occurs, resulting in the loss of an entire unit.

Symptom: No I/O activity for the affected disk address. Databases and tables residing on the affected unit are unavailable.

System action: None

System programmer action: None

Operator action: Attempt recovery with the following procedure:

1. Assure that there are no incomplete I/O requests against the failing device. One way to do this is to force the volume off line by issuing the following z/OS command:

   ```
   VARY xxx,OFFLINE,FORCE
   ```

   where *xxx* is the unit address.

   To check disk status you can issue:

   ```
   D U,DASD,ONLINE
   ```

   The following console message is displayed after you have forced a volume offline:

   ```
   UNIT  TYPE  STATUS   VOLSER  VOLSTATE
   4B1   3390  O-BOX    XTRA02  PRIV/RSDNT
   ```

   The disk unit is now available for service.

   If you have previously set the I/O timing interval for the device class, the I/O timing facility should terminate all incomplete requests at the end of the specified time interval, and you can proceed to the next step without varying the volume off line. You can set the I/O timing interval either through the IECIOSxx z/OS parameter library member or by issuing the z/OS command

   ```
   SETIOS MIH,DEV=devnum,IOTIMING=mm:ss.
   ```

   For more information about the I/O timing facility, see *z/OS MVS Initialization and Tuning Reference* and *z/OS MVS System Commands*.

2. An authorized operator issues the following command to stop all databases and table spaces residing on the affected volume:

   ```
   -STOP DATABASE(database-name) SPACENAM(space-name)
   ```

   If the disk unit must be disconnected for repair, all databases and table spaces on all volumes in the disk unit must be stopped.

3. Select a spare disk pack and use ICKDSF to initialize from scratch a disk unit with a different unit address (*yyy*) and the same volser.

   ```
   // Job
   //ICKDSF   EXEC PGM=ICKDSF
   //SYSPRINT DD   SYSOUT=*
   //SYSIN    DD   *
        REVAL UNITADDRESS(yyy) VERIFY(volser)
   ```

   If you are initializing a 3380 or 3390 volume, use REVAL with the VERIFY parameter to ensure you are initializing the volume you want, or to revalidate the volume's home address and record 0. Details are provided in *Device Support Facilities User's Guide and Reference*. Alternatively, use ISMF to initialize the disk unit.

4. Issue this z/OS console command. *yyy* is the new unit address.

   ```
   VARY yyy,ONLINE
   ```

5. To check disk status you can issue:

   ```
   D U,DASD,ONLINE
   ```

   The following console message is displayed:

   ```
   UNIT  TYPE  STATUS   VOLSER  VOLSTATE
   7D4   3390  O        XTRA02  PRIV/RSDNT
   ```

6. Issue the following command to start all the appropriate databases and table spaces that had been stopped previously:

   ```
   -START DATABASE(database-name) SPACENAM(space-name)
   ```

7. Delete all table spaces (VSAM linear data sets) from the ICF catalog by issuing the following access method services command for each one of them:

   ```
   DELETE catnam.DSNDBC.dbname.tsname.y0001.A00x CLUSTER NOSCRATCH
   ```

where *y* can be either I or J.

Access method services commands are described in detail in *z/OS DFSMS Access Method Services for Catalogs*.

8. For user-managed table spaces, the VSAM cluster and data components must be defined for the new volume by issuing the access method services DEFINE CLUSTER command with the data set name:

   `catnam.DSNDBC.dbname.tsname.y0001.A00x`

   where *y* can be either I or J, and *x* is C (for VSAM clusters) or D (for VSAM data components).

   This data set is the same as defined in Step 7. Detailed requirements for user-managed data sets are described in "Requirements for your own data sets" on page 38.

   For a user-defined table space, the new data set must be defined before an attempt to recover it. Table spaces defined in storage groups can be recovered without prior definition.

9. Recover the table spaces using the RECOVER utility. Additional information and procedures for recovering data can be found in "Recovering page sets and data sets" on page 491.

Application error recovery

Problem: An application program placed a logically incorrect value in a table.

Symptom: SQL SELECT returns unexpected data.

System action: The system returns SQLCODE 0 for the SELECT statement, because the error was not in SQL or DB2, but in the application program. That error can be identified and corrected, but the data in the table is now inaccurate.

System programmer action: You might be able to use RECOVER TOLOGPOINT to restore the database to a point before the error occurred, but there are many circumstances under which you must manually back out the changes introduced by the application. Among those are:

- Other applications changed the database after the error occurred. If you recover the table spaces modified by the bad application, you would lose all subsequent changes made by the other applications.
- DB2 checkpoints were taken after the error occurred. In this case, you can use RECOVER TOLOGPOINT to restore the data up to the last checkpoint before the error occurred, but all subsequent changes to the database are lost.

If you have a situation for which using RECOVER TOLOGPOINT is appropriate, you can use procedures similar to those that follow to back out the changes made by the bad application. For a discussion of RECOVER TOLOGPOINT, see "Using RECOVER to restore data to a previous point in time" on page 500.

Procedure 1: If you have established a quiesce point

1. Run the REPORT utility twice, once using the RECOVERY option and once using the TABLESPACESET option. On each run, specify the table space containing the inaccurate data. If you want to recover to the last quiesce point, specify the option CURRENT when running REPORT RECOVERY.
2. Examine the REPORT output to determine the RBA of the quiesce point.

3. Execute RECOVER TOLOGPOINT with the RBA that you found, specifying the names of all related table spaces. Recovering all related table spaces to the same quiesce point prevents violations of referential constraints.

Procedure 2: If you have not established a quiesce point

If you use this procedure, you will lose any updates to the database that occurred after the last checkpoint before the application error occurred.

1. Run the DSN1LOGP stand-alone utility on the log scope available at DB2 restart, using the SUMMARY(ONLY) option. For instructions on running DSN1LOGP, see Part 3 of *DB2 Utility Guide and Reference*.
2. Determine the RBA of the most recent checkpoint before the first bad update occurred, from one of the following sources:
   - Message DSNR003I on the operator's console. It looks (in part) like this:

     ```
     DSNR003I RESTART  ..... PRIOR CHECKPOINT
                             RBA=000007425468
     ```

     The required RBA in this example is X'7425468'.

     This technique works only if there have been no checkpoints since the application introduced the bad updates.
   - Output from the print log map utility. You must know the time that the first bad update occurred. Find the last BEGIN CHECKPOINT RBA before that time.
3. Run DSN1LOGP again, using SUMMARY(ONLY) and specify the checkpoint RBA as the value of RBASTART. The output lists the work in the recovery log, including information about the most recent complete checkpoint, a summary of all processing occurring, and an identification of the databases affected by each active user. Sample output is shown in Figure 61 on page 600.
4. One of the messages in the output (identified as DSN1151I or DSN1162I) describes the unit of recovery in which the error was made. To find the unit of recovery, use your knowledge of the time the program was run (START DATE= and TIME=), the connection ID (CONNID=), authorization ID (AUTHID=), and plan name (PLAN=). In that message, find the starting RBA as the value of START=.
5. Execute RECOVER TOLOGPOINT with the starting RBA you found in the previous step.
6. Recover any related table spaces or indexes to the same point in time.

Operator action: None.

IMS-related failure recovery

This section includes scenarios for problems that can be encountered in the IMS environment:

"IMS control region (CTL) failure recovery" on page 522
"Resolution of indoubt units of recovery" on page 522
"IMS application failure recovery" on page 524

DB2 can be used in an XRF (Extended Recovery Facility) recovery environment with IMS. See "Extended recovery facility (XRF) toleration" on page 472 for more information about using XRF with IMS.

IMS control region (CTL) failure recovery

Problem: The IMS control region fails.

Symptom:

- IMS waits, loops, or abends.
- DB2 attempts to send the following message to the IMS master terminal during an abend:

  ```
  DSNM002I  IMS/TM xxxx DISCONNECTED FROM SUBSYSTEM
            yyyy  RC=RC
  ```

  This message cannot be sent if the failure prevents messages from being displayed.
- DB2 does not send any messages related to this problem to the z/OS console.

System action:

- DB2 detects that IMS has failed.
- DB2 either backs out or commits work in process.
- DB2 saves indoubt units of recovery. (These must be resolved at reconnection time.)

System programmer action: None.

Operator action:

1. Use normal IMS restart procedures, which include starting IMS by issuing the z/OS START IMS command.
2. The following results occur:
   - All DL/I and DB2 updates that have not been committed are backed out.
   - IMS is automatically reconnected to DB2.
   - IMS passes the recovery information for each entry to DB2 through the IMS attachment facility. (IMS indicates whether to commit or roll back.)
   - DB2 resolves the entries according to IMS instructions.

Resolution of indoubt units of recovery

This section describes two different problems.

Problem 1

There are unresolved indoubt units of recovery. When IMS connects to DB2, DB2 has one or more indoubt units of recovery that have not been resolved.

Symptom: If DB2 has indoubt units of recovery that IMS did not resolve, the following message is issued at the IMS master terminal:

```
DSNM004I  RESOLVE INDOUBT ENTRY(S) ARE OUTSTANDING FOR
          SUBSYSTEM xxxx
```

When this message is issued, IMS was either cold started or it was started with an incomplete log tape. This message could also be issued if DB2 or IMS had an abend due to a software error or other subsystem failure.

System action:

- The connection remains active.
- IMS applications can still access DB2 databases.
- Some DB2 resources remain locked out.

If the indoubt thread is not resolved, the IMS message queues can start to back up. If the IMS queues fill to capacity, IMS terminates. Therefore, users must be aware of this potential difficulty and must monitor IMS until the indoubt units of work are fully resolved.

System programmer action:

1. Force the IMS log closed using /DBR FEOV, and then archive the IMS log. Use the command DFSERA10 to print the records from the previous IMS log tape for the last transaction processed in each dependent region. Record the PSB and the commit status from the X'37' log containing the recovery ID.
2. Run the DL/I batch job to back out each PSB involved that has not reached a commit point. The process might take some time because transactions are still being processed. It might also lock up a number of records, which could impact the rest of the processing and the rest of the message queues.
3. Enter the DB2 command DISPLAY THREAD (*imsid*) TYPE (INDOUBT).
4. Compare the NIDs (IMSID + OASN in hexadecimal) displayed in the DISPLAY THREAD messages with the OASNs (4 bytes decimal) shown in the DFSERA10 output. Decide whether to commit or roll back.
5. Use DFSERA10 to print the X'5501FE' records from the current IMS log tape. Every unit of recovery that undergoes indoubt resolution processing is recorded; each record with an 'IDBT' code is still indoubt. Note the correlation ID and the recovery ID, because they will be used during step 6.
6. Enter the following DB2 command, choosing to commit or roll back, and specifying the correlation ID:

   ```
   -RECOVER INDOUBT (imsid) ACTION(COMMIT|ABORT) NID (nid)
   ```

   If the command is rejected because there are more network IDs associated, use the same command again, substituting the recovery ID for the network ID.

(For a description of the OASN and the NID, see “Duplicate correlation IDs” on page 392.)

Operator action: Contact the system programmer.

Problem 2

Committed units of recovery should be aborted. At the time IMS connects to DB2, DB2 has committed one or more indoubt units of recovery that IMS says should be rolled back.

Symptom: By DB2 restart time, DB2 has committed and rolled back those units of recovery about which DB2 was not indoubt. DB2 records those decisions, and at connect time, verifies that they are consistent with the IMS/TM decisions.

An inconsistency can occur when the DB2 RECOVER INDOUBT command is used before IMS attempted to reconnect. If this happens, the following message is issued at the IMS master terminal:

```
DSNM005I  IMS/TM RESOLVE INDOUBT PROTOCOL PROBLEM WITH
          SUBSYSTEM xxxx
```

Because DB2 tells IMS to retain the inconsistent entries, the following message is issued when the resolution attempt ends:

```
DFS3602I  xxxx SUBSYSTEM RESOLVE-INDOUBT FAILURE,
          RC=yyyy
```

System action:
- The connection between DB2 and IMS remains active.
- DB2 and IMS continue processing.
- No DB2 locks are held.
- No units of work are in an incomplete state.

System programmer action: Do not use the DB2 command RECOVER INDOUBT. The problem is that DB2 was *not* indoubt but should have been.

Database updates have most likely been committed on one side (IMS or DB2) and rolled back on the other side. (For a description of the OASN and the NID, see "Duplicate correlation IDs" on page 392.)

1. Enter the IMS command /DISPLAY OASN SUBSYS DB2 to display the IMS list of units of recovery that need to be resolved. The /DISPLAY OASN SUBSYS DB2 command produces the OASNs in a decimal format, not a hexadecimal format.
2. Issue the IMS command /CHANGE SUBSYS DB2 RESET to reset all the entries in the list. (No entries are passed to DB2.)
3. Use DFSERA10 to print the log records recorded at the time of failure and during restart. Look at the X'37', X'56', and X'5501FE' records at reconnect time. Notify the IBM support center about the problem.
4. Determine what the inconsistent unit of recovery was doing by using the log information, and manually make the DL/I and DB2 databases consistent.

Operator action: None.

IMS application failure recovery

This section describes two different problems.

Problem 1

An IMS application abends.

Symptom: The following messages appear at the IMS master terminal and at the LTERM that entered the transaction involved:

```
DFS555  - TRAN tttttttt ABEND (SYSIDssss);
          MSG IN PROCESS: xxxx (up to 78 bytes of data) timestamp
DFS555A - SUBSYSTEM xxxx OASN yyyyyyyyyyyyyyyy STATUS COMMIT|ABORT
```

System action:
The failing unit of recovery is backed out by both DL/I and DB2.
The connection between IMS and DB2 remains active.

System programmer action: None.

Operator action: If you think the problem was caused by a user error, refer to Part 2 of *DB2 Application Programming and SQL Guide*. For procedures to diagnose DB2 problems, rather than user errors, refer to Part 3 of *DB2 Diagnosis Guide and Reference*. If necessary, contact the IBM support center for assistance.

Problem 2

DB2 has failed or is not running.

Symptom: One of the following status situations exists:

- If you specified error option Q, the program terminates with a U3051 user abend completion code.

- If you specified error option A, the program terminates with a U3047 user abend completion code.

In both cases, the master terminal receives a message (IMS message number DFS554), and the terminal involved also receives a message (DFS555).

System action: None.

System programmer action: None.

Operator action:
1. Restart DB2.
2. Follow the standard IMS procedures for handling application abends.

CICS-related failure recovery

This section includes scenarios for problems that can be encountered in the CICS environment:

"CICS application failure recovery"
"Recovery when CICS is not operational"
"Recovery when CICS cannot connect to DB2" on page 526
"Manually recovering CICS indoubt units of recovery" on page 527
"CICS attachment facility failure recovery" on page 530

DB2 can be used in an XRF (Extended Recovery Facility) recovery environment with CICS. See "Extended recovery facility (XRF) toleration" on page 472 for more information about using XRF with CICS.

CICS application failure recovery

Problem: A CICS application abends.

Symptom: The following message is sent to the user's terminal.

```
DFH2206 TRANSACTION tranid ABEND abcode BACKOUT SUCCESSFUL
```

tranid can represent any abending CICS transaction and *abcode* is the abend code.

System action:
The failing unit of recovery is backed out in both CICS and DB2.
The connection remains.

System programmer action: None.

Operator action:
1. For information about the CICS attachment facility abend, refer to Part 2 of *DB2 Messages*.
2. For an *AEY9* abend, start the CICS attachment facility.
3. For an *ASP7* abend, determine why the CICS SYNCPOINT was unsuccessful.
4. For other abends, see *DB2 Diagnosis Guide and Reference* or *CICS Transaction Server for z/OS Problem Determination Guide* for diagnostic procedures.

Recovery when CICS is not operational

Problem: CICS is not operational.

Symptom: More than one symptom is possible.

- CICS waits or loops.

  Because DB2 cannot detect a wait or loop in CICS, you must find the origin of the wait or the loop. The origin can be in CICS, CICS applications, or in the CICS attachment facility. For diagnostic procedures for waits and loops, see Part 2 of *DB2 Diagnosis Guide and Reference*.
- CICS abends.
  - CICS issues messages indicating an abend occurred and requests abend dumps of the CICS region. See *CICS Transaction Server for z/OS Problem Determination Guide* for more information.
  - If threads are connected to DB2 when CICS terminates, DB2 issues message DSN3201I. The message indicates that DB2 end-of-task (EOT) routines have been run to clean up and disconnect any connected threads.

System action: DB2 performs each of the following actions:
Detects the CICS failure.
Backs out inflight work.
Saves indoubt units of recovery to be resolved when CICS is reconnected.

Operator action:

1. Correct the problem that caused CICS to terminate abnormally.
2. Do an emergency restart of CICS. The emergency restart performs each of the following actions:
   - Backs out inflight transactions that changed CICS resources
   - Remembers the transactions with access to DB2 that might be indoubt.
3. Start the CICS attachment facility by entering the appropriate command for your release of CICS. See "Connecting from CICS" on page 384. The CICS attachment faciliy performs the following actions:
   - Initializes and reconnects to DB2.
   - Requests information from DB2 about the indoubt units of recovery and passes the information to CICS.
   - Allows CICS to resolve the indoubt units of recovery.

Recovery when CICS cannot connect to DB2

Problem: The CICS attachment facility cannot connect to DB2.

Symptom:

- CICS remains operative, but the CICS attachment facility abends.
- The CICS attachment facility issues a message giving the reason for the connection failure, or it requests an X'04E' dump.
- The reason code in the X'04E' dump gives the reason for failure.
- CICS issues message DFH2206 indicating that the CICS attach facility has terminated abnormally with the DSNC abend code.
- CICS application programs trying to access DB2 while the CICS attachment facility is inactive are abnormally terminated. The code AEY9 is issued.

System Action: CICS backs out the abnormally terminated transaction and treats it like an application abend.

Operator action:

1. Start the CICS attachment facility by entering the appropriate command for your release of CICS. See "Connecting from CICS" on page 384.

2. The CICS attachment facility initializes and reconnects to DB2.
3. The CICS attachment facility requests information about the indoubt units of recovery and passes the information to CICS.
4. CICS resolves the indoubt units of recovery.

Manually recovering CICS indoubt units of recovery

When the attachment facility abends, CICS and DB2 build indoubt lists either dynamically or during restart, depending on the failing subsystem.

For CICS, a DB2 unit of recovery could be indoubt if the forget entry (X'FD59') of the task-related installation exit is absent from the CICS system journal. The indoubt condition applies only to the DB2 UR, because CICS will have already committed or backed out any changes to its resources.

A DB2 unit of recovery is indoubt for DB2 if an End Phase 1 is present and the Begin Phase 2 is absent.

Problem: When CICS connects to DB2, there are one or more indoubt units of recovery that have not been resolved.

Symptom: One of the following messages is sent to the user-named CICS destination specified in the ERRDEST field in the resource control table (RCT): DSN2001I, DSN2034I, DSN2035I, or DSN2036I.

System action: The system action is summarized in Table 99:

Table 99. CICS abnormal indoubt unit of recovery situations

Message ID	Meaning
DSN2001I	The named unit of recovery cannot be resolved by CICS because CICS was cold started. The CICS attachment facility continues the startup process.
DSN2034I	The named unit of recovery is not indoubt for DB2, but is indoubt according to CICS log information. The reason is probably a CICS restart with the wrong tape. It could also be caused by a DB2 restart to a prior point in time.
DSN2035I	The named unit of recovery is indoubt for DB2, but is not in the CICS indoubt list. This is most likely due to an incorrect CICS restart. The CICS attachment facility continues the startup process and provides a transaction dump. It could also be caused by a DB2 restart to a prior point in time.
DSN2036I	CICS indicates roll back for the named unit of recovery, but DB2 has already committed the unit of recovery. The CICS attachment facility continues the startup process.

CICS retains details of indoubt units of recovery that were not resolved during connection start up. An entry is purged when it no longer appears on the list presented by DB2 or, when present, DB2 solves it.

System programmer action: Any indoubt unit of recovery that CICS cannot resolve must be resolved manually by using DB2 commands. This manual procedure should be used rarely within an installation, because it is required only where operational errors or software problems have prevented automatic resolution. *Any inconsistencies found during indoubt resolution must be investigated.*

To recover an indoubt unit, follow these steps:

Step 1: Obtain a list of the indoubt units of recovery from DB2:

Issue the following command:

```
-DISPLAY THREAD (connection-name) TYPE (INDOUBT)
```

You will receive the following messages:

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV406I - INDOUBT THREADS -
COORDINATOR      STATUS      RESET  URID         AUTHID
coordinator-name status      yes/no urid        authid
DISPLAY INDOUBT REPORT COMPLETE
DSN9022I - DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
```

The *corr_id* (correlation ID) for CICS Transaction Server for z/OS 1.1 and previous releases of CICS consists of:

Byte 1 Connection type: G = group, P = pool

Byte 2 Thread type: T = transaction (TYPE=ENTRY), G = group, C = command (TYPE=COMD)

Bytes 3, 4
Thread number

Bytes 5 - 8
Transaction ID

The *corr_id* (correlation ID) for CICS Transaction Server for z/OS 1.2 and subsequent releases of CICS consists of:

Bytes 1 - 4
Thread type: COMD, POOL, or ENTR

Bytes 5 - 8
Transaction ID

Bytes 9 - 12
Unique thread number

Two threads can sometimes have the same correlation ID when the connection has been broken several times and the indoubt units of recovery have not been resolved. In this case, the network ID (NID) must be used instead of the correlation ID to uniquely identify indoubt units of recovery.

The network ID consists of the CICS connection name and a unique number provided by CICS at the time the syncpoint log entries are written. This unique number is an 8-byte store clock value that is stored in records written to both the CICS system log and to the DB2 log at syncpoint processing time. This value is referred to in CICS as the *recovery token*.

Step 2: Scan the CICS log for entries related to a particular unit of recovery: To do this, search the CICS log, looking for a PREPARE record (JCRSTRIDX'F959'), for the task-related installation where the recovery token field (JCSRMTKN) equals the value obtained from the network-ID. The network ID is supplied by DB2 in the DISPLAY THREAD command output.

Locating the prepare log record in the CICS log for the indoubt unit of recovery provides the CICS task number. All other entries on the log for this CICS task can be located using this number.

CICS journal print utility DFHJUP can be used when scanning the log. See *CICS Transaction Server for z/OS Operations and Utilities Guide* for details on how to use this program.

Step 3: Scan the DB2 log for entries related to a particular unit of recovery: To do this, scan the DB2 log to locate the End Phase 1 record with the network ID required. Then use the URID from this record to obtain the rest of the log records for this unit of recovery.

When scanning the DB2 log, note that the DB2 start up message DSNJ099I provides the start log RBA for this session.

The DSN1LOGP utility can be used for that purpose. See Part 3 of *DB2 Utility Guide and Reference* for details on how to use this program.

Step 4: If needed, do indoubt resolution in DB2: DB2 can be directed to take the recovery action for an indoubt unit of recovery using a DB2 RECOVER INDOUBT command. Where the correlation ID is unique, use the following command:

```
DSNC -RECOVER INDOUBT (connection-name)
              ACTION (COMMIT/ABORT)
              ID (correlation-id)
```

If the transaction is a pool thread, use the value of the correlation ID (*corr_id*) returned by DISPLAY THREAD for *thread#.tranid* in the command RECOVER INDOUBT. In this case, the first letter of the correlation ID is P. The transaction ID is in characters five through eight of the correlation ID.

If the transaction is assigned to a group (group is a result of using an entry thread), use *thread#.groupname* instead of *thread#.tranid*. In this case, the first letter of the correlation ID is a G and the group name is in characters five through eight of the correlation ID. *groupname* is the first transaction listed in a group.

Where the correlation ID is not unique, use the following command:

```
DSNC -RECOVER INDOUBT (connection-name)
              ACTION (COMMIT/ABORT)
              NID (network-id)
```

When two threads have the same correlation ID, use the NID keyword instead of the ID keyword. The NID value uniquely identifies the work unit.

To recover all threads associated with *connection-name*, omit the ID option.

The command results that are in either of the following messages to indicate whether the thread is committed or rolled back:

```
DSNV414I - THREAD thread#.tranid COMMIT SCHEDULED
DSNV415I - THREAD thread#.tranid ABORT SCHEDULED
```

When performing indoubt resolution, note that CICS and the attachment facility are not aware of the commands to DB2 to commit or abort indoubt units of recovery, because only DB2 resources are affected. However, CICS keeps details

about the indoubt threads that could not be resolved by DB2. This information is purged either when the list presented is empty, or when the list does not include a unit of recovery that CICS remembers.

Operator action: Contact the system programmer.

CICS attachment facility failure recovery

Problem: The CICS attachment facility abends, or a CICS attachment thread subtask abends. CICS and DB2 remain active.

Symptom:

- If the main subtask abends, an abend dump is requested. The contents of the dump indicate the cause of the abend. When the dump is issued, shutdown of the CICS attachment facility begins.
- If a thread subtask terminates abnormally, an X'04E' dump is issued and the CICS application abends with a DSNC dump code. The X'04E' dump should show the cause of the abend. The CICS attachment facility remains active.

System action:

- The CICS attachment facility shuts down if there is a main subtask abend.
- The matching CICS application abends with a DSNC dump code if a thread subtask abends.

System programmer action: None.

Operator action: Correct the problem that caused the abend by analyzing the CICS formatted transaction dump or subtask SNAP dump. For more information about analyzing these dumps, see Part 2 of *DB2 Messages*. If the CICS attachment facility shuts down, use CICS commands to stop the execution of any CICS-DB2 applications.

Subsystem termination recovery

Problem: Subsystem termination has been started by DB2 or by an operator cancel.

Symptom: Subsystem termination occurs. Usually some specific failure is identified by DB2 messages, and the following messages appear.

On the z/OS console:

```
DSNV086E - DB2 ABNORMAL TERMINATION REASON=XXXXXXXX
DSN3104I - DSN3EC00 - TERMINATION COMPLETE
DSN3100I - DSN3EC00 - SUBSYSTEM ssnm READY FOR -START COMMAND
```

On the IMS master terminal:

```
DSNM002I   IMS/TM xxxx DISCONNECTED FROM SUBSYSTEM
           yyyy  RC=rc
```

On the CICS transient data error destination defined in the RCT:

```
DSNC2025I - THE ATTACHMENT FACILITY IS INACTIVE
```

System action:

- IMS and CICS continue.
- In-process CICS and IMS applications receive SQLCODE -923 (SQLSTATE '57015') when accessing DB2.

In most cases, if an IMS or CICS application program is running when a -923 SQLCODE is returned, an abend occurs. This is because the application program generally terminates when it receives a -923 SQLCODE. To terminate, some synchronization processing occurs (such as a commit). If DB2 is not operational when synchronization processing is attempted by an application program, the application program abends. In-process applications can abend with an abend code X'04F'.

- New IMS applications are handled according to the error options.
  - For option R, SQL return code -923 is sent to the application, and IMS pseudo abends.
  - For option Q, the message is enqueued again and the transaction abends.
  - For option A, the message is discarded and the transaction abends.
- New CICS applications are handled as follows:
  - If the CICS attachment facility has not terminated, the application receives a -923 SQLCODE.
  - If the CICS attachment facility has terminated, the application abends (code AEY9).

Operator action:

1. Restart DB2 by issuing the command START DB2.
2. Reestablish the IMS connection by issuing the IMS command /START SUBSYS DB2.
3. Reestablish the CICS connection by issuing the CICS attachment facility command DSNC STRT.

System programmer action:

1. Use the IFCEREP1 service aid to obtain a listing of the SYS1.LOGREC data set containing the SYS1.LOGREC entries. (For more information about this service aid, refer to the z/OS diagnostic techniques publication about SYS1.LOGREC.)
2. If the subsystem termination was due to a failure, collect material to determine the reason for failure (console log, dump, and SYS1.LOGREC).

Resource failure recovery

This section includes scenarios for problems that can be encountered in the DB2 environment:

"Active log failure recovery"
"Archive log failure recovery" on page 535
"BSDS failure recovery" on page 538
"Recovering the BSDS from a backup copy" on page 539

Active log failure recovery

This section covers some of the more likely active log problems. The following problems are not covered in this section:

- Active log dynamic allocation problems are indicated by message DSNJ103I at startup time.
- Active log open/close problems are indicated by message DSNJ104I.

See Chapter 23, "Recovery from BSDS or log failure during restart," on page 591 for more information about correcting problems that result in a DSNJ103I or a DSNJ104I message.

Problem 1 - Out of space in active logs

The available space in the active log is finite and can be exhausted. It can fill to capacity for one of several reasons, such as delays in offloading and excessive logging.

Symptom: An out of space condition on the active log has very serious consequences. When the active log becomes full, the DB2 subsystem cannot do any work that requires writing to the log until an offload is completed.

Due to the serious implications of this event, the DB2 subsystem issues the following warning message when the last available active log data set is 5% full and reissues the message after each additional 5% of the data set space is filled. Each time the message is issued, the offload process is started. IFCID trace record 0330 is also issued if statistics class 3 is active.

```
DSNJ110E - LAST COPYn ACTIVE LOG DATA SET IS nnn PERCENT FULL
```

If the active log fills to capacity, after having switched to single logging, the following message is issued, and an offload is started. The DB2 subsystem then halts processing until an offload has completed.

```
DSNJ111E - OUT OF SPACE IN ACTIVE LOG DATA SETS
```

Corrective action is required before DB2 can continue processing.

System action: DB2 waits for an available active log data set before resuming normal DB2 processing. Normal shutdown, with either QUIESCE or FORCE, is not possible because the shutdown sequence requires log space to record system events related to shutdown (for example, checkpoint records).

Operator action: Make sure offload is not waiting for a tape drive. If it is, mount a tape and DB2 will process the offload command.

If you are uncertain about what is causing the problem, enter the following command:

```
-ARCHIVE LOG CANCEL OFFLOAD
```

This command causes DB2 to restart the offload task. This might solve the problem.

If this command does not solve the problem, you must determine the cause of the problem and then reissue the command again. If the problem cannot be solved quickly, have the system programmer define additional active logs.

System programmer action: Additional active log data sets can permit DB2 to continue its normal operation while the problem causing the offload failures is corrected.

1. Use the z/OS command CANCEL command to bring DB2 down.
2. Use the access method services DEFINE command to define new active log data sets. Run utility DSNJLOGF to initialize the new active log data sets.

   To minimize the number of offloads taken per day in your installation, consider increasing the size of the active log data sets.
3. Define the new active log data sets in the BSDS by using the change log inventory utility (DSNJU003). For additional details, see Part 3 of *DB2 Utility Guide and Reference*.

4. Restart DB2. Offload is started automatically during startup, and restart processing occurs.

Problem 2 - Write I/O error on active log data set

Symptom: The following message appears:

```
DSNJ105I - csect-name LOG WRITE ERROR DSNAME=..., LOGRBA=...,
           ERROR STATUS=ccccffss
```

System action:

- Marks the failing log data set TRUNCATED in the BSDS.
- Goes on to the next available data set.
- If dual active logging is used, truncates the other copy at the same point.
- The data in the truncated data set is offloaded later, as usual.
- The data set is not stopped. It is reused on the next cycle. However, if there is a DSNJ104 message indicating that there is a CATUPDT failure, then the data set is marked "stopped".

System programmer action: If you get the DSNJ104 message indicating CATUPDT failure, you must use access method services and the change log inventory utility (DSNJU003) to add a replacement data set. This requires that you bring DB2 down. When you do this depends on how widespread the problem is.

- If the problem is localized and does not affect your ability to recover from any further problems, you can wait until the earliest convenient time.
- If the problem is widespread (perhaps affecting an entire set of active log data sets), take DB2 down after the next offload.

For instructions on using the change log inventory utility, see Part 3 of *DB2 Utility Guide and Reference*.

Problem 3 - Dual logging is lost

Symptom: The following message appears:

```
DSNJ004I - ACTIVE LOG COPYn INACTIVE, LOG IN SINGLE MODE,
           ENDRBA=...
```

Having completed one active log data set, DB2 found that the subsequent (COPY n) data sets were not offloaded or were marked "stopped".

System action: Continues in single mode until offloading completes, then returns to dual mode. If the data set is marked "stopped", however, then intervention is required.

System programmer action: Check that offload is proceeding and is not waiting for a tape mount. It might be necessary to run the print log map utility to determine the status of all data sets.

If there are "stopped" data sets, you must use IDCAMS to delete the data sets, and then re-add them using the change log inventory utility (DSNJU003). See Part 3 of *DB2 Utility Guide and Reference* for information about using the change log inventory utility.

Problem 4 - I/O errors while reading the active log

Symptom: The following message appears:

```
DSNJ106I - LOG READ ERROR DSNAME=..., LOGRBA=...,
           ERROR STATUS=ccccffss
```

System action:

- If the error occurs during offload, offload tries to pick the RBA range from a second copy.
  - If no second copy exists, the data set is stopped.
  - If the second copy also has an error, only the original data set that triggered the offload is stopped. Then the archive log data set is terminated, leaving a discontinuity in the archived log RBA range.
  - The following message is issued.

    ```
    DSNJ124I - OFFLOAD OF ACTIVE LOG SUSPENDED FROM RBA xxxxxx
               TO RBA xxxxxx DUE TO I/O ERROR
    ```

  - If the second copy is satisfactory, the first copy is not stopped.
- If the error occurs during recovery, DB2 provides data from specific log RBAs requested from another copy or archive. If this is unsuccessful, recovery fails and the transaction cannot complete, but no log data sets are stopped. However, the table space being recovered is not accessible.

System programmer action: If the problem occurred during offload, determine which databases are affected by the active log problem and take image copies of those. Then proceed with a new log data set.

Also, you can use IDCAMS REPRO to archive as much of the stopped active log data set as possible. Then run the change log inventory utility to notify the BSDS of the new archive log and its log RBA range. Repairing the active log does not solve the problem, because offload does not go back to unload it.

If the active log data set has been stopped, it is not used for logging. The data set is not deallocated; it is still used for reading.

If the data set is not stopped, an active log data set should nevertheless be replaced if persistent errors occur. The operator is not told explicitly whether the data set has been stopped. To determine the status of the active log data set, run the print log map utility (DSNJU004). For more information about the print log map utility, see Part 3 of *DB2 Utility Guide and Reference*.

To replace the data set, take the following steps:

1. Be sure the data is saved.

   If you have dual active logs, the data is saved on the other active log and it becomes your *new data set*. Skip to step 4 on page 535.

   If you have not been using dual active logs, take the following steps to determine whether the data set with the error has been offloaded:

   a. Use the print log map to list information about the archive log data sets from the BSDS.
   b. Search the list for a data set whose RBA range includes the range of the data set with the error.
2. If the data set with the error has been offloaded (that is, if the value for High RBA offloaded in the print log map output is greater than the RBA range of the data set with the error), you need to manually add a new archive log to the BSDS using the change log inventory utility (DSNJU003). Use IDCAMS to define a new log having the same LRECL and BLKSIZE values as that defined in DSNZP*xxx*. You can use the access method services REPRO command to copy a data set with the error to the new archive log. If the archive log is not cataloged, DB2 can locate it from the UNIT and VOLSER values in the BSDS.
3. If an active log data set has been stopped, an RBA range has not been offloaded; copy from the data set with the error to a new data set. If additional

I/O errors prevent you from copying the entire data set, a gap occurs in the log and restart might fail, though the data still exists and is not overlaid. If this occurs, see Chapter 23, "Recovery from BSDS or log failure during restart," on page 591.

4. Stop DB2, and use change log inventory to update information in the BSDS about the data set with the error.
   a. Use DELETE to remove information about the bad data set.
   b. Use NEWLOG to name the new data set as the new active log data set and to give it the RBA range that was successfully copied.

      The DELETE and NEWLOG operations can be performed by the same job step; put the DELETE statement before the NEWLOG statement in the SYSIN input data set. This step will clear the stopped status and DB2 will eventually archive it.
5. Delete the data set in error by using access method services.
6. Redefine the data set so you can write to it. Use access method services DEFINE command to define the active log data sets. Run utility DSNJLOGF to initialize the active log data sets. If using dual logs, use access method services REPRO to copy the good log into the redefined data set so that you have two consistent, correct logs again.

Archive log failure recovery

This section covers some of the more likely archive log problems. Problems not covered here include archive log open/close problems that are indicated by the message DSNJ104I. Most archive log problems are described in Chapter 23, "Recovery from BSDS or log failure during restart," on page 591.

Problem 1 - Allocation problems

Symptom: The following message appears:

```
DSNJ103I - csect-name LOG ALLOCATION ERROR DSNAME=dsname,
ERROR STATUS=eeeeiiii, SMS REASON CODE=ssssssss
```

z/OS dynamic allocation provides the ERROR STATUS. If the allocation was for offload processing, the message is also displayed:

```
DSNJ115I - OFFLOAD FAILED, COULD NOT ALLOCATE AN ARCHIVE DATA SET
```

System action: One of the system actions occurs:

- The RECOVER utility is executing and requires an archive log. If neither log can be found or used, recovery fails.
- The active log became full and an offload was scheduled. Offload tries again the next time it is triggered. The active log does not wrap around; therefore, if there are no more active logs, data is not going to be lost.
- The input is needed for restart, which fails; refer to Chapter 23, "Recovery from BSDS or log failure during restart," on page 591.

Operator action: Check the allocation error code for the cause of the problem and correct it. Ensure that drives are available and run the recovery job again. Caution must be exercised if a DFSMSdfp ACS user-exit filter has been written for an archive log data set, because this can cause the DB2 subsystem to fail on a device allocation error attempting to read the archive log data set.

Problem 2 - Write I/O errors during archive log offload

Symptom: No specific DB2 message is issued for write I/O errors. Only a z/OS error recovery program message appears. If you get DB2 message DSNJ128I, the offload task has abended and you should consult Part 2 of *DB2 Messages*.

System action:

- Offload abandons that output data set (no entry in BSDS).
- Offload dynamically allocates a new archive and restarts offloading from the point at which it was previously triggered. If there is dual archiving, the second copy waits.
- If an error occurs on the new data set, the additional actions occur:
  - If in dual archive mode, message DSNJ114I is generated and the offload processing changes to single mode.

    ```
    DSNJ114I - ERROR ON ARCHIVE DATA SET, OFFLOAD CONTINUING
               WITH ONLY ONE ARCHIVE DATA SET BEING GENERATED
    ```
  - If in single mode, it abandons the output data set. Another attempt to offload this RBA range is made the next time offload is triggered.
  - The active log does not wrap around; if there are no more active logs, data is not lost.

Operator action: Ensure that offload is allocated on a good drive and control unit.

Problem 3 - Read I/O errors on archive data set during recover

Symptom: No specific DB2 message is issued, only the z/OS error recovery program message appears.

System action:

- If a second copy exists, it is allocated and used.
- If a second copy does not exist, recovery fails.

Operator action: If you are recovering from tape, try recovering using a different drive. If this does not work, contact the system programmer.

System programmer action: The only option is to recover to the last image copy or the last quiesce point RBA. See Part 2 of *DB2 Utility Guide and Reference* for more information about using the RECOVER utility.

Problem 4 - Insufficient disk space for offload processing

Symptom: While offloading the active log data sets to disk, DB2 offload processing terminates unexpectedly. DB2 does not issue any specific message other than:

```
DSNJ128I - LOG OFFLOAD TASK FAILED FOR ACTIVE LOG nnnnn
```

The failure is preceded by z/OS ABEND messages IEC030I, IEC031I, or IEC032I.

System action: DB2 deallocates the data set on which the error occurred. If in dual archive mode, DB2 changes to single archive mode and continues the offload. If the offload cannot compete in single archive mode, the active log data sets cannot be offloaded, and the status of the active log data sets remains NOTREUSEABLE. Another attempt to offload the RBA range of the active log data sets is made the next time offload is invoked.

System programmer action: If DB2 is operating with restricted active log resources (see message DSNJ110E), quiesce the DB2 subsystem to restrict logging activity until the z/OS ABEND is resolved.

This message is generated for a variety of reasons. When accompanied by the preceding z/OS abends, the most likely failures are as follows:

- The size of the archive log data set is too small to contain the data from the active log data sets during offload processing. All secondary space allocations have been used. This condition is normally accompanied by z/OS ABEND message IEC030I.

  To solve the problem, increase the primary or secondary allocations (or both) for the archive log data set in DSNZP*xxx*. Another option is to reduce the size of the active log data set. If the data to be offloaded is particularly large, you can mount another online storage volume or make one available to DB2. Modifications to DSNZP*xxx* require that you stop and start DB2 to take effect.
- All available space on the disk volumes to which the archive data set is being written has been exhausted. This condition is normally accompanied by z/OS ABEND message IEC032I.

  To solve the problem, make space available on the disk volumes, or make available another online storage volume for DB2. Then issue the DB2 command ARCHIVE LOG CANCEL OFFLOAD to get DB2 to retry the offload.
- The primary space allocation for the archive log data set (as specified in the load module for subsystem parameters) is too large to allocate to any available online disk device. This condition is normally accompanied by z/OS ABEND message IEC032I.

  To solve the problem, make space available on the disk volumes, or make available another online storage volume for DB2. If this is not possible, an adjustment to the value of PRIQTY in the DSNZP*xxx* module is required to reduce the primary allocation. (For instructions, see Part 2 of *DB2 Installation Guide*. If the primary allocation is reduced, the size of the secondary space allocation might have to be increased to avoid future IEC030I abends.

Temporary resource failure recovery

DB2 sometimes experiences a temporary problem when it accesses log data sets. For example, DB2 might experience a problem when it attempts to allocate or open archive log data sets during the rollback of a long-running unit of recovery. These temporary failures can be caused by:

- A temporary problem with DFHSM recall
- A temporary problem with the tape subsystem
- Uncataloged archive logs
- Archive tape mount requests being cancelled

In these cases, DB2 issues messages for the access failure for each log data set. These messages provide information needed to resolve the access error. For example:

```
DSNJ104I  ( DSNJR206 RECEIVED ERROR STATUS 00000004
          FROM DSNPCLOC FOR DSNAME=DSNC710.ARCHLOG1.A0000049

DSNJ104I  ( DSNJR206 RECEIVED ERROR STATUS 00000004
          FROM DSNPCLOC FOR DSNAME=DSNC710.ARCHLOG2.A0000049

*DSNJ153E ( DSNJR006 CRITICAL LOG READ ERROR
          CONNECTION-ID = TEST0001
          CORRELATION-ID = CTHDCORID001
          LUWID = V71A.SYEC1DB2.B3943707629D=10
          REASON-CODE = 00D10345
```

You can attempt to recover from temporary failures by issuing a positive reply to message:

```
*26 DSNJ154I  ( DSNJR126 REPLY Y TO RETRY LOG READ REQUEST, N TO ABEND
```

If the problem persists, quiesce other work in the system before replying **N**, which terminates DB2.

BSDS failure recovery

For information about the BSDS, see "Managing the bootstrap data set (BSDS)" on page 436. Normally, there are two copies of the BSDS; but if one is damaged, DB2 immediately falls into single BSDS mode processing. The damaged copy of the BSDS must be recovered prior to the next restart. If you are in single mode and damage the only copy of the BSDS, or if you are in dual mode and damage both copies, DB2 stops until the BSDS is recovered. To proceed under these conditions, see "Recovering the BSDS from a backup copy" on page 539.

This section covers some of the BSDS problems that can occur. Problems not covered here include:
- RECOVER BSDS command failure (messages DSNJ301I through DSNJ307I)
- Change log inventory utility failure (message DSNJ123E)
- Errors in the BSDS backup being dumped by offload (message DSNJ125I).

See Part 2 of *DB2 Messages* for information about those problems.

Problem 1 - An I/O error occurs

Symptom: The message appears:

```
DSNJ126I - BSDS ERROR FORCED SINGLE BSDS MODE
```

It is followed by one of these messages:

```
DSNJ107I - READ ERROR ON BSDS
           DSNAME=... ERROR STATUS=...

DSNJ108I - WRITE  ERROR ON BSDS
           DSNAME=... ERROR STATUS=...
```

System action: The BSDS mode changes from dual to single.

System programmer action:

1. Use access method services to rename or delete the damaged BSDS and to define a new BSDS with the same name as the failing BSDS. Control statements can be found in job DSNTIJIN.
2. Issue the DB2 command RECOVER BSDS to make a copy of the good BSDS in the newly allocated data set and to reinstate dual BSDS mode.

Problem 2 - An error occurs while opening

Symptom: The message appears:

```
DSNJ100I - ERROR OPENING BSDSn DSNAME=..., ERROR STATUS=eeii
```

The error status is VSAM return code/feedback. For information about VSAM codes, refer to *z/OS DFSMS: Macro Instructions for Data Sets*.

System action: None.

System programmer action:

1. Use access method services to delete or rename the damaged data set, to define a replacement data set, and to copy the remaining BSDS to the replacement with the REPRO command.
2. Use the command START DB2 to start the DB2 subsystem.

Problem 3 - Unequal timestamps exist

Symptom: The following message appears:

```
DSNJ120I - DUAL BSDS DATA SETS HAVE UNEQUAL TIMESTAMPS,
           BSDS1 SYSTEM=..., UTILITY=..., BSDS2 SYSTEM=..., UTILITY=...
```

Unequal timestamps can occur for the following reasons:

- One of the volumes containing the BSDS has been restored. All information of the restored volume is down-level. If the volume contains any active log data sets or DB2 data, their contents are also down-level. The down-level volume has the lower timestamp.

  For information about resolving this problem, see "Failure during a log RBA read request" on page 609.
- Dual BSDS mode has degraded to single BSDS mode, and you are trying to start without recovering the bad BSDS.
- The DB2 subsystem abended after updating one copy of the BSDS, but prior to updating the second copy.

System action: DB2 attempts to resynchronize the BSDS data sets and restore dual BSDS mode. If this attempt succeeds, DB2 restart continues automatically.

Operator action: If DB2 restart fails, notify the system programmer.

System programmer action: If DB2 fails to automatically resynchronize the BSDS data sets, perform the following procedure:

1. Run the print log map utility (DSNJU004) on both copies of the BSDS; compare the lists to determine which copy is accurate or current.
2. Rename the down-level data set and define a replacement for it.
3. Copy the good data set to the replacement data set, using the REPRO command of access method services.
4. Use access method services REPRO to copy the current version of the active log to the down-level data set if all the following condtitions are true:
   - The problem was caused by a restored down-level BSDS volume.
   - The restored volume contains active log data.
   - You were using dual active logs on separate volumes.

   If you were not using dual active logs, you must cold start the subsystem. (For this procedure, see "Failure resulting from total or excessive loss of log data" on page 613).

   If the restored volume contains database data, use the RECOVER utility to recover that data after successful restart.

Recovering the BSDS from a backup copy

If DB2 is operating in single BSDS mode and the BSDS is damaged, or if DB2 is operating in dual BSDS mode and both BSDSs are damaged, DB2 stops and does not restart until dual BSDS is restored. In this case, take the following steps:

1. Locate the BSDS associated with the most recent archive log data set. The data set name of the most recent archive log appears on the z/OS console in the last occurrence of message DSNJ003I, which indicates that offloading has successfully completed. In preparation for the rest of this procedure, it is a good practice to keep a log of all successful archives noted by that message.
   - If archive logs are on disk, the BSDS is allocated on any available disk. The BSDS name is like the corresponding archive log data set name; change only the first letter of the last qualifier, from A to B, as in the following example:

Archive log name
DSN.ARCHLOG1.**A**0000001

BSDS copy name
DSN.ARCHLOG1.**B**0000001

- If archive logs are on tape, the BSDS is the first data set of the first archive log volume. The BSDS is not repeated on later volumes.

2. If the most recent archive log data set has no copy of the BSDS (presumably because an error occurred when offloading it), then locate an earlier copy of the BSDS from an earlier offload.
3. Rename any *damaged* BSDS by using the access method services ALTER command with the NEWNAME option. If the decision is made to delete any damaged BSDS, use the access method services DELETE command. For each damaged BSDS, use access method services to define a new BSDS as a replacement data set. Job DSNTIJIN contains access method services control statements to define a new BSDS.

   The BSDS is a VSAM key-sequenced data set that has three components: cluster, index, and data. You must rename *all* components of the data set. Avoid changing the high-level qualifier. See *z/OS DFSMS Access Method Services for Catalogs* for detailed information about using the access method services ALTER command.
4. Use the access method services REPRO command to copy the BSDS from the archive log to one of the replacement BSDSs you defined in step 3. Do not copy any data to the second replacement BSDS; data is placed in the second replacement BSDS in a later step in this procedure.
   a. Print the contents of the replacement BSDS.

      Use the print log map utility (DSNJU004) to print the contents of the replacement BSDS. This enables you to review the contents of the replacement BSDS before continuing your recovery work.
   b. Update the archive log data set inventory in the replacement BSDS.

      Examine the print log map output and note that the replacement BSDS does not obtain a record of the archive log from which the BSDS was copied. If the replacement BSDS is a particularly old copy, it is missing all archive log data sets that were created later than the BSDS backup copy. Thus, the BSDS inventory of the archive log data sets must be updated to reflect the current subsystem inventory.

      Use the change log inventory utility (DSNJU003) NEWLOG statement to update the replacement BSDS, adding a record of the archive log from which the BSDS was copied. Make certain the CATALOG option of the NEWLOG statement is properly set to CATALOG = YES if the archive log data set is cataloged. Also, use the NEWLOG statement to add any additional archive log data sets that were created later than the BSDS copy.
   c. Update DDF information in the replacement BSDS.

      If your installation's DB2 is part of a distributed network, the BSDS contains the DDF control record. You must review the contents of this record in the output of the print log map utility. If changes are required, use the change log inventory DDF statement to update the BSDS DDF record.
   d. Update the active log data set inventory in the replacement BSDS.

      In unusual circumstances, your installation could have added, deleted, or renamed active log data sets since the BSDS was copied. In this case, the replacement BSDS does not reflect the actual number or names of the active log data sets your installation has currently in use.

If you must delete an active log data set from the replacement BSDS log inventory, use the change log inventory utility DELETE statement.

If you need to add an active log data set to the replacement BSDS log inventory, use the change log inventory utility NEWLOG statement. Be certain that the RBA range is specified correctly on the NEWLOG statement.

If you must rename an active log data set in the replacement BSDS log inventory, use the change log inventory utility DELETE statement, followed by the NEWLOG statement. Be certain that the RBA range is specified correctly on the NEWLOG statement.

e. Update the active log RBA ranges in the replacement BSDS.

   Later, when a restart is performed, DB2 compares the RBAs of the active log data sets listed in the BSDS with the RBAs found in the actual active log data sets. If the RBAs do not agree, DB2 does not restart. The problem is magnified when a particularly old copy of the BSDS is used. To resolve this problem, you can use the change log inventory utility to change the RBAs found in the BSDS to the RBAs in the actual active log data sets. Use the following procedure to change RBAs in the BSDS:

   - If you are not certain of the RBA range of a particular active log data set, use DSN1LOGP to print the contents of the active log data set. Obtain the logical starting and ending RBA values for the active log data set from the DSN1LOGP output. The STARTRBA value you use in the change log inventory utility must be at the beginning of a control interval. Similarly, the ENDRBA value you use must be at the end of a control interval. To get these values, round the starting RBA value from the DSN1LOGP output down so that it ends in X'000'. Round the ending RBA value up so that it ends in X'FFF'.
   - When the RBAs of all active log data sets are known, compare the actual RBA ranges with the RBA ranges found in the BSDS (listed in the print log map utility output).

     If the RBA ranges are equal for all active log data sets, you can proceed to the next recovery step without any additional work.

     If the RBA ranges are not equal, then the values in the BSDS must be adjusted to reflect the actual values. For each active log data set that needs to have the RBA range adjusted, use the change log inventory utility DELETE statement to delete the active log data set from the inventory in the replacement BSDS. Then use the NEWLOG statement to redefine the active log data set to the BSDS.

f. If only two active log data sets are specified in the replacement BSDS, add a new active log data set for each copy of the active log and define each new active log data set of the replacement BSDS log inventory.

   If only two active log data sets are specified for each copy of the active log, DB2 can have difficulty during restart. The difficulty can arise when one of the active log data sets is full and has not been offloaded, while the second active log data set is close to filling. Adding a new active log data set for each copy of the active log can alleviate difficulties on restart in this scenario.

   To add a new active log data set for each copy of the active log, use the access method services DEFINE command to define a new active log data set for each copy of the active log. The control statements to accomplish this task can be found in job DSNTIJIN. Once the active log data sets are physically defined and allocated, use the change log inventory utility

NEWLOG statement to define the new active log data sets of the replacement BSDS. The RBA ranges need not be specified on the NEWLOG statement.

5. Copy the updated BSDS copy to the second new BSDS data set. The dual bootstrap data sets are now identical.

   You should consider using the print log map utility (DSNJU004) to print the contents of the second replacement BSDS at this point.
6. See Chapter 23, "Recovery from BSDS or log failure during restart," on page 591 for information about what to do if you have lost your current active log data set. For a discussion of how to construct a conditional restart record, see "Step 4: Truncate the log at the point of error" on page 601.
7. Restart DB2, using the newly constructed BSDS. DB2 determines the current RBA and what active logs need to be archived.

DB2 database failures

Problem: Allocation or open problems occur.

Symptom 1: The following message indicates an allocation problem:

```
DSNB207I - DYNAMIC ALLOCATION OF DATA SET FAILED.
           REASON=rrrr DSNAME=dsn
```

where *rrrr* is an z/OS dynamic allocation reason code. For information about these reason codes, see *z/OS MVS Programming: Authorized Assembler Services Guide*.

Symptom 2: The following messages indicate a problem at open:

```
IEC161I  rc[(sfi)] - ccc, iii,  sss, ddn,
         ddd, ser, xxx, dsn, cat
```

where:

rc Is a return code
sfi Is subfunction information (*sfi* only appears with certain return codes)
ccc Is a function code
iii Is a job name
sss Is a step name
ddn Is a *ddname*
ddd Is a device number (if the error is related to a specific device)
ser Is a volume serial number (if the error is related to a specific volume)
xxx Is a VSAM cluster name
dsn Is a data set name
cat Is a catalog name.

For information about these codes, see *z/OS MVS System Messages Volumes 1-10*.

```
DSNB204I - OPEN OF DATA SET FAILED. DSNAME = dsn
```

System action:
- The table space is automatically stopped.
- Programs receive an -904 SQLCODE (SQLSTATE '57011').
- If the problem occurs during restart, the table space is marked for deferred restart, and restart continues. The changes are applied later when the table space is started.

System programmer action: None.

Operator action:

1. Check reason codes and correct.
2. Ensure that drives are available for allocation.
3. Enter the command START DATABASE.

Recovery from down-level page sets

When using a stand-alone or non-DB2 utility, such as DSN1COPY or DFSMShsm, you might inadvertently replace a DB2 data set with an incorrect or outdated copy. Such a copy is called *down-level*; using it can cause complex problems.

Other reasons for a down-level condition are:

- A cold start of DB2 was performed.
- The VSAM high-used RBA of a table space has become corrupted.

DB2 associates a level ID with every page set or partition. Most operations detect a down-level ID, and return an error condition, when the page set or partition is first opened for mainline or restart processing. The exceptions are operations that do not use the data:

LOAD REPLACE
RECOVER
REBUILD INDEX
DSN1COPY
DSN1PRNT

The RESET option of DSN1COPY resets the level ID on its output to a neutral value that passes any level check. Hence, you can still use DSN1COPY to move data from one system or table space to another.

The LEVELID option of the REPAIR utility marks a down-level table space or index as current. See *DB2 Utility Guide and Reference* for details on using REPAIR.

For directory page sets, and the page sets for SYSIBM.SYSCOPY and SYSIBM.SYSGROUP, a down-level ID is detected only at restart and not during mainline operations.

Symptom: The following message appears:

```
DSNB232I csect-name - UNEXPECTED DATA SET LEVEL ID ENCOUNTERED
```

The message contains also the level ID of the data set, the level ID that DB2 expects, and the name of the data set.

System action:

- If the error was reported during mainline processing, DB2 sends back a "resource unavailable" SQLCODE to the application and a reason code explaining the error.
- If the error was detected while a utility was processing, the utility gives a return code 8.

System programmer action: You can recover by using any of the following methods:

If the message occurs during restart:

- Replace the data set with one at the proper level, using DSN1COPY, DFSMShsm, or some equivalent method. To check the level ID of the new data set, run the

stand-alone utility DSN1PRNT on it, with the options PRINT(0) (to print only the header page) and FORMAT. The formatted print identifies the level ID.

- Recover the data set to the current time, or to a prior time, using the RECOVER utility.
- Replace the contents of the data set, using LOAD REPLACE.

If the message occurs during normal operation, use any of the preceding methods in addition to one of the following action:

- Accept the down-level data set by changing its level ID.

  The REPAIR utility contains a statement for that purpose. Run a utility job with the statement REPAIR LEVELID. The LEVELID statement cannot be used in the same job step with any other REPAIR statement.

> **Important**
> If you accept a down-level data set or disable down-level detection, your data might be inconsistent.

For more information about using the utilities, see *DB2 Utility Guide and Reference*.

You can control down-level detection. Use the LEVELID UPDATE FREQ field of panel DSNTIPL to either disable down-level detection or control how often the level ID of a page set or partition is updated. DB2 accepts any value between 0 and 32767.

To disable down-level detection, specify 0 in the LEVELID UPDATE FREQ field of panel DSNTIPL.

To control how often level ID updates are taken, specify a value between 1 and 32767. See Part 2 of *DB2 Installation Guide* for more information about choosing the frequency of level ID updates.

Procedure for recovering invalid LOBs

Unless your LOBs are fairly small, specifying LOG NO for LOB objects is recommended. The performance cost of logging exceeds the benefits you can receive from logging such large amounts of data. If no changes are made to LOB data, this is not an issue. However, you should make image copies of the LOB table space to prepare for failures. The frequency with which you make image copies is based on how often you update LOB data.

If you need to recover LOB data that changed after your last image copy, follow this procedure:

1. Run the RECOVER utility as you do for other table spaces:

   ```
   RECOVER TABLESPACE dbname.lobts
   ```

   If changes were made after the image copy, DB2 puts the table space in *Aux Warning* status. The purpose of this status is let you know that some of your LOBs are invalid. Applications that try to retrieve the values of those LOBs will receive SQLCODE -904. Applications can still access other LOBs in the LOB table space.
2. Get a report of the invalid LOBs by running CHECK LOB on the LOB table space:

```
CHECK LOB TABLESPACE dbname.lobts
```

DB2 generates the following messages:

```
LOB WITH ROWID = 'xxxxxxx' VERSION = n  IS INVALID
```

3. Fix the invalid LOBs, by updating the LOBs or setting them to the null value. For example, suppose you determine from the CHECK LOB utility that the row of the EMP_PHOTO_RESUME table with ROWID X'C1BDC4652940D40A81C201AA0A28' has an invalid value for column RESUME. If host variable hvlob contains the correct value for RESUME, you can use this statement to correct the value:

```
UPDATE DSN8810.EMP_PHOTO_RESUME
  SET RESUME = :hvlob
  WHERE EMP_ROWID = ROWID(X'C1BDC4652940D40A81C201AA0A28');
```

Table space input/output error recovery

Problem: A table space failed.

Symptom: The following message is issued:

```
DSNU086I  DSNUCDA1 READ I/O ERRORS ON SPACE= dddddddd.
          DATA SET NUMBER= nnn.
          I/O ERROR PAGE RANGE= aaaaaa, bbbbbb.
```

where *dddddddd* is a table space name.

Any table spaces identified in DSNU086I messages must be recovered using one of the procedures in this section listed under “Operator Action”.

System action: DB2 remains active.

Operator action: Fix the error range.

1. Use the command STOP DATABASE to stop the failing table space.
2. Use the command START DATABASE ACCESS (UT) to start the table space for utility-only access.
3. Start a RECOVER utility step to recover the error range by using the DB2 RECOVER (*dddddddd*) ERROR RANGE statement.

   If you receive message DSNU086I again, indicating the error range recovery cannot be performed, use the recovery procedure that follows this procedure.
4. Issue the command START DATABASE to start the table space for RO or RW access, whichever is appropriate. If the table space is recovered, you do not need to continue with the following procedure.

If error range recovery fails: If the error range recovery of the table space failed because of a hardware problem, proceed as follows:

1. Use the command STOP DATABASE to stop the table space or table space partition that contains the error range. This causes all the in-storage data buffers associated with the data set to be externalized to ensure data consistency during the subsequent steps.
2. Use the INSPECT function of the IBM Device Support Facility, ICKDSF, to check for track defects and to assign alternate tracks as necessary. The physical location of the defects can be determined by analyzing the output of messages DSNB224I, DSNU086I, IOS000I, which were displayed on the system operator’s console at the time the error range was created. If damaged storage media is

suspected, then request assistance from hardware support personnel before proceeding. Refer to *Device Support Facilities User's Guide and Reference* for information about using ICKDSF.

3. Use the command START DATABASE to start the table space with ACCESS(UT) or ACCESS(RW).
4. Run the utility RECOVER ERROR RANGE that, from image copies, locates, allocates, and applies the pages within the tracks affected by the error ranges.

DB2 catalog or directory input/output errors

Problem: The DB2 catalog or directory failed.

Symptom: The following message is issued:

```
DSNU086I  DSNUCDA1 READ I/O ERRORS ON SPACE= dddddddd.
          DATA SET NUMBER= nnn.
          I/O ERROR PAGE RANGE= aaaaaa, bbbbbb.
```

where *dddddddd* is a table space name from the catalog or directory. *dddddddd* is the table space that failed (for example, SYSCOPY, abbreviation for SYSIBM.SYSCOPY, or SYSLGRNX, abbreviation for DSNDB01.SYSLGRNX). This message can indicate either read or write errors. You can also get a DSNB224I or DSNB225I message, which could indicate an input or output error for the catalog or directory.

Any catalog or directory table spaces that are identified in DSNU086I messages must be recovered with this procedure.

System action: DB2 remains active.

If the DB2 directory or any catalog table is damaged, only user IDs with the RECOVERDB privilege in DSNDB06, or an authority that includes that privilege, can do the recovery. Furthermore, until the recovery takes place, only those IDs can do anything with the subsystem. If an ID without proper authorization attempts to recover the catalog or directory, message DSNU060I is displayed. If the authorization tables are unavailable, message DSNT500I is displayed indicating the resource is unavailable.

System programmer action: None.

Operator action: Take the following steps for each table space in the DB2 catalog and directory that has failed. If there is more than one, refer to the description of RECOVER in Part 2 of *DB2 Utility Guide and Reference* for more information about the specific order of recovery.

1. Stop the failing table spaces.
2. Determine the name of the data set that failed. There are two ways to do this:
   - Check *prefix*.SDSNSAMP (DSNTIJIN), which contains the JCL for installing DB2. Find the fully qualified name of the data set that failed by searching for the name of the table space that failed (the one identified in the message as SPACE = *dddddddd*).
   - Construct the data set name by performing one of the following actions:
     – If the table space is in the DB2 catalog, the data set name format is:

       `DSNC810.DSNDBC.DSNDB06.dddddddd.I0001.A001`

       where *dddddddd* is the name of the table space that failed.
     – If the table space is in the DB2 directory, the data set name format is:

DSNC810.DSNDBC.DSNDB01.*dddddddd*.I0001.A001

where *dddddddd* is the name of the table space that failed.

If you do not use the default (IBM-supplied) formats, the formats for data set names can be different.

3. Use access method services DELETE to delete the data set, specifying the fully qualified data set name.
4. After the data set has been deleted, use access method services DEFINE to redefine the same data set, again specifying the same fully qualified data set name. Use the JCL for installing DB2 to determine the appropriate parameters.

   Important: The REUSE parameter must be coded in the DEFINE statements.
5. Issue the command START DATABASE ACCESS(UT), naming the table space involved.
6. Use the RECOVER utility to recover the table space that failed.
7. Issue the command START DATABASE, specifying the table space name and RO or RW access, whichever is appropriate.

Integrated catalog facility failure recovery

This section includes information regarding volume data set failures. The following topics are described:

"Recovery when the VSAM volume data set (VVDS) is destroyed"
"Out of disk space or extent limit recovery" on page 548

Recovery when the VSAM volume data set (VVDS) is destroyed

Problem: A VSAM volume data set (VVDS) is either out of space or destroyed.

Symptom: DB2 sends the following message to the master console:

```
DSNP012I - DSNPSCT0 - ERROR IN VSAM CATALOG LOCATE FUNCTION
           FOR data_set_name
           CTLGRC=50
           CTLGRSN=zzzzRRRR
           CONNECTION-ID=xxxxxxxx,
           CORRELATION-ID=yyyyyyyyyyyy
           LUW-ID=logical-unit-of-work-id=token
```

For a detailed explanation of this message, see Part 2 of *DB2 Messages*.

VSAM can also issue the following message:

```
IDC3009I VSAM CATALOG RETURN CODE IS 50, REASON CODE IS
         IGGOCLaa - yy
```

In this VSAM message, *yy* is 28, 30, or 32 for an out-of-space condition. Any other values for *yy* indicate a damaged VVDS.

System action: Your program is terminated abnormally and one or more messages are issued.

System programmer action: None.

Operator action: For information about recovering the VVDS, consult *z/OS DFSMS: Managing Catalogs*.

The procedures given in these books describe three basic recovery scenarios. First determine which scenario exists for the specific VVDS in error. Then, before beginning the appropriate procedure, take the following steps:

1. Determine the names of all table spaces residing on the same volume as the VVDS. To determine the table space names, look at the VTOC entries list for that volume, which indicates the names of all the data sets on that volume. For information about how to determine the table space name from the data set name, refer to Part 2, "Designing a database: advanced topics," on page 23.
2. Use the DB2 COPY utility to take image copies of all table spaces of the volume. Taking image copies minimizes reliance on the DB2 recovery log and can speed up the processing of the DB2 RECOVER utility (to be mentioned in a subsequent step).

   If the COPY utility cannot be used, continue with this procedure. Be aware that processing time increases because more information is obtained from the DB2 recovery log.
3. Use the command STOP DATABASE for all the table spaces that reside on the volume, or use the command STOP DB2 to stop the entire DB2 subsystem if an unusually large number or critical set of table spaces are involved.
4. If possible, use access method services to export all non-DB2 data sets residing on that volume. For more information, see *DFSMS/MVS: Access Method Services for the Integrated Catalog* and *z/OS DFSMS: Managing Catalogs*.
5. To recover all non-DB2 data sets on the volume, see *DFSMS/MVS: Access Method Services for the Integrated Catalog* and *z/OS DFSMS: Managing Catalogs*.
6. Use access method services DELETE and DEFINE commands to delete and redefine the data sets for all user-defined table spaces and DB2-defined data sets when the physical data set has been destroyed. DB2 automatically deletes and redefines all other STOGROUP defined table spaces.

   You do not need to do this for those table spaces that are STOGROUP defined; DB2 takes care of them automatically.
7. Issue the DB2 START DATABASE command to restart all the table spaces stopped in step 3. If the entire DB2 subsystem was stopped, issue the START DB2 command.
8. Use the DB2 RECOVER utility to recover any table spaces and indexes. For information about recovering table spaces, refer to Chapter 21, "Backing up and recovering databases," on page 471.

Out of disk space or extent limit recovery

Problem: There is no more space on the volume on which the data set is stored or the data set might have reached its maximum DB2 size or its maximum number of VSAM extents.

Symptom: One of the following messages:

1. Extend request failure

   When an insert or update requires additional space, but the space is not available in the current table or index space, DB2 issues the following message:

   ```
   DSNP007I - DSNPmmmm - EXTEND FAILED FOR
              data-set-name.  RC=rrrrrrrr
              CONNECTION-ID=xxxxxxxx,
              CORRELATION-ID=yyyyyyyyyyyy
              LUWID-ID=logical-unit-of-work-id=token
   ```
2. Look ahead warning

A look ahead warning occurs when there is enough space for a few inserts and updates, but the index space or table space is almost full. On an insert or update at the end of a page set, DB2 determines whether the data set has enough available space. DB2 uses the following values in this space calculation:

- The primary space quantity from the integrated catalog facility (ICF) catalog
- The secondary space quantity from the ICF catalog
- The allocation unit size

If enough space does not exist, DB2 tries to extend the data set. If the extend request fails, then DB2 issues the following message:

```
DSNP001I - DSNPmmmm - data-set-name IS WITHIN
           nK BYTES OF AVAILABLE SPACE.
           RC=rrrrrrrr
           CONNECTION-ID=xxxxxxxx,
           CORRELATION-ID=yyyyyyyyyyyy
           LUW-ID=logical-unit-of-work-id=token
```

System action: For a demand request failure during restart, the object supported by the data set (an index space or a table space) is stopped with deferred restart pending. Otherwise, the state of the object remains unchanged. Programs receive a -904 SQL return code (SQLSTATE '57011').

System programmer action: None.

Operator action: The appropriate choice of action depends on particular circumstances. The following topics are described in this section; the text following the list describes how to choose which action to take:

- "Procedure 1. Extend a data set" on page 550
- "Procedure 2. Enlarge a fully extended data set (user-managed)" on page 550
- "Procedure 3. Enlarge a fully extended data set (in a DB2 storage group)" on page 551
- "Procedure 4. Add a data set" on page 551
- "Procedure 5. Redefine a partition (index-based partitioning)" on page 552
- "Procedure 6. Redefine a partition (table-based partitioning)" on page 552
- "Procedure 7. Enlarge a fully extended data set for the work file database" on page 552

If the database qualifier of the data set name is DSNDB07, then the condition is on your work file database. Use "Procedure 7. Enlarge a fully extended data set for the work file database" on page 552.

In all other cases, if the data set has *not* reached its maximum DB2 size, then you can enlarge it. (The maximum size is 2 GB for a data set of a simple space, and 1, 2, or 4 GB for a data set containing a partition. Large partitioned table spaces and indexes on large partitioned table spaces have a maximum data set size of 4 GB.)

- If the data set has *not* reached the maximum number of VSAM extents, use "Procedure 1. Extend a data set" on page 550.
- If the data set *has* reached the maximum number of VSAM extents, use either "Procedure 2. Enlarge a fully extended data set (user-managed)" on page 550 or "Procedure 3. Enlarge a fully extended data set (in a DB2 storage group)" on page 551, depending on whether the data set is user-managed or DB2-managed. User-managed data sets include essential data sets such as the catalog and the directory.

If the data set *has* reached its maximum DB2 size, then your action depends on the type of object it supports.

- If the object is a simple space, add a data set, using "Procedure 4. Add a data set" on page 551.
- If the object is partitioned, each partition is restricted to a single data set. You must redefine the partitions; use either "Procedure 5. Redefine a partition (index-based partitioning)" on page 552 or "Procedure 6. Redefine a partition (table-based partitioning)" on page 552.

Procedure 1. Extend a data set: If the data set is user-defined, provide more VSAM space. You can add volumes with the access method services command ALTER ADDVOLUMES or make room on the current volume.

If the data set is defined in a DB2 storage group, add more volumes to the storage group by using the SQL ALTER STOGROUP statement.

For more information about DB2 data set extension, refer to "Extending DB2-managed data sets" on page 31.

Procedure 2. Enlarge a fully extended data set (user-managed):

1. To allow for recovery in case of failure during this procedure, be sure that you have a recent full image copy (for table spaces or if you copy your indexes). Use the DSNUM option to identify the data set for table spaces or partitioning indexes.
2. Issue the command STOP DATABASE SPACENAM for the last data set of the object supported.
3. Depending on which version of z/OS you have and how many extents you need, complete one of the following tasks:

   You do not have z/OS V1.7 or later:

   (You can use this option with any version of z/OS, but you are limited to 255 extents.)

   Delete the last data set by using access method services. Then redefine it and enlarge it as necessary, by giving new values for PRIQTY and SECQTY.

   The object must be user-defined and a linear data set, and should not have reached the maximum number of 32 data sets for a nonpartitioned table space (or 254 data sets for LOB table spaces). For a partitioned table space, a partitioning index, or a nonpartitioning index on a partitioned table space, the maximum is 4096 data sets.

   You have z/OS V1.7 or later:

   If the DB2 subsystem is running on z/OS V1.7 or later, and the data set will not be shared with any z/OS systems at an earlier level, convert the data set to SMS-managed with the Extent Constraint Removal option set to YES in the SMS data class. If you do this, the maximum number of extents is 7257.

   If you have z/OS V1.7 and an older version coexisting in a data-sharing environment, a DB2 data set extended over 255 extents in V1.7 is not accessible by the lower release.
4. Issue the command START DATABASE ACCESS (UT) to start the object for utility-only access.

5. To recover the data set that was redefined, use RECOVER on the table space or index, and identify the data set by the DSNUM option (specify this DSNUM option for table spaces or partitioning indexes only).

   RECOVER lets you specify a single data set number for a table space. Thus, only the last data set (the one that needs extension) must be redefined and recovered. This can be better than using REORG if the table space is very large and contains multiple data sets, and if the extension must be done quickly.

   If you do not copy your indexes, then use the REBUILD INDEX utility.
6. Issue the command START DATABASE to start the object for either RO or RW access, whichever is appropriate.

Procedure 3. Enlarge a fully extended data set (in a DB2 storage group):

1. Depending on which version of z/OS you have and how many extents you need, complete one of the following tasks:

   You do not have z/OS V1.7 or later:
   (You can use this option with any version of z/OS, but you are limited to 255 extents.)

   Use ALTER TABLESPACE or ALTER INDEX with a USING clause. (You do not have to stop the table space before you use ALTER TABLESPACE.) You can give new values for PRIQTY and SECQTY in the storage group.

   You have z/OS V1.7 or later:
   If the DB2 subsystem is running on z/OS V1.7 or later, and the data set will not be shared with any z/OS systems at an earlier level, convert the data set to SMS-managed with the Extent Constraint Removal option set to YES in the SMS data class. If you do this, the maximum number of extents is 7257.

   If you have z/OS V1.7 and an older version coexisting in a data-sharing environment, a DB2 data set extended over 255 extents in V1.7 is not accessible by the lower release.
2. Use one of the following procedures. Keep in mind that no movement of data occurs until this step is completed.
   - For indexes:

     If you have taken full image copies of the index, run the RECOVER INDEX utility. Otherwise, run the REBUILD INDEX utility.
   - For table spaces other than LOB table space:

     Run one of the following utilities on the table space: REORG, RECOVER, or LOAD REPLACE.
   - For LOB table spaces defined with LOG YES:

     Run the RECOVER utility on the table space.
   - For LOB table spaces defined with LOG NO, follow these steps:
     a. Start the table space in read-only (RO) mode to ensure that no updates are made during this process.
     b. Make an image copy of the table space.
     c. Run the RECOVER utility on the table space.
     d. Start the table space in read-write (RW) mode.

Procedure 4. Add a data set: If the object supported is user-defined, use the access method services to define another data set. The name of the new data set must continue the sequence begun by the names of the existing data sets that support

the object. The last four characters of each name are a relative data set number: If the last name ended with A001, the next must end with A002, and so on. Also, be sure to add either **I** or **J** in the name of the data set.

If the object is defined in a DB2 storage group, DB2 automatically tries to create an additional data set. If that fails, access method services messages are sent to an operator indicating the cause of the problem. Correcting that problem allows DB2 to get the additional space.

Procedure 5. Redefine a partition (index-controlled partitioning):

1. Use ALTER INDEX ALTER PARTITION to alter the key range values of the partitioning index.
2. Use REORG with inline statistics on the partitions that are affected by the change in key range.
3. Use RUNSTATS on the nonpartitioned indexes.
4. Rebind the dependent packages and plans.

Procedure 6. Redefine a partition (table-controlled partitioning):

1. Use ALTER TABLE ALTER PARTITION to alter the partition boundaries.
2. Use REORG with inline statistics on the partitions that are affected by the change in partition boundries.
3. Use RUNSTATS on the indexes.
4. Rebind the dependent packages and plans.

Procedure 7. Enlarge a fully extended data set for the work file database:

Use one of the following methods to add space for extension to the DB2 storage group:

- Use SQL to create more table spaces in database DSNDB07.

Or,

- Execute these steps:
  1. Use the command STOP DATABASE(DSNDB07) to ensure that no users are accessing the database.
  2. Use SQL to alter the storage group, adding volumes as necessary.
  3. Use the command START DATABASE(DSNDB07) to allow access to the database.

Violation of referential constraint recovery

Problem: A table space can contain violations of referential constraints.

Symptom: One of the following messages is issued at the end of utility processing, depending upon whether or not the table space is partitioned.

```
DSNU561I csect-name - TABLESPACE= tablespace-name PARTITION= partnum
         IS IN CHECK PENDING
DSNU563I csect-name - TABLESPACE= tablespace-name IS IN CHECK PENDING
```

System action: None. The table space is still available; however, it is not available to the COPY, REORG, and QUIESCE utilities, or to SQL select, insert, delete, or update operations that involve tables in the table space.

System programmer action: None.

Operator action:

1. Use the START DATABASE ACCESS (UT) command to start the table space for utility-only access.
2. Run the CHECK DATA utility on the table space. Take the following recommendations into consideration:
   - If you do not believe that violations exist, specify DELETE NO. If, indeed, violations do not exist, this resets the check-pending status; however, if violations do exist, the status is not going to be reset.
   - If you believe that violations exist, specify the DELETE YES option and an appropriate exception table (see Part 2 of *DB2 Utility Guide and Reference* for the syntax of this utility). This deletes all rows in violation, copies them to an exception table, and resets the check-pending status.
   - If the check-pending status was set during execution of the LOAD utility, specify the SCOPE PENDING option. This checks only those rows added to the table space by LOAD, rather than every row in the table space.
3. Correct the rows in the exception table, if necessary, and use the SQL INSERT statement to insert them into the original table.
4. Issue the command START DATABASE to start the table space for RO or RW access, whichever is appropriate. The table space is no longer in check-pending status and is available for use. If you use the ACCESS (FORCE) option of this command, the check-pending status is reset. However, this is not recommended because it does not correct violations of referential constraints.

Distributed data facility failure recovery

The following failures related to the DDF are discussed in this section:

"Conversation failure recovery"
"Communications database failure recovery" on page 554
"Database access thread failure recovery" on page 555
"VTAM failure recovery" on page 555
"TCP/IP failure recovery" on page 556
"Remote logical unit failure recovery" on page 556
"Indefinite wait condition recovery" on page 557
"Security failure recovery for database access threads" on page 557

Conversation failure recovery

Problem: A VTAM APPC or TCP/IP conversation failed during or after allocation and is unavailable for use.

Symptom: VTAM or TCP/IP returns a resource unavailable condition along with the appropriate diagnostic reason code and message. A DSNL500 or DSNL511 (conversation failed) message is sent to the console for the first failure to a location for a specific logical unit (LU) mode or TCP/IP address. All other threads detecting a failure from that LU mode or IP address are suppressed until communications to that LU using that mode are successful.

DB2 returns messages DSNL501I and DSNL502I. Message DSNL501I usually means that the other subsystem is not up.

System action: When the error is detected, it is reported by a console message and the application receives an SQL return code. For DB2 private protocol access, SQLCODE -904 (SQLSTATE '57011') is returned with resource type 1001, 1002, or 1003. The resource name in the SQLCA contains VTAM return codes such as

RTNCD, FDBK2, RCPRI, and RCSEC, and any SNA SENSE information. See *VTAM for MVS/ESA Messages and Codes* for more information.

If you use application directed access or DRDA as the database protocols, SQLCODE -30080 is returned to the application. The SQLCA contains the VTAM diagnostic information, which contains only the RCPRI and RCSEC codes. For SNA communications errors, SQLCODE -30080 is returned. For TCP/IP connections, SQLCODE -30081 is returned. See *DB2 Codes*for more information about those SQL return codes.

The application can choose to request rollback or commit. Commit or rollback processing deallocates all but the first conversation between the allied thread and the remote database access thread. A commit or rollback message is sent over this remaining conversation.

Errors during the conversation's deallocation process are reported through messages, but do not stop the commit or rollback processing. If the conversation used for the commit or roll back message fails, the error is reported. If the error occurred during a commit process, the commit process continues, provided the remote database access was read only; otherwise the commit process is rolled back.

System programmer action: The system programmer needs to review the VTAM or TCP/IP return codes and might need to discuss the problem with a communications expert. Many VTAM or TCP/IP errors, besides the error of an inactive remote LU or TCP/IP errors, require a person who has a knowledge of VTAM or TCP/IP and the network configuration to diagnose them.

Operator action: Correct the cause of the unavailable resource condition by taking action required by the diagnostic messages appearing on the console.

Communications database failure recovery

This section describes two different problems.

Problem 1

A failure occurs during an attempt to access the DB2 CDB (after DDF is started).

Symptom: A DSNL700I message, indicating that a resource unavailable condition exists, is sent to the console. Other messages describing the cause of the failure are also sent to the console.

System action: The distributed data facility (DDF) does not terminate if it has already started and an individual CDB table becomes unavailable. Depending on the severity of the failure, threads will either receive a -904 SQL return code (SQLSTATE '57011') with resource type 1004 (CDB), or continue using VTAM defaults. Only the threads that access locations that have not had any prior threads will receive a -904 SQL return code. DB2 and DDF remain up.

Operator action: Correct the error based on the messages received, then stop and restart DDF.

Problem 2

The DB2 CDB is not defined correctly. This occurs when DDF is started and the DB2 catalog is accessed to verify the CDB definitions.

Symptom: A DSNL701I, 702I, 703I, 704I, or 705I message is issued to identify the problem. Other messages describing the cause of the failure are also sent to the console.

System action: DDF fails to start. DB2 continues to run.

Operator action: Correct the error based on the messages received and restart DDF.

Database access thread failure recovery

Problem: A database access thread has been deallocated and a conversation failure occurs.

Symptom: In the event of a failure of a database access thread, the DB2 server terminates the database access thread only if a unit of recovery exists. The server deallocates the database access thread and then deallocates the conversation with an abnormal indication (a negative SQL code), which is subsequently returned to the requesting application. The returned SQL code depends on the type of remote access:

- DB2 private protocol access

  The application program receives a -904 SQL return code (SQLSTATE '57011') with a resource type 1005 at the requesting site. The SNA sense in the resource name contains the DB2 reason code describing the failure.
- DRDA access

  For a database access thread or non-DB2 server, a DDM error message is sent to the requesting site and the conversation is deallocated normally. The SQL error status code is a -30020 with a resource type ’1232’ (agent permanent error received from the server).

System action: Normal DB2 error recovery mechanisms apply with the following exceptions:

- Errors caught in the functional recovery routine are automatically converted to rollback situations. The allied thread sees conversation failures.
- Errors occurring during commit, roll back, and deallocate *within the DDF function* do not normally cause DB2 to abend. Conversations are deallocated and the database access thread is terminated. The allied thread sees conversation failures.

System programmer action: All diagnostic information related to the failure must be collected at the serving site. For a DB2 DBAT, a dump is produced at the server.

Operator action: Communicate with the operator at the other site to take the appropriate corrective action, regarding the messages appearing on consoles at both the requesting and responding sites. Operators at both sites should gather the appropriate diagnostic information and give it to the programmer for diagnosis.

VTAM failure recovery

Problem: VTAM terminates or fails.

Symptom: VTAM messages and DB2 messages are issued indicating that DDF is terminating and explaining why.

System action: DDF terminates.

An abnormal VTAM failure or termination causes DDF to issue a STOP DDF MODE(FORCE) command. The VTAM commands Z NET,QUICK or Z NET,CANCEL causes an abnormal VTAM termination. A Z NET,HALT causes a STOP DDF MODE(QUIESCE) to be issued by DDF.

System programmer action: None.

Operator action: Correct the condition described in the messages received at the console, and restart VTAM and DDF.

TCP/IP failure recovery

Problem: TCP/IP terminates or fails.

Symptom: TCP/IP messages and DB2 messages are issued indicating that TCP/IP is unavailable.

System action: DDF periodically attempts to reconnect to TCP/IP. If the TCP/IP listener fails, DDF automatically tries to reestablish the TCP/IP listener for the SQL port or the resync port every three minutes. TCP/IP connections cannot be established until the TCP/IP listener is reestablished.

System programmer action: None.

Operator action: Correct the condition described in the messages received at the console, restart TCP/IP. You do not need to restart DDF after a TCP/IP failure.

Remote logical unit failure recovery

Problem: A series of conversation or change number of sessions (CNOS) failures occur from a remote LU.

Symptom: Message DSNL501I is issued when a CNOS request to a remote LU fails. The CNOS request is the first attempt to connect to the remote site and must be negotiated before any conversations can be allocated. Consequently, if the remote LU is not active, message DSNL500I is displayed indicating that the CNOS request cannot be negotiated. Message DSNL500I is issued only once for all the SQL conversations that fail because of a remote LU failure.

Message DSNL502I is issued for system conversations that are active to the remote LU at the time of the failure. This message contains the VTAM diagnostic information about the cause of the failure.

System action: Any application communications with a failed LU receives a message indicating a resource unavailable condition. The application programs receive SQL return code -904 (SQLSTATE '57011') for DB2 private protocol access and SQL return code -30080 for DRDA access. Any attempt to establish communication with such an LU fails.

Operator action: Communicate with the other sites involved regarding the unavailable resource condition, and request that appropriate corrective action be taken. If a DSNL502 message is received, the operator should activate the remote LU.

Indefinite wait condition recovery

Problem: An allied thread is waiting indefinitely for a response from a remote subsystem or a database access thread is waiting for a response from the local subsystem.

Symptom: An application is in an indefinitely long wait condition. This can cause other DB2 threads to fail due to resources held by the waiting thread. DB2 sends an error message to the console and the application program receives an SQL return code.

System action: None.

System programmer action: None.

Operator action: Use the DISPLAY THREAD command with the LOCATION and DETAIL options to identify the LUWID and the session's allocation for the waiting thread. Then use the CANCEL DDF THREAD command to cancel the waiting thread. If the CANCEL DDF THREAD command fails to break the wait (because the thread is not suspended in DB2), try using VTAM commands such as VARY TERM,SID=xxx. For additional information concerning canceling DDF threads, see "The command CANCEL THREAD" on page 411 and "Using VTAM commands to cancel threads" on page 412.

To check for very long waits, look to see if the conversation timestamp is changing from the last time used. If it is changing, the conversation thread is not hung, but is taking more time for a long query. Also, look for conversation state changes and determine what they mean.

Security failure recovery for database access threads

Problem: During database access thread allocation, the remote user does not have the proper security to access DB2 through the DDF.

Symptom: Message DSNL500I is issued at the requester for VTAM conversations (if it is a DB2 subsystem) with return codes RTNCD=0, FDBK2=B, RCPRI=4, RCSEC=5 meaning "Security Not Valid." The server has deallocated the conversation because the user is not allowed to access the server. For conversations using DRDA access, LU 6.2 communications protocols present specific reasons for why the user failed, to be returned to the application. If the server is a DB2 database access thread, message DSNL030I is issued to describe what caused the user to be denied access into DB2 through DDF. No message is issued for TCP/IP connections.

System action: If the server is a DB2 subsystem, message DSNL030I is issued. Otherwise, the system programmer needs to refer to the documentation of the server. If the application uses DB2 private protocol access, it receives SQLCODE -904 (SQLSTATE '57011') with a reason code 00D3103D, indicating that a resource is unavailable. For DRDA access, SQLCODE –30082 is returned. See *DB2 Codes* for more information about those messages.

System programmer action: Refer to the description of 00D3103D in Part 3 of *DB2 Codes*.

Operator action: If it is a DB2 database access thread, the operator should provide the DSNL030I message to the system programmer. If it is not a DB2 server, the

operator needs to work with the operator or programmer at the server to get diagnostic information needed by the system programmer.

Remote site recovery from a disaster at the local site

Problem: Your local system experiences damage or disruption that prevents recovery from that site.

Symptom: Your local system hardware has suffered physical damage and is not operational.

System programmer action: Coordinate the activities detailed in "Restoring data from image copies and archive logs."

Operator action (at the recovery site): The procedures in this scenario differ from other recovery procedures because you cannot use the hardware at your local DB2 site to recover data. You use hardware at a remote site to recover after a disaster using one of the following three methods:

- "Restoring data from image copies and archive logs"
- "Using a tracker site for disaster recovery" on page 568
- "Using data mirroring" on page 576

Restoring data from image copies and archive logs

This scenario bases recovery on the latest available archive log and assumes that all copies and reports have arrived at the recovery site as specified in "Preparing for disaster recovery" on page 483. For data sharing, see Chapter 5 of *DB2 Data Sharing: Planning and Administration* for recovery procedures that are specific to data sharing.

1. For scenarios other than data sharing, continue with the next step.

Data sharing

Remove old information from the coupling facility, if you have information in your coupling facility from practice startups. If you do not have old information in the coupling facility, you can omit this step.

a. Enter the following z/OS command to display the structures for this data sharing group:

```
D XCF,STRUCTURE,STRNAME=grpname*
```

b. For group buffer pools and the lock structure, enter the following command to force the connections off those structures:

```
SETXCF FORCE,CONNECTION,STRNAME=strname,CONNAME=ALL
```

Connections for the SCA are not held at termination, so you do not need to force off any SCA connections.

c. Delete all the DB2 coupling facility structures by using the following command for each structure:

```
SETXCF FORCE,STRUCTURE,STRNAME=strname
```

This step is necessary to remove old information that exists in the coupling facility from your practice startup when you installed the group.

2. If an integrated catalog facility catalog does not already exist, run job DSNTIJCA to create a user catalog.

3. Use the access method services IMPORT command to import the integrated catalog facility catalog.
4. Restore DB2 libraries, such as DB2 reslibs, SMP libraries, user program libraries, user DBRM libraries, CLISTs, SDSNSAMP, or where the installation jobs are, JCL for user-defined table spaces, and so on.
5. Use IDCAMS DELETE NOSCRATCH to delete all catalog and user objects. (Because step 3 imports a user ICF catalog, the catalog reflects data sets that do not exist on disk.)

   # Obtain a copy of installation job DSNTIJIN. This job creates DB2 VSAM and
   # non-VSAM data sets. Change the volume serial numbers in the job to volume
   # serial numbers that exist at the recovery site. Comment out the steps that
   # create DB2 non-VSAM data sets, if these data sets already exist. Run
   # DSNTIJIN. However, do not run DSNTIJID.

   Data sharing
   Obtain a copy of the installation job DSNTIJIN for the first data sharing member to be migrated. Run DSNTIJIN on the first data sharing member. For subsequent members of the data sharing group, run the DSNTIJIN that defines the BSDS and logs.

6. Recover the BSDS:
   a. Use the access method services REPRO command to restore the contents of one BSDS data set (allocated in the previous step). The most recent BSDS image can be found in the last file (archive log with the highest number) on the latest archive log tape.

      Data sharing
      The BSDS data sets on each data sharing member need to be restored.

   b. To determine the RBA range for this archive log, use the print log map utility (DSNJU004) to list the current BSDS contents. Find the most recent archive log in the BSDS listing and add 1 to its ENDRBA value. Use this as the STARTRBA. Find the active log in the BSDS listing that starts with this RBA and use its ENDRBA as the ENDRBA.

      Data Sharing
      The LRSNs are also required.

   # c. Delete the oldest archive log from the BSDS.
   d. Use the change log inventory utility (DSNJU003) to register this latest archive log tape data set in the archive log inventory of the BSDS just restored. This is necessary because the BSDS image on an archive log tape does not reflect the archive log data set residing on that tape.

Data sharing

Running DSNJU003 is critical for data sharing groups. Group buffer pool checkpoint information is stored in the BSDS and needs to be included from the most recent archive log.

After these archive logs are registered, use the print log map utility (DSNJU004) with the GROUP option to list the contents of all the BSDSs. You recieve output that includes the start and end LRSN and RBA values for the latest active log data sets (shown as NOTREUSABLE). If you did not save the values from the DSNJ003I message, you can get those values by running DSNJU004, which creates output similar the output that is shown in Figure 53 and Figure 54.

Figure 53 shows a partial example of output from the DSNJU004 utility. This output contains the BSDS information for the archive log member DB1G.

```
ACTIVE LOG COPY 1 DATA SETS
  START RBA/LRSN/TIME   END RBA/LRSN/TIME      DATE    LTIME  DATA SET INFORMATION
 --------------------  --------------------  --------  -----  --------------------
   000001C20000           000001C67FFF       1996.358  17:25  DSN=DSNDB0G.DB1G.LOGCOPY1.DS03
     ADFA0FB26C6D           ADFA208AA36B                      STATUS=TRUNCATED, REUSABLE
 1996.361  23:37:48.4  1996.362  00:53:10.1
   000001C68000           000001D4FFFF       1996.358  17:25  DSN=DSNDB0G.DB1G.LOGCOPY1.DS01
     ADFA208AA36C           AE3C45273A77                      STATUS=TRUNCATED, NOTREUSABLE
 1996.362  00:53:10.1  1997.048  15:28:23.5
   000001D50000           0000020D3FFF       1996.358  17:25  DSN=DSNDB0G.DB1G.LOGCOPY1.DS02
     AE3C45273A78           ............                      STATUS=NOTREUSABLE
 1997.048  15:28:23.5  ........  ..........
```

Figure 53. BSDS contents (partial) of member DB1G

Figure 54 shows a partial example of output from the DSNJU004 utility. This output contains the BSDS information for the archive log member DB2G.

```
ACTIVE LOG COPY 1 DATA SETS
  START RBA/LRSN/TIME   END RBA/LRSN/TIME      DATE    LTIME  DATA SET INFORMATION
 --------------------  --------------------  --------  -----  --------------------
                 EMPTY DATA SET              1996.361  14:14  DSN=DSNDB0G.DB2G.LOGCOPY1.DS03
     000000000000           000000000000                      STATUS=NEW, REUSABLE
 0000.000  00:00:00.0  0000.000  00:00:00.0
   000000000000           0000000D6FFF       1996.361  14:14  DSN=DSNDB0G.DB2G.LOGCOPY1.DS01
     ADFA00BB70FB           AE3C45276DD7                      STATUS=TRUNCATED, NOTREUSABLE
 1996.361  22:30:51.4  1997.048  15:28:23.7
   0000000D7000           00000045AFFF       1996.361  14:14  DSN=DSNDB0G.DB2G.LOGCOPY1.DS02
     AE3C45276DD8           ............                      STATUS=NOTREUSABLE
 1997.048  15:28:23.7  ........  ..........
```

Figure 54. BSDS contents (partial) of member DB2G

Data sharing

Do all other preparatory activities as you would for a single system. Do these activities for *each* member of the data sharing group.

e. Use the change log inventory utility to adjust the active logs:

1) Use the DELETE option of the change log inventory utility (DSNJU003) to delete all active logs in the BSDS. Use the BSDS listing produced in step 6d on page 559 to determine the active log data set names.
2) Use the NEWLOG statement of the change log inventory utility (DSNJU003) to add the active log data sets to the BSDS. Do not specify a STARTRBA or ENDRBA value in the NEWLOG statement. This indicates to DB2 that the new active logs are empty.

f. If you are using the DB2 distributed data facility, run the change log inventory utility with the DDF statement to update the LOCATION and the LUNAME values in the BSDS.

g. Use the print log map utility (DSNJU004) to list the new BSDS contents and ensure that the BSDS correctly reflects the active and archive log data set inventories. In particular, ensure that:
   - All active logs show a status of NEW and REUSABLE
   - The archive log inventory is complete and correct (for example, the start and end RBAs should be correct).

h. If you are using dual BSDSs, make a copy of the newly restored BSDS data set to the second BSDS dataset.

7. Optionally, you can restore archive logs to disk. Archive logs are typically stored on tape, but restoring them to disk could speed later steps. If you elect this option, and the archive log data sets are not cataloged in the primary integrated catalog facility catalog, use the change log inventory utility to update the BSDS. If the archive logs are listed as cataloged in the BSDS, DB2 allocates them using the integrated catalog and not the unit or volser specified in the BSDS. If you are using dual BSDSs, remember to update both copies.

8. Use the DSN1LOGP utility to determine which transactions were in process at the end of the last archive log. Use the following job control language where *yyyyyyyyyyyy* is the STARTRBA of the last complete checkpoint within the RBA range on the last archive log from the previous print log map:

```
//SAMP  EXEC PGM=DSN1LOGP
//SYSPRINT DD SYSOUT=*
//SYSSUMRY DD SYSOUT=*
//ARCHIVE  DD DSN=last-archive,DISP=(OLD,KEEP),UNIT=TAPE,
              LABEL=(2,SL),VOL=SER=volser1
              (NOTE FILE 1 is BSDS COPY)
//SYSIN    DD *
  STARTRBA(yyyyyyyyyyyy) SUMMARY(ONLY)
/*
```

DSN1LOGP gives a report. For sample output and information about how to read it, see Part 3 of *DB2 Utility Guide and Reference*.

Note whether any utilities were executing at the end of the last archive log. You will have to determine the appropriate recovery action to take on each table space involved in a utility job.

If DSN1LOGP showed that utilities are inflight (PLAN=DSNUTIL), you need SYSUTILX to identify the utility status and determine the recovery approach. See "What to do about utilities in progress" on page 565.

9. Modify DSNZP*xxx* parameters:

a. Run the DSNTINST CLIST in UPDATE mode. See Part 2 of *DB2 Installation Guide*.

b. To defer processing of all databases, select "Databases to Start Automatically" from panel DSNTIPB. You are presented with panel DSNTIPS. Type DEFER in the first field, ALL in the second, and press Enter. You are returned to DSNTIPB.

c. To specify where you are recovering, select “Operator Functions” from panel DSNTIPB. You are presented with panel DSNTIPO. Type RECOVERYSITE in the SITE TYPE field. Press Enter to continue.

d. To optionally specify which archive log to use, select “Operator Functions” from panel DSNTIPB. You are presented with panel DSNTIPO. Type YES in the READ ARCHIVE COPY2 field if you are using dual archive logging and want to use the second copy of the archive logs. Press Enter to continue.

e. Reassemble DSNZP*xxx* using job DSNTIJUZ (produced by the CLIST started in the first step).

At this point, you have the log, but the table spaces have not been recovered. With DEFER ALL, DB2 assumes that the table spaces are unavailable, but does the necessary processing to the log. This step also handles the units of recovery in process.

10. Use the change log inventory utility to create a conditional restart control record. In most cases, you can use this form of the CRESTART statement:

```
CRESTART CREATE,ENDRBA=nnnnnnnnn000,FORWARD=YES,
         BACKOUT=YES
```

where *nnnnnnnnn*000 equals a value one more than the ENDRBA of the latest archive log.

Data sharing

If you are recovering a data sharing group, and your logs are not at a single point of consistency, use this form of the CRESTART statement:

```
CRESTART CREATE,ENDLRSN=nnnnnnnnnnnn,FORWARD=YES,BACKOUT=YES
```

where *nnnnnnnnnnnn* is the LRSN of the last log record to be used during restart. Use the same LRSN for **all** members in a data sharing group. Determine the ENDLRSN value using one of the following methods:

- Use the DSN1LOGP summary utility to obtain the ENDLRSN value. In the ‘Summary of Completed Events’ section, find the lowest LRSN value listed in the DSN1213I message, for the data sharing group. Use this value for the ENDLRSN in the CRESTART statement.
- Use the print log map utility (DSNJU004) to list the BSDS contents. Find the ENDLRSN of the last log record available for each active member of the data sharing group. Subtract 1 from the **lowest** ENDLRSN in the data sharing group. Use this value for the ENDLRSN in the CRESTART statement. (In our example in Figure 53 on page 560, that is AE3C45273A77 - 1, which is AE3C45273A76.)
- If only the console logs are available, use the archive offload message, DSNJ003I to obtain the ENDLRSN. Compare the ending LRSN values for all members’ archive logs. Subtract 1 from the **lowest** LRSN in the data sharing group. Use this value for the ENDLRSN in the CRESTART statement. (In our example in Figure 53 on page 560, that is AE3C45273A77 - 1, which is AE3C45273A76.)

DB2 discards any log information in the bootstrap data set and the active logs with an RBA greater than or equal to *nnnnnnnnn*000 or an LRSN greater than *nnnnnnnnnnnn* as listed in the preceding CRESTART statements.

Use the print log map utility to verify that the conditional restart control record that you created in the previous step is active.

11. Enter the command START DB2 ACCESS(MAINT).

> **Data Sharing**
>
> If a discrepancy exists among the print log map reports as to the number of members in the group, record the one that shows the highest number of members. (This is an unlikely occurrence.) Start this DB2 subsystem first using ACCESS(MAINT). DB2 prompts you to start each additional DB2 subsystem in the group.
>
> After all additional members are successfully restarted, and if you are going to run single-system data sharing at the recovery site, stop all DB2 subsystems but one by using the STOP DB2 command with MODE(QUIESCE).
>
> If you planned to use the light mode when starting the DB2 group, add the LIGHT parameter to the START command. Start the members that run in LIGHT(NO) mode first, followed by the LIGHT(YES) members. See "Preparing for disaster recovery" on page 483 for details on using restart light at a recovery site.

Even though DB2 marks all table spaces for deferred restart, log records are written so that in-abort and inflight units of recovery are backed out. In-commit units of recovery are completed, but no additional log records are written at restart to cause this. This happens when the original redo log records are applied by the RECOVER utility.

At the primary site, DB2 probably committed or aborted the inflight units of recovery, but you have no way of knowing.

During restart, DB2 accesses two table spaces that result in DSNT501I, DSNT500I, and DSNL700I resource unavailable messages, regardless of DEFER status. The messages are normal and expected, and you can ignore them.

The following return codes can accompany the message. Other codes are also possible.

00C90081
: This return code occurs if there is activity against the object during restart as a result of a unit of recovery or pending writes. In this case the status shown as a result of DISPLAY is STOP,DEFER.

00C90094
: Because the table space is currently only a defined VSAM data set, it is in an unexpected state to DB2.

00C900A9
: This codes indicates that an attempt was made to allocate a deferred resource.

12. Resolve the indoubt units of recovery.

    The RECOVER utility, which you will soon invoke, will fail on any table space that has indoubt units of recovery. Because of this, you must resolve them first. Determine the proper action to take (commit or abort) for each unit of recovery. To resolve indoubt units of recovery, see "Resolving indoubt units of recovery" on page 459. From an install SYSADM authorization ID, enter the RECOVER INDOUBT command for all affected transactions.

13. To recover the catalog and directory, follow these instructions:

    The RECOVER function includes: RECOVER TABLESPACE, RECOVER INDEX, or REBUILD INDEX. If you have an image copy of an index, use

RECOVER INDEX. If you do not have an image copy of an index, use REBUILD INDEX to reconstruct the index from the recovered table space.

a. Recover DSNDB01.SYSUTILX. This must be a separate job step.
b. Recover all indexes on SYSUTILX. This must be a separate job step.
c. Your recovery strategy for an object depends on whether a utility was running against it at the time the latest archive log was created. To identify the utilities that were running, you must recover SYSUTILX.

   You cannot restart a utility at the recovery site that was interrupted at the disaster site. You must use the TERM command to terminate it. The TERM UTILITY command can be used on any object except DSNDB01.SYSUTILX.

   Determine which utilities were executing and the table spaces involved by performing the following procedure:

   1) Enter the DISPLAY UTILITY(*) command and record the utility and the current phase.
   2) Run the DIAGNOSE utility with the DISPLAY SYSUTILX statement. The output consists of information about each active utility, including the table space name (in most instances). It is the only way to correlate the object name with the utility. Message DSNU866I gives information about the utility, while DSNU867I gives the database and table space name in USUDBNAM and USUSPNAM respectively.

d. Use the command TERM UTILITY to terminate any utilities in progress on catalog or directory table spaces.

   See "What to do about utilities in progress" on page 565 for information about how to recover catalog and directory table spaces on which utilities were running.
e. Recover the rest of the catalog and directory objects starting with DBD01, in the order shown in the description of the RECOVER utility in Part 2 of *DB2 Utility Guide and Reference*.

14. Use any method desired to verify the integrity of the DB2 catalog and directory. Migration step 1 in Chapter 1 of *DB2 Installation Guide* lists one option for verification. The catalog queries in member DSNTESQ of data set DSN810.SDSNSAMP can be used after the work file database is defined and initialized.
15. Define and initialize the work file database.
    a. Define temporary work files. Use installation job DSNTIJTM as a model.
    b. Issue the command START DATABASE(*work-file-database*) to start the work file database.
16. If you use data definition control support, recover the objects in the data definition control support database.
17. If you use the resource limit facility, recover the objects in the resource limit control facility database.
18. Modify DSNZP*xxx* to restart all databases:
    a. Run the DSNTINST CLIST in UPDATE mode. See Part 2 of *DB2 Installation Guide*.
    b. From panel DSNTIPB select "Databases to Start Automatically". You are presented with panel DSNTIPS. Type RESTART in the first field, ALL in the second and press Enter. You are returned to DSNTIPB.
    c. Reassemble DSNZP*xxx* using job DSNTIJUZ (produced by the CLIST started in the first step).
19. Stop and start DB2.
20. Make a full image copy of the catalog and directory.

21. Recover user table spaces and index spaces. See "What to do about utilities in progress" for information about how to recover table spaces or index spaces on which utilities were running. You cannot restart a utility at the recovery site that was interrupted at the disaster site. Use the TERM command to terminate any utilities running against user table spaces or index spaces.
    a. To determine which, if any, of your table spaces or index spaces are user-managed, perform the following queries for table spaces and index spaces.

       Table spaces:

       ```
       SELECT * FROM SYSIBM.SYSTABLEPART WHERE STORTYPE='E';
       ```

       Index spaces:

       ```
       SELECT * FROM SYSIBM.SYSINDEXPART WHERE STORTYPE='E';
       ```

       To allocate user-managed table spaces or index spaces, use the access method services DEFINE CLUSTER command. To find the correct IPREFIX for the DEFINE CLUSTER command, perform the following queries for table spaces and index spaces.

       Table spaces:

       ```
       SELECT DBNAME, TSNAME, PARTITION, IPREFIX FROM SYSIBM.SYSTABLEPART
       WHERE DBNAME=dbname AND TSNAME=tsname
       ORDER BY PARTITION;
       ```

       Index spaces:

       ```
       SELECT IXNAME, PARTITION, IPREFIX FROM SYSIBM.SYSINDEXPART
       WHERE IXCREATOR=ixcreator AND IXNAME=ixname
       ORDER BY PARTITION;
       ```

       Now you can perform the DEFINE CLUSTER command with the correct IPREFIX (I or J) in the data set name:

       ```
       catname.DSNDBC.dbname.spname.y0001.A00x
       ```

       where *y* can be either I or J, *x* is C (for VSAM clusters) or D (for VSAM data components), and *spname* is either the table space or index space name.

       Access method services commands are described in detail in *z/OS DFSMS Access Method Services for Catalogs*.
    b. If your user table spaces or index spaces are STOGROUP-defined, and if the volume serial numbers at the recovery site are different from those at the local site, use ALTER STOGROUP to change them in the DB2 catalog.
    c. Recover all user table spaces and index spaces from the appropriate image copies. If you do not copy your indexes, use the REBUILD INDEX utility to reconstruct the indexes.
    d. Start all user table spaces and index spaces for read or write processing by issuing the command START DATABASE with the ACCESS(RW) option.
    e. Resolve any remaining check pending states that would prevent COPY execution.
    f. Run select queries with known results.
22. Make full image copies of all table spaces and indexes with the COPY YES attribute.
23. Finally, compensate for lost work since the last archive was created by rerunning online transactions and batch jobs.

What to do about utilities in progress: If any utility jobs were running after the last time that the log was offloaded before the disaster, you might need to take

some additional steps. After restarting DB2, the following utilities only need to be terminated with the TERM UTILITY command:

- CHECK INDEX
- MERGECOPY
- MODIFY
- QUIESCE
- RECOVER
- RUNSTATS
- STOSPACE

It is preferable to allow the RECOVER utility to reset pending states. However, it is occasionally necessary to use the REPAIR utility to reset them. Do not start the table space with ACCESS(FORCE) because FORCE resets any page set exception conditions described in "Database page set control records" on page 1116.

For the following utility jobs, perform the actions indicated:

CHECK DATA
Terminate the utility and run it again after recovery is complete.

COPY After you enter the TERM command, DB2 places a record in the SYSCOPY catalog table indicating that the COPY utility was terminated. This makes it necessary for you to make a full image copy. When you copy your environment at the completion of the disaster recovery scenario, you fulfill that requirement.

LOAD
Find the options you specified in Table 100, and perform the specified actions.

Table 100. Actions when LOAD is interrupted

LOAD options specified	What to do
LOG YES	If the RELOAD phase completed, recover to the current time. Recover the indexes. If the RELOAD phase did not complete, recover to a prior point in time. The SYSCOPY record inserted at the beginning of the RELOAD phase contains the RBA or LRSN.
LOG NO and *copy-spec*	If the RELOAD phase completed, the table space is complete after you recover it to the current time. Recover the indexes. If the RELOAD phase did not complete, then recover the table space to a prior point in time. Recover the indexes.
LOG NO, *copy-spec*, and SORTKEYS *integer*[1]	If the BUILD or SORTBLD phase completed, recover to the current time, and recover the indexes. If the BUILD or SORTBLD phase did not complete, recover to a prior point in time. Recover the indexes.
LOG NO	Recover the table space to a prior point in time. You can use TOCOPY to do this.

Note: [1]You must specify a value that is greater than zero for *integer*. If you specify zero for *integer* the SORTKEYS option does not apply.

To avoid extra loss of data in a future disaster situation, run QUIESCE on table spaces before invoking LOAD. This enables you to recover a table space using TOLOGPOINT instead of TOCOPY.

REORG

For a **user** table space, find the options you specified in Table 101, and perform the specified actions.

Table 101. Actions when REORG is interrupted

REORG options specified	What to do
LOG YES	If the RELOAD phase completed, recover to the current time. Recover the indexes. If the RELOAD phase did not complete, recover to the current time to restore the table space to the point before REORG began. Recover the indexes.
LOG NO	If the build or SORTBLD phase completed, recover to the current time, and recover the indexes. If the build or SORTBLD phase did not complete, recover to the current time to restore the table space to the point before REORG began. Recover the indexes.
SHRLEVEL CHANGE	If the SWITCH phase completed, terminate the utility. Recover the table space to the current time. Recover the indexes. If the SWITCH phase did not complete, recover the table space to the current time. Recover the indexes.
SHRLEVEL REFERENCE	Same as for SHRLEVEL CHANGE.

For a **catalog or directory** table space, follow these instructions:

For those table spaces that were using online REORG, find the options you specified in Table 101, and perform the specified actions.

If you have no image copies from immediately before REORG failed, use this procedure:

1. From your DISPLAY UTILITY and DIAGNOSE output, determine what phase REORG was in and which table space it was reorganizing when the disaster occurred.
2. Run RECOVER on the catalog and directory in the order shown in Part 2 of *DB2 Utility Guide and Reference*. Recover all table spaces to the current time, *except* the table space that was being reorganized. If the RELOAD phase of the REORG on that table space had not completed when the disaster occurred, recover the table space to the current time. Because REORG does not generate any log records prior to the RELOAD phase for catalog and directory objects, the RECOVER to current restores the data to the state it was in before the REORG. If the RELOAD phase completed, perform the following actions:
   a. Run DSN1LOGP against the archive log data sets from the disaster site.
   b. Find the begin-UR log record for the REORG that failed in the DSN1LOGP output.
   c. Run RECOVER with the TOLOGPOINT option on the table space that was being reorganized. Use the URID of the begin-UR record as the TOLOGPOINT value.
3. Recover or rebuild all indexes.

If you have image copies from immediately before REORG failed, run RECOVER with the option TOCOPY to recover the catalog and directory, in the order shown in Part 2 of *DB2 Utility Guide and Reference*.

Recommendation: Make full image copies of the catalog and directory before you run REORG on them.

Using a tracker site for disaster recovery

This section describes a different method for disaster recovery from that described in "Remote site recovery from a disaster at the local site" on page 558. Many steps are similar to a regular disaster recovery, so these steps are not described in detail here.

Recommendation: Test and document a disaster procedure that is customized for your location.

Overview: A DB2 *tracker* site is a separate DB2 subsystem or data sharing group that exists solely for the purpose of keeping shadow copies of your primary site's data. No independent work can be run on the tracker site.

From the primary site, you transfer the BSDS and the archive logs, and that tracker site runs periodic LOGONLY recoveries to keep the shadow data up-to-date. If a disaster occurs at the primary site, the tracker site becomes the *takeover* site. Because the tracker site has been shadowing the activity on the primary site, you do not have to constantly ship image copies; the takeover time for the tracker site might be faster because DB2 recovery does not have to use image copies.

The following topics are described in this section:

"Characteristics of a tracker site"
"Setting up a tracker site" on page 569
"Using RESTORE SYSTEM LOGONLY to establish a recovery cycle" on page 569
"Using the RECOVER utility to establish a recovery cycle" on page 571
"Media failures during LOGONLY recovery" on page 574
"Maintaining the tracker site" on page 574
"Making the tracker site the takeover site" on page 574

Characteristics of a tracker site

Because the tracker site must use only the primary site's logs for recovery, you must not update the catalog and directory or the data at the tracker site. The DB2 subsystem at the tracker site disallows updates. In summary:

- The following SQL statements are not allowed at a tracker site:
  - GRANT or REVOKE
  - DROP, ALTER, or CREATE
  - UPDATE, INSERT, or DELETE

  Dynamic read-only SELECT statements are also allowed, but not recommended. At the end of each tracker site recovery cycle, databases might contain uncommitted data and indexes might be inconsistent with the data at the tracker site.
- The only online utilities that are allowed are REPORT, DIAGNOSE, RECOVER, REBUILD, and RESTORE SYSTEM LOGONLY. Recovery to a prior point in time is not allowed.
- BIND is not allowed.
- TERM UTIL is not allowed for LOAD, REORG, REPAIR, and COPY.

- The START DATABASE command is not allowed when LPL or GRECP status exists for the object of the command. It is not necessary to use START DATABASE to clear LPL or GRECP conditions, because you are going to be running RECOVERY jobs that clear the conditions.
- The START DATABASE command with ACCESS(FORCE) is not allowed.
- Down-level detection is disabled.
- Log archiving is disabled.
- Real-time statistics are disabled.

Setting up a tracker site

To set up the tracker site:

1. Create a mirror image of your primary DB2 subsystem or data sharing group. This process is described in steps 1 through 4 of the normal disaster recovery procedure, which includes such creating catalogs and restoring DB2 libraries.
2. Modify the subsystem parameters as follows:
   - Set the TRKSITE subsystem parameter to YES.
   - Optionally, set the SITETYP parameter to RECOVERYSITE if the full image copies this site will be receiving are created as remote site copies.
3. Use the access method services DEFINE CLUSTER command to allocate data sets for all user-managed table spaces that you will be sending over from the primary site.
4. Optionally, you can allocate data sets for user-managed indexes that you want to rebuild during recovery cycles. The main reason you rebuild indexes during recovery cycles is for running efficient queries on the tracker site. If you do not require indexes, you do not have to rebuild them for recovery cycles. For nonpartitioning indexes on very large tables, you can include indexes for LOGONLY recovery during the recovery cycle, which can reduce the amount of time it takes to bring up the disaster site. Be sure you define data sets with the proper **I** or **J** prefix for both indexes and table spaces. See REORG INDEX and REORG TABLESPACE in *DB2 Utility Guide and Reference* for more information about index and table space prefixes.
5. Send full image copies of all the primary site's DB2 data to the tracker site. Optionally, you can use the BACKUP SYSTEM utility with the DATA ONLY option and send copies of the database copy pool to the tracker site. If you send copies that the BACKUP SYSTEM utility creates, this step completes the tracker site setup procedure.
6. If you did not use the BACKUP SYSTEM utility in step 5, tailor installation job DSNTIJIN to create DB2 catalog data sets.

Important: Do not attempt to start the tracker site when you are setting it up. You must follow the procedure described in "Using the RECOVER utility to establish a recovery cycle" on page 571.

Using RESTORE SYSTEM LOGONLY to establish a recovery cycle

When full image copies of all the data at the primary site are available to your tracker site, you can use the LOGONLY option of the RESTORE SYSTEM utility to periodically apply the active log, archive logs, and the BSDS from the primary site at the tracker site. Each time you restore the logs and the BSDS from the primary site at your tracker site, you establish a new recovery cycle.

Complete the following procedure to use the RESTORE SYSTEM utility to establish a recovery cycle at your tracker site:

1. While your primary site continues its usual workload, send a copy of the primary site's active log, archive logs, and BSDS to the tracker site.

   Send full image copies for the following objects:

   - Table spaces or partitions that are reorganized, loaded, or repaired with the LOG NO option after the latest recovery cycle.
   - Objects that, after the latest recovery cycle, have been recovered to a point in time

   Recommendation: If you are taking incremental image copies, run the MERGECOPY utility at the primary site before sending the copy to the tracker site.
2. At the tracker site, restore the BSDS that was received from the primary site.
   - Locate the BSDS in the latest archive log that is now at the tracker site.
   - Use the change log inventory utility (DSNJU003) to register this archive log in the archive log inventory of the new BSDS.
   - Use the change log inventory utility (DSNJU003) to register the primary site's active log in the new BSDS.

   For more details about restoring the BSDS, see step 6 of "Remote site recovery from a disaster at the local site" on page 558.
3. Use the change log inventory utility (DSNJU003) with the following CRESTART control statement:

   ```
   CRESTART CREATE,ENDRBA=nnnnnnnnn000,FORWARD=NO,BACKOUT=NO
   ```

   In this control statement, *nnnnnnnnn*000 equals the RBA at which the latest archive log record ends plus one. You must not specify the RBA at which the an archive log begins because you cannot cold start or skip logs in tracker mode.

   Data sharing

   If you are recovering a data sharing group, you **must** use the following CRESTART control statement on all members of the data sharing group. The ENDLRSN value must be the same for all members.

   ```
   CRESTART CREATE,ENDLSRN=nnnnnnnnnnnn,FORWARD=NO,BACKOUT=NO
   ```

   In this control statement, *nnnnnnnnnnnn* is the lowest LRSN of all the members to be read during restart. You must specify one of the following values for the ENDLRSN:

   - If you receive the ENDLRSN from the output of the print log map utility (DSNJU004) or from the console logs using message DSNJ003I, you must use ENDLRSN-1 as the input to the conditional restart.
   - If you receive the ENDLRSN from the output of the DSN1LOGP utility (DSN1213I message), you can use the value that is displayed.

   The ENDLRSN or ENDRBA value indicates the end log point for data recovery and for truncating the archive log. With ENDLRSN, the *missing* log records between the lowest and highest ENDLRSN values for all the members are applied during the next recovery cycle.
4. If the tracker site is a data sharing group, delete all DB2 coupling facility structures before restarting the tracker members.
5. If you used DSN1COPY to create a copy of SYSUTILX during the last tracker cycle, restore this copy with DSN1COPY.

6. At the tracker site, restart DB2 to begin a **tracker site recovery cycle**.

> **Data sharing**
> For data sharing, restart **every** member of the data sharing group.

7. At the tracker site, run the RESTORE SYSTEM utility with the LOGONLY option to apply the logs (both archive and active) to the data at the tracker site. See "Media failures during LOGONLY recovery" on page 574 for information about what to do if a media failure occurs during LOGONLY recovery.
8. If the RESTORE SYSTEM utility issues a return code of 4, use DSN1COPY to make a copy of SYSUTILX and indexes that are associated with SYSUTILX before you recover or rebuild those objects. DSN1COPY issues a return code of 4 if applying the log marks one or more DB2 objects as RECP or RBDP.
9. Restart DB2 at the tracker site.
10. Issue the DISPLAY DATABASE RESTRICT command to display objects that are marked RECP, RBDP, or LPL and identify which objects are in a utility progress state (such as UTUT or UTRO). Run the RECOVER or REBUILD INDEX utility on these objects, or record which objects are in an exception state so that you can recover them at a later time. The exception states of these objects will be lost in the next recovery cycle.
11. After all recovery has completed at the tracker site, shut down the tracker site DB2. This is the end of the tracker site recovery cycle.

    If you choose to, you can stop and start the tracker DB2 several times before completing a recovery cycle.

Using the RECOVER utility to establish a recovery cycle

When full image copies of all the data at the primary site are available to your tracker site, you can use the RECOVER utility to periodically apply the active log, archive logs, and BSDS from the primary site to the tracker site.

The following procedure establishes a recovery cycle at your tracker site:

1. While your primary site continues its usual workload, send a copy of the primary site's active log, archive logs, and BSDS to the tracker site.

   Send full image copies for the following objects:

   - Table spaces or partitions that are reorganized, loaded, or repaired with the LOG NO option after the latest recovery cycle.
   - Objects that, after the latest recovery cycle, have been recovered to a point in time.
   - SYSUTILX. Send a full image copy to DSNDB01.SYSUTILX for normal (full image copy and log) recoveries. For LOGONLY recoveries, create a DSN1COPY of DSNDB01.SYSUTILX.

   DB2 does not write SYSLGRNX entries for DSNDB01.SYSUTILX, which can lead to long recovery times at the tracker site. In addition, SYSUTILX and its indexes are updated during the tracker cycle when you run your recoveries. Because SYSUTILX must remain consistent with the SYSUTILX at the primary site, the tracker cycle updates must be discarded before the next tracker cycle.

   Recommendation: If you are taking incremental image copies, run the MERGECOPY utility at the primary site before sending the copy to the tracker site.
2. At the tracker site, restore the BSDS that was received from the primary site.

- Locate the BSDS in the latest archive log that is now at the tracker site.
- Use the change log inventory utility (DSNJU003) to register this archive log in the archive log inventory of the new BSDS.
- Use the change log inventory utility (DSNJU003) to register the primary site's active log in the new BSDS.

For more details about restoring the BSDS, see step 6 of "Remote site recovery from a disaster at the local site" on page 558.

3. Use the change log inventory utility (DSNJU003) with the following CRESTART control statement:

   ```
   CRESTART CREATE,ENDRBA=nnnnnnnnn000,FORWARD=NO,BACKOUT=NO
   ```

   In this control statement, *nnnnnnnnn*000 equals ENDRBA + 1 of the latest archive log. You must not specify STARTRBA, because you cannot cold start or skip logs in a tracker system.

   Data sharing

   If you are recovering a data sharing group, you **must** use the following CRESTART control statement on all members of the data sharing group. The ENDLRSN value must be the same for all members.

   ```
   CRESTART CREATE,ENDLRSN=nnnnnnnnnnnn,FORWARD=NO,BACKOUT=NO
   ```

   In this control statement, *nnnnnnnnnnnn* is the lowest ENDLRSN of all the members to be read during restart. You must specify one of the following values for the ENDLRSN:

   - If you receive the ENDLRSN from the output of the print log map utility (DSNJU004) or from the console logs using message DSNJ003I, you must use ENDLRSN-1 as the input to the conditional restart.
   - If you receive the ENDLRSN from the output of the DSN1LOGP utility (DSN1213I message), you can use the value that is displayed.

   The ENDLRSN or ENDRBA value indicates the end log point for data recovery and for truncating the archive log. With ENDLRSN, the *missing* log records between the lowest and highest ENDLRSN values for all the members are applied during the next recovery cycle.

4. If the tracker site is a data sharing group, delete all DB2 coupling facility structures before restarting the tracker members.
5. At the tracker site, restart DB2 to begin a **tracker site recovery cycle**.

   Data sharing

   For data sharing, restart **every** member of the data sharing group.

6. At the tracker site, submit RECOVER jobs to recover database objects. Run the RECOVER utility with the LOGONLY option on all database objects that do not require recovery from an image copy. See "Media failures during LOGONLY recovery" on page 574 for information about what to do if a media failure occurs during LOGONLY recovery.

   You must recover database objects as the following procedure specifies:

   a. Restore the full image copy or DSN1COPY of SYSUTILX.

      Recovering SYSUTILX: If you are doing a LOGONLY recovery on SYSUTILX from a previous DSN1COPY backup, make another DSN1COPY

copy of that table space after the LOGONLY recovery is complete and before recovering any other catalog or directory objects.

After you recover SYSUTILX and either recover or rebuild its indexes, and before recovering other system and user table spaces, find out what utilities were running at the primary site.

b. Recover the catalog and directory.

See *DB2 Utility Guide and Reference* for information about the order of recovery for the catalog and directory objects.

User-defined catalog indexes: Unless you require them for catalog query performance, it is not necessary to rebuild user-defined catalog indexes until the tracker DB2 becomes the takeover DB2. However, if you are recovering user-defined catalog indexes, do the recover in this step.

c. If needed, recover other system data such as the data definition control support table spaces and the resource limit facility table spaces.

d. Recover user data and, optionally, rebuild your indexes.

It is not necessary to rebuild indexes unless you intend to run dynamic queries on the data at the tracker site.

Because this is a tracker site, DB2 stores the conditional restart ENDRBA or ENDLRSN in the page set after each recovery completes successfully. By storing the log truncation value in the page set, DB2 ensures that it does not skip any log records between recovery cycles.

7. Enter DISPLAY UTIL(*) for a list of currently running utilities.
8. Run the DIAGNOSE utility with the DISPLAY SYSUTIL statement to find out the names of the object on which the utilities are running. Installation SYSOPR authority is required.
9. Perform the following actions for objects at the tracker site on which utilities are pending. Restrictions apply to these objects with because DB2 prevents you from using the TERM UTIIL command to remove pending statuses at a tracker site.
   - If a LOAD, REORG, REPAIR, or COPY is in progress on any catalog or directory object at the primary site, shut down DB2. You cannot continue recovering through the list of catalog and directory objects. Therefore, you cannot recover any user data. At the next recovery cycle, send a full image copy of the object from the primary site. At the tracker site, use the RECOVER utility to restore the object.
   - If a LOAD, REORG, REPAIR, or COPY utility is in progress on any user data, at the next recovery cycle, send a full image copy of the object from the primary site. At the tracker site, use the RECOVER utility to restore the object.
   - If an object is in the restart pending state, use LOGONLY recovery to recover the object when that object is no longer in restart pending state.

Data sharing

If read/write shared data (GPB-dependent data) is in the advisory recovery pending state, the tracker DB2 performs recovery processing. Because the tracker DB2 always performs a conditional restart, the postponed indoubt units of recovery are not recognized after the tracker DB2 restarts.

10. After all recovery has completed at the tracker site, shut down the tracker site DB2. This is the end of the tracker site recovery cycle. If you choose to, you can stop and start the tracker DB2 several times before completing a recovery cycle.

Media failures during LOGONLY recovery

As documented in *DB2 Utility Guide and Reference*, if there is an I/O error during a LOGONLY recovery, recover the object using the image copies and logs after you correct the media failure.

If an entire volume is corrupted and you are using DB2 storage groups, you cannot use the ALTER STOGROUP statement to remove the corrupted volume and add another as is documented for a non-tracker system. Instead, you must remove the corrupted volume and reinitialize another volume with the same volume serial number before you invoke the RECOVER utility for all table spaces and indexes on that volume.

Maintaining the tracker site

It is recommended that the tracker site and primary site be at the same maintenance level to avoid unexpected problems. Between recovery cycles, you can apply maintenance as you normally do, by stopping and restarting the DB2 subsystem or DB2 member.

If a tracker site fails, you can restart it normally.

Because bringing up your tracker site as the takeover site destroys the tracker site environment, you should save your complete tracker site prior to takeover site testing. The tracker site can then be restored after the takeover site testing, and the tracker site recovery cycles can be resumed.

Data sharing group restarts

During recovery cycles, the first member that comes up puts the ENDLRSN value in the shared communications area (SCA) of the coupling facility. If an SCA failure occurs during a recovery cycle, you must go through the recovery cycle again, using the same ENDLRSN value for your conditional restart.

Making the tracker site the takeover site

If a disaster occurs at the primary site, the tracker site must become the *takeover* site. After the takeover site is restarted, you must apply log data or image copies that were enroute when the disaster occurred. The procedure to make the tracker site the takeover depends on whether you use RESTORE SYSTEM LOGONLY or the RECOVER utility in your tracker site recovery cycles.

Recovering at a tracker site that uses RESTORE SYSTEM LOGONLY: If you use RESTORE SYSTEM LOGONLY in the recovery cycles at the tracker site, use the following procedure after a disaster to make the tracker site the takeover site:

1. If log data for a recovery cycle is enroute or is available but has not yet been used in a recovery cycle perform the procedure in "Using RESTORE SYSTEM LOGONLY to establish a recovery cycle" on page 569.
2. Ensure that the TRKSITE NO subsystem parameter is specified.
3. For scenarios other than data sharing, continue with the next step.

Data sharing
If this is a data sharing system, delete the coupling facility structures.

4. Start DB2 at the same RBA or ENDLSRN that you used in the most recent tracker site recovery cycle. Specify FORWARD=YES and BACKOUT=YES in the CRESTART statement (this takes care of uncommitted work).
5. Restart the objects that are in GRECP or LPL status using the following START DATABASE command:

   ```
   START DATABASE(*) SPACENAM(*)
   ```

6. If you used to DSN1COPY to create a copy of SYSUTILX in the last recovery cycle, use DSN1COPY to restore that copy.
7. Terminate any in-progress utilities by using the following procedure:
   a. Enter the command DISPLAY UTIL(*).
   b. Run the DIAGNOSE utility with DISPLAY SYSUTIL to get the names of objects on which utilities are being run.
   c. Terminate in-progress utilities using the command TERM UTIL(*).

      See "What to do about utilities in progress" on page 565 for more information about how to terminate in-progress utilities and how to recover an object on which a utility was running.
8. Rebuild indexes, including IBM and user-defined indexes on the DB2 catalog and user-defined indexes on table spaces.

Recovering at a tracker site that uses the RECOVER utility: If you use the RECOVER utility in the recovery cycles at your tracker site, use the following procedure after a disaster to make the tracker site the takeover site:

1. Restore the BSDS and register the archive log from the last archive you received from the primary site.
2. For scenarios other than data sharing, continue with the next step.

Data sharing
If this is a data sharing system, delete the coupling facility structures.

3. Ensure that the DEFER ALL and TRKSITE NO subsystem parameters are specified.
4. If this is a non-data-sharing DB2, the log truncation point varies depending on whether you have received more logs from the primary site since the last recovery cycle:
   - If you received no more logs from the primary site:

     Start DB2 using the same ENDRBA that you used on the last tracker cycle. Specify FORWARD=YES and BACKOUT=YES (this takes care of uncommitted work). If you have fully recovered the objects during the previous cycle, then they are current except for any objects that had outstanding units of recovery during restart. Because the previous cycle specified NO for FORWARD and BACKOUT and you have now specified YES, affected data sets are placed in LPL. Restart the objects that are in LPL status using the following START DATABASE command:

     ```
     START DATABASE(*) SPACENAM(*)
     ```

After you issue the command, all table spaces and indexes that were previously recovered are now current. Remember to rebuild any indexes that were not recovered during the previous tracker cycle, including user-defined indexes on the DB2 catalog.

- If you received more logs from the primary site:

  Start DB2 using the truncated RBA *nnnnnnnnn*000, which is the ENDRBA + 1 of the latest archive log. Specify FORWARD=YES and BACKOUT=YES. Run your recoveries as you did during recovery cycles.

Data sharing

You must restart **every** member of the data sharing group, the following CRESTART statement:

```
CRESTART CREATE,ENDLRSN=nnnnnnnnnnnn,FORWARD=YES,BACKOUT=YES
```

In this statement, *nnnnnnnnnnnn* is the LRSN of the last log record to be used during restart. See step 3 of "Using the RECOVER utility to establish a recovery cycle" on page 571 for more information about determining this value. The takeover DB2s must specify conditional restart with a common ENDLRSN value to allow all remote members to logically truncate the logs at a consistent point.

5. As described for a tracker recovery cycle, recover SYSUTILX from an image copy from the primary site, or from a previous DSN1COPY taken at the tracker site.
6. Terminate any in-progress utilities by using the following procedure:
   a. Enter the command DISPLAY UTIL(*).
   b. Run the DIAGNOSE utility with DISPLAY SYSUTIL to get the names of objects on which utilities are being run.
   c. Terminate in-progress utilities using the command TERM UTIL(*).

      See "What to do about utilities in progress" on page 565 for more information about how to terminate in-progress utilities and how to recover an object on which a utility was running.
7. Continue with your recoveries either with the LOGONLY option or image copies. Do not forget to rebuild indexes, including IBM and user-defined indexes on the DB2 catalog and user-defined indexes on table spaces.

Using data mirroring

Data mirroring is the automatic replication of current data from your primary site to a secondary site. You can use this secondary site for your recovery site to recover after a disaster without the need to restore DB2 image copies or apply DB2 logs to bring DB2 data to the current point in time.

Important: The scenarios and procedures for data mirroring are intended for environments that mirror an entire DB2 subsystem or data sharing group, which includes the catalog, directory, user data, BSDS, and active logs. You must mirror all volumes in such a way that they terminate at exactly the same point. You can achieve this final condition using consistency groups, which are described in "Consistency groups" on page 578.

The rolling disaster

In a real disaster, your local site gradually and intermittently fails over a number of seconds. This kind of DB2 failure is known as a *rolling disaster*.

To use data mirroring for disaster recovery, you must mirror data from your local site with a method that does not reproduce a rolling disaster at your recovery site. To recover DB2 with data integrity, you must use volumes that end at a consistent point in time for each DB2 subsystem or data sharing group. Mirroring a rolling disaster causes volumes at your recovery site to end over a span of time rather than at one single point.

Figure 55 shows how a rolling can cause data to become inconsistent between two subsystems.

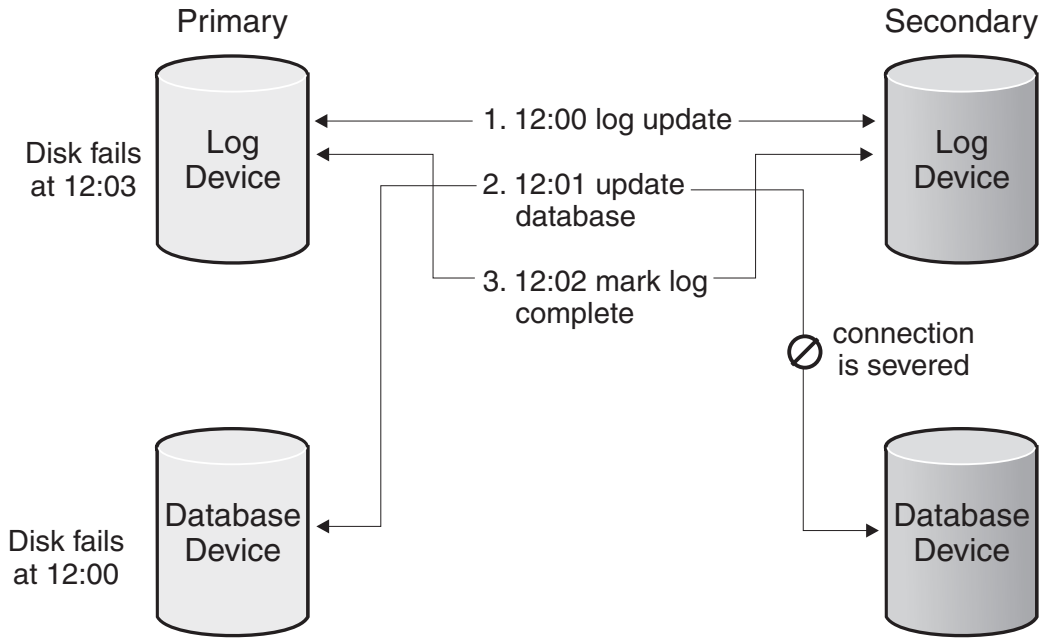


Figure 55. Data inconsistency caused by a rolling disaster

Example: In a rolling disaster, the following events at the primary site cause data inconsistency at your recovery site. This data inconsistency example follows the same scenario that Figure 55 depicts.

1. A table space is updated in the buffer pool (11:58).
2. The log record is written to disk on logical storage subsystem 1 (12:00).
3. Logical storage subsystem 2 fails (12:01).
4. The update to the table space is externalized to logical storage subsystem 2 but is not written because subsystem 2 failed (12:02).
5. The log record that table space update was made is written to disk on logical storage subsystem 1 (12:03).
6. Logical storage subsystem 1 fails (12:04).

Because the logical storage subsystems do not fail at the same point in time, they contain inconsistent data. In this scenario, the log indicates that the update is applied to the table space, but the update is not applied to the data volume that holds this table space.

Attention: Any disaster recovery solution that uses data mirroring must guarantee that all volumes at the recovery site contain data for the same point in time.

Consistency groups

Generally, a *consistency group* is a collection of volumes that contain consistent, related data. This data can span logical storage subsystems and disk subsystems. For DB2 specifically, a consistency group contains an entire DB2 subsystem or a DB2 data sharing group. The following DB2 elements comprise a consistency group:

- Catalog tables
- Directory tables
- BSDS
- Logs
- All user data
- ICF catalogs

Additionally, all objects within a consistency group must represent the same point in time in at least one of the following situations:

- At the time of a backup
- After a normal DB2 restart

You can use various methods to create consistency groups. The following DB2 services enable you to create consistency groups:

- XRC I/O timestamping and system data mover
- FlashCopy consistency groups
- GDPSfreeze policies
- The DB2 SET LOG SUSPEND command
- The DB2 BACKUP SYSTEM utility

When a rolling disaster strikes your primary site, consistency groups guarantee that all volumes at the recovery site contain data for the same point in time. In a data mirroring environment, you must perform both of the following actions for each consistency group that you maintain:

- Mirror data to the secondary volumes in the same sequence that DB2 writes data to the primary volumes.

  In many processing situations, DB2 must complete one write operation before it begins another write operation on a different disk group or a different storage server. A write operation that depends on a previous write operation is called a *dependent write*. Do not mirror a dependent write if you have not mirrored the write operation on which the dependent write depends. If you mirror data out of sequence, your recovery site will contain inconsistent data that you cannot use for disaster recovery.

- Temporarily suspend and queue write operations to create a group point of consistency when an error occurs between any pair of primary and secondary volumes.

  When an error occurs that prevents the update of a secondary volume in a single-volume pair, this error might mark the beginning of a rolling disaster. To prevent your secondary site from mirroring a rolling disaster, you must suspend and queue data mirroring with the following steps after a write error between any pairs:

  1. Suspend and queue all write operations in the volume pair that experiences a write error.
  2. Invoke automation that temporarily suspends and queues data mirroring to all your secondary volumes.
  3. Save data at the secondary site at a point of consistency.
  4. If a rolling disaster does not strike your primary site, resume normal data mirroring after some amount of time that you have defined. If a rolling

disaster does strike your primary site, follow the recovery procedure in "Recovering in a data mirroring environment."

Recovering in a data mirroring environment

In a data mirroring environment, you can use a general procedure to recover at your secondary site from a disaster at your primary site. This procedure applies to all DB2 data mirroring scenarios except those that use Extended Remote Copy (XRC). This general procedure is valid only if you have established and maintained consistency groups before the disaster struck the primary site. If you use data mirroring to recover, you must recover your entire DB2 subsystem or data sharing group with data mirroring.

You do not need to restore DB2 image copies or apply DB2 logs to bring DB2 data to the current point in time when you use data mirroring. However, you might need image copies at the recovery site if the LOAD or RECOVER utilities were active at the time of the disaster.

To recover at the secondary site after a disaster, use the following procedure:

1. IPL all z/OS images at your recovery site that correspond to the z/OS images that you lost at your primary site.
2. For scenarios other than data sharing, continue with the next step.

Data sharing

For data sharing groups, you must remove old information from the coupling facility.

a. Enter the following z/OS command to display the structures for this data sharing group:

```
D XCF,STRUCTURE,STRNAME=grpname*
```

b. For group buffer pools and the lock structure, enter the following command to force off the connections in those structures:

```
SETXCF FORCE,CONNECTION,STRNAME=strname,CONNAME=ALL
```

c. Delete all the DB2 coupling facility structures by using the following command for each structure:

```
SETXCF FORCE,STRUCTURE,STRNAME=strname
```

3. If you are using the distributed data facility, set LOCATON and LUNAME in the BSDS to values that are specific to your new primary site. To set LOCATION and LUNAME run the stand-alone utility DSNJU003 (change log inventory) with the following control statement:

```
DDF LOCATION=locname, LUNAME=luname
```

4. Start all DB2 members using local DSNZPARM data sets and perform a normal restart.

Data sharing

For data sharing groups, DB2 performs group restart. Shared data sets are set to GRECP (group buffer pool RECOVER-pending) status, and pages are added to the LPL (logical page list).

5. For scenarios other than data sharing, continue to the next step.

Data sharing

For data sharing groups, perform the following procedure:

a. Display all data sets with GRECP or LPL (logical page list) status with the following DB2 command:

```
-DISPLAY DATABASE(*) SPACENAM(*) RESTRICT(GRECP, LPL) LIMIT(*)
```

Record the output that this command generates.

b. Start the DB2 directory with the following DB2 command:

```
-START DATABASE(DSNDB01) SPACENAM(*)
```

c. Start the DB2 catalog with the following DB2 command:

```
-START DATABASE(DSNDB06) SPACENAM(*)
```

6. Use the following DB2 command to display all utilities that the failure interrupted:

```
-DISPLAY UTILITY(*)
```

If utilities are pending, record the output that this command and continue to step 7. You cannot restart utilities at a recovery site. You will terminate these utilities in step number 8. If no utilities are pending, continue to step number 9.

7. Use the DIAGNOSE utility to access the SYSUTILX directory table. This directory table is not listed in the *DB2 SQL Reference* because you can access it only when you use the DIAGNOSE utility. (The DIAGNOSE utility is normally intended to be used under the direction of IBM Software Support.)

Use the following control statement when you run the DIAGNOSE utility:

```
DIAGNOSE DISPLAY SYSUTILX
END DIAGNOSE
```

Record the phase in which each pending utility was interrupted, and record the object on which each utility was operating.

8. Terminate all pending utilities with the following command:

```
-TERM UTILITY(*)
```

9. For scenarios other than data sharing, continue to the next step.

Data sharing

For data sharing groups, use the following START DATABASE command on each database that contains objects in GRECP or LPL status:

```
-START DATABASE(database) SPACENAM(*)
```

When you use the START DATABASE command to recover objects, you do not need to provide DB2 with image copies.

Tip: Use up to 10 START DATABASE commands for each DB2 subsystem to increase the speed at which DB2 completes this operation. Multiple commands that run in parallel complete faster than a single command that specifies the same databases.

10. Start all remaining database objects with the following START DATABASE command:

```
START DATABASE(*) SPACENAM(*)
```

11. For each object that the LOAD utility has placed in a restrictive status, take one of the following actions:
    - If the object was a target of a LOAD utility control statement that specified SHRLEVEL CHANGE, restart the LOAD utility on this object at your convenience. This object contains valid data.
    - If the object was a target of a LOAD utility control statement that specified SHRLEVEL REFERENCE and the LOAD job was interrupted before the RELOAD phase, rebuild the indexes on this object.
    - If the object was a target of a LOAD utility control statement that specified SHRLEVEL REFERENCE and the LOAD job was interrupted during or after the RELOAD phase, you must recover this object to a point in time before this utility was run.
    - Otherwise, recover the object to a point in time before the LOAD job was run.
12. For each object that the REORG utility has placed in a restrictive status, take one of the following actions:
    - When the object was a target of a REORG utility control statement that specified SHERLEVEL NONE:
      - If the REORG job was interrupted before the RELOAD phase, no further action is required. This object contains valid data, and the indexes on this object are valid.
      - If the REORG job was interrupted during the RELOAD phase, you must recover this object to a point in time before this utility was run.
      - If the REORG job was interrupted after the RELOAD phase, rebuild the indexes on the object.
    - When the object was a target of a REORG utility control statement that does not specify SHERLEVEL NONE:
      - If the REORG job was interrupted before the SWITCH phase, no further action is required. This object contains valid data, and the indexes on this object are valid.
      - If the REORG job was interrupted during the SWITCH phase, no further action is required. This object contains valid data, and the indexes on this object are valid.
      - If the REORG job was interrupted after the SWITCH phase, you might need to rebuild non-partitioned secondary indexes.

Recovering with Extended Remote Copy

One method that ensures that data volumes remain consistent at your recovery site is by using Extended Remote Copy (XRC). In XRC remote mirroring, the DFSMS Advanced Copy services function automatically replicates current data from your primary site to a secondary site and establishes consistency groups. For specific information about extended remote copy or the extended remote copy environment, see *DFSMSdfp Advanced Services, SC26-7400*.

To recover at an XRC secondary site after a disaster, use the following procedure:

1. Issue the XEND XRC TSO command to end the XRC session.
2. Issue the XRECOVER XRC TSO command. This command changes your secondary site to your primary site and applies the XRC journals to recover data that was in transit when your primary site failed.
3. Complete the procedure in “Recovering in a data mirroring environment” on page 579.

Resolving indoubt threads

This section describes problem scenarios involving indoubt threads resulting from the following types of error conditions:

"Communication failure recovery" on page 583
"Making a heuristic decision" on page 584
"Recovery from an IMS outage that results in an IMS cold start" on page 585
"Recovery from a DB2 outage at a requester that results in a DB2 cold start" on page 586
"Recovery from a DB2 outage at a server that results in a DB2 cold start" on page 589
"Correcting a heuristic decision" on page 589

All scenarios are developed in the context presented in "Description of the recovery environment." System programmer, operator, and database administrator actions are indicated for the examples as appropriate. In these descriptions, the term *administrator* refers to the database administrator (DBA) if not otherwise specified.

Description of the recovery environment

Configuration

The configuration is composed of four systems at three geographic locations: Seattle (SEA), San Jose (SJ) and Los Angeles (LA). The system descriptions are as follows:

- DB2 subsystem at Seattle, Location name = IBMSEADB20001, Network name = IBM.SEADB21
- DB2 subsystem at San Jose, Location name = IBMSJ0DB20001 Network name = IBM.SJDB21
- DB2 subsystem at Los Angeles, Location name = IBMLA0DB20001 Network name = IBM.LADB21
- IMS subsystem at Seattle, Connection name = SEAIMS01

Applications

The following IMS and TSO applications are running at Seattle and accessing both local and remote data.

- IMS application, IMSAPP01, at Seattle, accessing local data and remote data by DRDA access at San Jose, which is accessing remote data on behalf of Seattle by DB2 private protocol access at Los Angeles.
- TSO application, TSOAPP01, at Seattle, accessing data by DRDA access at San Jose and at Los Angeles.

Threads

The following threads are described and keyed to Figure 56 on page 583. Data base access threads (DBAT) access data on behalf of a thread (either allied or DBAT) at a remote requester.

- Allied IMS thread **A** at Seattle accessing data at San Jose by DRDA access.
  - DBAT at San Jose accessing data for Seattle by DRDA access **1** and requesting data at Los Angeles by DB2 private protocol access **2** .
  - DBAT at Los Angeles accessing data for San Jose by DB2 private protocol access **2** .
- Allied TSO thread **B** at Seattle accessing local data and remote data at San Jose and Los Angeles, by DRDA access.
  - DBAT at San Jose accessing data for Seattle by DRDA access **3** .

– DBAT at Los Angeles accessing data for Seattle by DRDA access **4** .

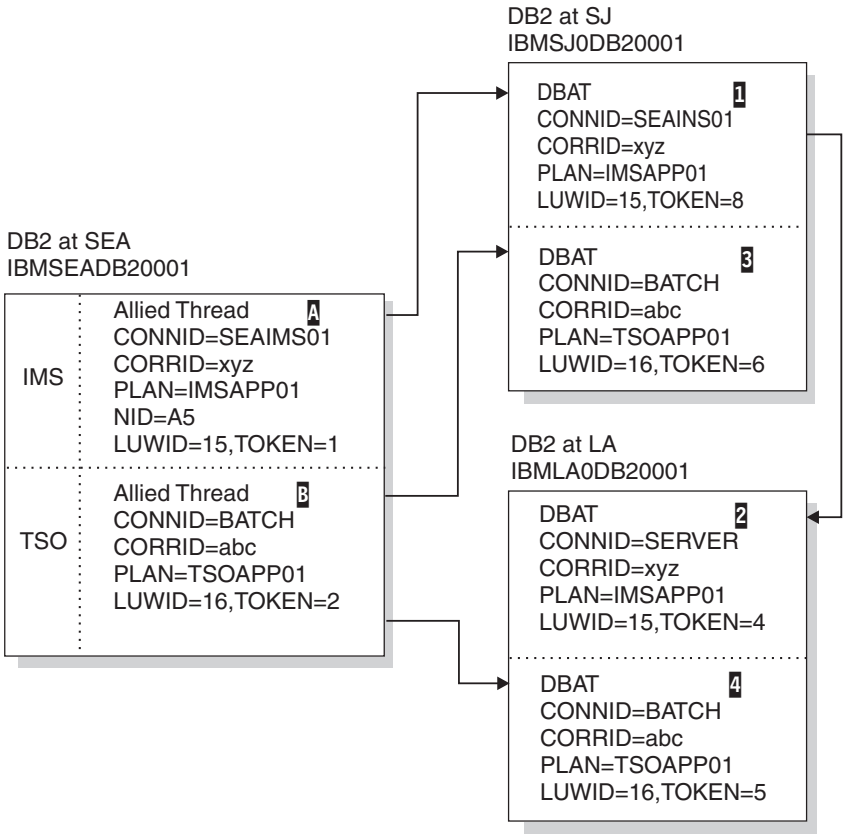


Figure 56. Resolving indoubt threads. Results of issuing DIS THD TYPE(ACTIVE) at each DB2 system.

The results of issuing the DISPLAY THREAD TYPE(ACTIVE) command to display the status of threads at all DB2 locations are summarized in the boxes of Figure 56. The logical unit of work IDs (LUWIDs) have been shortened for readability:

- LUWID=15 would be IBM.SEADB21.15A86A876789.0010
- LUWID=16 would be IBM.SEADB21.16B57B954427.0003

For the purposes of this section, assume that both applications have updated data at all DB2 locations. In the following problem scenarios, the error occurs after the coordinator has recorded the commit decision, but before the affected participants have recorded the commit decision. These participants are therefore indoubt.

Communication failure recovery

Problem: A communication failure occurred between Seattle and Los Angeles after the DBAT at LA completed phase 1 of commit processing. At SEA, the TSO thread, LUWID=16 and TOKEN=2 **B** , cannot complete the commit with the DBAT at LA **4** .

Symptom: At SEA, NetView alert A006 is generated and message DSNL406 is displayed, indicating an indoubt thread at LA because of communication failure. At LA, alert A006 is generated and message DSNL405 is displayed, indicating a thread has entered indoubt state because of communication failure with SEA.

System action: At SEA, an IFCID 209 trace record is written. After the alert has been generated, and after the message has been displayed, the thread completes the commit, which includes the DBAT at SJ **3** . Concurrently, the thread is added to the list of threads for which the SEA DB2 has an indoubt resolution responsibility. The thread appears in a display thread report for indoubt threads. The thread also appears in a display thread report for active threads until the application terminates.

The TSO application is told that the commit succeeded. If the application continues and processes another SQL request, it is rejected with an SQL code indicating it must roll back before any more SQL requests can be processed. This is to insure that the application does not proceed with an assumption based upon data retrieved from LA, or with the expectation that cursor positioning at LA is still intact.

At LA, an IFCID 209 trace record is written. After the alert is generated and the message displayed, the DBAT **4** is placed into the indoubt state. All locks remain held until resolution occurs. The thread appears in a display thread report for indoubt threads.

The DB2 systems, at both SEA and LA, periodically attempt reconnecting and automatically resolving the indoubt thread. If the communication failure only affects the session being used by the TSO application, and other sessions are available, automatic resolution occurs in a relatively short time. At this time, message DSNL407 is displayed by both DB2 subsystems.

Operator action: If message DSNL407 or DSNL415 for the thread identified in message DSNL405 does not appear in a reasonable period of time, call the system programmer. A communication failure is making database resources unavailable.

System programmer action: Determine and correct the cause of the communication failure. When corrected, automatic resolution of the indoubt thread occurs within a short time. If the failure cannot be corrected for a long time, call the database administrator. The database administrator might want to make a heuristic decision to release the database resources held for the indoubt thread. See "Making a heuristic decision."

Making a heuristic decision

Problem: The indoubt thread at LA is holding database resources that are needed by other applications.

Symptom: Many symptoms can be present, including:

- Message DSNL405 indicating a thread in the indoubt state.
- A display thread report of active threads showing a larger than normal number of threads.
- A display thread report of indoubt threads continuing to show the same thread.
- A display database report with the LOCKS option showing a large number of threads waiting for the locks held by the indoubt thread.
- Some threads terminating due to time out.
- IMS and CICS transactions not completing.

Database administrator action: Determine whether to commit or abort the indoubt thread. First, determine the name of the commit coordinator for the indoubt thread. This is the location name of the DB2 subsystem at SEA, and is included in

the DB2 indoubt thread display report at LA. Then, have an authorized person at SEA perform one of the following actions:

- If the coordinator DB2 subsystem is active, or can be started, request a display thread report for indoubt threads, specifying the LUWID of the thread. (Remember that the token used at LA is different than the token used at SEA). *If there is no report entry for the LUWID, then the proper action is to abort. If there is an entry for the LUWID, it shows the proper action to take.*
- If the coordinator DB2 subsystem is not active and cannot be started, and if statistics class 4 was active when DB2 was active, search the SEA SMF data for an IFCID 209 event entry containing the indoubt LUWID. This entry indicates whether the commit decision was commit or abort.
- If statistics class 4 is not available, then run, at SEA, the DSN1LOGP utility requesting a summary report. The volume of log data to be searched can be restricted if you can determine the approximate SEA log RBA value in effect at the time of the communication failure. A DSN1LOGP entry in the summary report for the indoubt LUWID indicates whether the decision was commit or abort.

After determining the correct action to take, issue the RECOVER INDOUBT command at the LA DB2 subsystem, specifying the LUWID and the correct action.

System action: Issuing the RECOVER INDOUBT command at LA results in committing or aborting the indoubt thread. Locks are released. The thread does not disappear from the indoubt thread display until resolution with SEA is completed. The recover indoubt report shows that the thread is either committed or aborted by a heuristic decision. An IFCID 203 trace record is written, recording the heuristic action.

Recovery from an IMS outage that results in an IMS cold start

Problem: The abnormal termination of IMS has left one allied thread **A** at the SEA DB2 subsystem indoubt. This is the thread having LUWID=15. Because the SEA DB2 still has effective communication with the DB2 subsystem at SJ, the LUWID=15 DBAT **1** at this system is waiting for the SEA DB2 to communicate the final decision and is not aware that IMS has failed. Also, the LUWID=15 DBAT at LA **2** which is connected to SJ is also waiting for SJ to communicate the final decision. This cannot be done until SEA communicates the decision to SJ.

Symptom: When IMS is cold started, and later reconnects with the SEA DB2 subsystem, IMS is not able to resolve the indoubt thread with DB2. Message DSNM004I is displayed at the IMS master terminal. This is the same process as described in “Resolution of indoubt units of recovery” on page 522.

System action: This is the same process as described in “Resolution of indoubt units of recovery” on page 522.

System programmer action: This is the same process as described in “Resolution of indoubt units of recovery” on page 522.

When the indoubt thread at the SEA DB2 subsystem is resolved by issuing the RECOVER INDOUBT command, completion of the two-phase commit process with the DB2 subsystems at SJ and LA occurs, and the unit of work commits or aborts.

Operator action: This is the same process as described in “Resolution of indoubt units of recovery” on page 522.

Recovery from a DB2 outage at a requester that results in a DB2 cold start

Problem: The abnormal termination of the SEA DB2 has left the two DBATs at SJ **1**, **3** and the LUWID=16 DBAT at LA **4** indoubt. The LUWID=15 DBAT at LA **2**, connected to SJ, is waiting for the SJ DB2 to communicate the final decision.

The IMS subsystem at SEA is operational and has the responsibility of resolving indoubt units with the SEA DB2.

Symptom: The DB2 subsystem at SEA is started with a conditional restart record in the BSDS indicating a cold start:

- When the IMS subsystem reconnects, it attempts to resolve the indoubt thread identified in IMS as NID=A5. IMS has a resource recovery element (RRE) for this thread. The SEA DB2 informs IMS that it has no knowledge of this thread. IMS does not delete the RRE and it can be displayed by using the IMS DISPLAY OASN command. The SEA DB2 also:
  - Generates message DSN3005 for each IMS RRE for which DB2 has no knowledge.
  - Generates an IFCID 234 trace event.
- When the DB2 subsystems at SJ and LA reconnect with SEA, each detects that the SEA DB2 has cold started. Both the SJ DB2 and the LA DB2:
  - Display message DSNL411.
  - Generate alert A001.
  - Generate an IFCID 204 trace event.
- A display thread report of indoubt threads at both the SJ and LA DB2 subsystems shows the indoubt threads and indicates that the coordinator has cold started.

System action: The DB2 subsystem at both SJ and LA accept the cold start connection from SEA. Processing continues, waiting for a heuristic decision to resolve the indoubt threads.

System programmer action: Call the database administrator.

Operator action: Call the database administrator.

Database administrator action: At this point, neither the SJ nor the LA administrator know if the SEA coordinator was a participant of another coordinator. In this scenario, the SEA DB2 subsystem originated LUWID=16. However, it was a participant for LUWID=15, being coordinated by IMS.

Also not known to the administrator at LA is the fact that SEA distributed the LUWID=16 thread to SJ where it is also indoubt. Likewise, the administrator at SJ does not know that LA has an indoubt thread for the LUWID=16 thread. It is important that both SJ and LA make the same heuristic decision. It is also important that the administrators at SJ and LA determine the originator of the two-phase commit.

The recovery log of the originator indicates whether the decision was commit or abort. The originator might have more accessible functions to determine the decision. Even though the SEA DB2 cold started, you might be able to determine the decision from its recovery log. Or, if the failure occurred before the decision was recorded, you might be able to determine the name of the coordinator, if the

SEA DB2 was a participant. A summary report of the SEA DB2 recovery log can be provided by execution of the DSN1LOGP utility.

The LUWID contains the name of the logical unit (LU) where the distributed logical unit of work originated. This logical unit is most likely in the system that originated the two-phase commit.

If an application is distributed, any distributed piece of the application can initiate the two-phase commit. In this type of application, the originator of two-phase commit can be at a different system than that identified by the LUWID. With DB2 private protocol access, the two-phase commit can flow only from the system containing the application that initiates distributed SQL processing. In most cases, this is where the application originates.

The administrator must determine if the LU name contained in the LUWID is the same as the LU name of the SEA DB2 subsystem. If this is not the case (it is the case in this example), then the SEA DB2 is a participant in the logical unit of work, and is being coordinated by a remote system. You must communicate with that system and request that facilities of that system be used to determine if the logical unit of work is to be committed or aborted.

If the LUWID contains the LU name of the SEA DB2 subsystem, then the logical unit of work originated at SEA and is either an IMS, CICS, TSO, or BATCH allied thread of the SEA DB2. The display thread report for indoubt threads at a DB2 participant includes message DSNV458 if the coordinator is remote. This line provides external information provided by the coordinator to assist in identifying the thread. A DB2 coordinator provides the following identifier:

```
connection-name.correlation-id
```

where *connection-name* is:

- SERVER: the thread represents a remote application to the DB2 coordinator and uses DRDA access.
- BATCH: the thread represents a local batch application to the DB2 coordinator.

Anything else represents an IMS or CICS connection name. The thread represents a local application and the commit coordinator is the IMS or CICS system using this connection name.

In our example, the administrator at SJ sees that both indoubt threads have a LUWID with the LU name the same as the SEA DB2 LU name, and furthermore, that one thread (LUWID=15) is an IMS thread and the other thread (LUWID=16) is a batch thread. The LA administrator sees that the LA indoubt thread (LUWID=16) originates at SEA DB2 and is a batch thread.

The originator of a DB2 batch thread is DB2. To determine the commit or abort decision for the LUWID=16 indoubt threads, the SEA DB2 recovery log must be analyzed, if it can be. The DSN1LOGP utility must be executed against the SEA DB2 recovery log, looking for the LUWID=16 entry. There are three possibilities:

1. No entry is found. That portion of the DB2 recovery log was not available.
2. An entry is found but incomplete.
3. An entry is found and the status is committed or aborted.

In the third case, the heuristic decision at SJ and LA for indoubt thread LUWID=16 is indicated by the status indicated in the SEA DB2 recovery log. In the other two cases, the recovery procedure used when cold starting DB2 is important. If

recovery was to a previous point in time, the correct action is to abort. If recovery included repairing the SEA DB2 database, the SEA administrator might know what decision to make.

The recovery logs at SJ and LA can help determine what activity took place. If it can be determined that updates were performed at either SJ, LA, or both (but not SEA), then if both SJ and LA make the same heuristic action, there should be no data inconsistency. If updates were also performed at SEA, then looking at the SEA data might help determine what action to take. In any case, both SJ and LA should make the same decision.

For the indoubt thread with LUWID=15 (the IMS coordinator), there are several alternative paths to recovery. The SEA DB2 has been restarted. When it reconnects with IMS, message DSN3005 is issued for each thread that IMS is trying to resolve with DB2. The message indicates that DB2 has no knowledge of the thread that is identified by the IMS assigned NID. The outcome for the thread, commit or abort, is included in the message. Trace event IFCID=234 is also written to statistics class 4 containing the same information.

If there is only one such message, or one such entry in statistics class 4, then the decision for indoubt thread LUWID=15 is known and can be communicated to the administrator at SJ. If there are multiple such messages, or multiple such trace events, you must match the IMS NID with the network LUWID. Again, DSN1LOGP should be used to analyze the SEA DB2 recovery log if possible. There are now four possibilities:

1. No entry is found. That portion of the DB2 recovery log was not available.
2. An entry is found but incomplete because of lost recovery log.
3. An entry is found and the status is indoubt.
4. An entry is found and the status is committed or aborted.

In the fourth case, the heuristic decision at SJ for the indoubt thread LUWID=15 is determined by the status indicated in the SEA DB2 recovery log. If an entry is found whose status is indoubt, DSN1LOGP also reports the IMS NID value. The NID is the unique identifier for the logical unit of work in IMS and CICS. Knowing the NID allows correlation to the DSN3005 message, or to the 234 trace event, which provides the correct decision.

If an incomplete entry is found, the NID may or may not have been reported by DSN1LOGP. If it was, use it as previously discussed. If no NID is found, or the SEA DB2 has not been started, or reconnecting to IMS has not occurred, then the *correlation-id* used by IMS to correlate the IMS logical unit of work to the DB2 thread must be used in a search of the IMS recovery log. The SEA DB2 provided this value to the SJ DB2 when distributing the thread to SJ. The SJ DB2 displays this value in the report generated by DISPLAY THREAD TYPE(INDOUBT).

For IMS, the *correlation-id* is:

```
pst#.psbname
```

In CICS, the *correlation-id* consists of four parts:

```
Byte 1 - Connection Type - G=Group, P=Pool
Byte 2 - Thread Type - T=transaction, G=Group, C=Command
Bytes 3-4 - Thread Number
Bytes 5–8 - Transaction-id
```

Recovery from a DB2 outage at a server that results in a DB2 cold start

Problem: This problem is similar to "Recovery from a DB2 outage at a requester that results in a DB2 cold start" on page 586. If the DB2 subsystem at SJ is cold started instead of the DB2 at SEA, then the LA DB2 has the LUWID=15 **2** thread indoubt. The administrator would see that this thread did not originate at SJ, but did originate at SEA. To determine the commit or abort action, the LA administrator would request that DISPLAY THREAD TYPE(INDOUBT) be issued at the SEA DB2, specifying LUWID=15. IMS would not have any indoubt status for this thread, because it would complete the two-phase commit process with the SEA DB2.

As described in "Communication failure recovery" on page 583, the DB2 at SEA tells the application that the commit succeeded.

When a participant cold starts, a DB2 coordinator continues to include in the display of indoubt threads all committed threads where the cold starting participant was believed to be indoubt. These entries must be explicitly purged by issuing the RESET INDOUBT command. If a participant has an indoubt thread that cannot be resolved because of coordinator cold start, it can request a display of indoubt threads at the DB2 coordinator to determine the correct action.

Correcting a heuristic decision

Problem: Assume the conditions of "Communication failure recovery" on page 583. The LA administrator is called to make a heuristic decision and decides to abort the indoubt thread with LUWID=16. The decision is made without communicating with SEA to determine the proper action. The thread at LA is aborted, while the threads at SEA and SJ are committed. Processing continues at all systems. DB2 at SEA has indoubt resolution responsibility with LA for LUWID=16.

Symptom: When the DB2 at SEA reconnects with the DB2 at LA, indoubt resolution occurs for LUWID=16. Both systems detect heuristic damage and both generate alert A004; each writes an IFCID 207 trace record. Message DSNL400 is displayed at LA and message DSNL403 is displayed at SEA..

System action: Processing continues. Indoubt thread resolution responsibilities have been fulfilled and the thread completes at both SJ and LA.

System programmer action: Call the database administrator.

Operator action: Call the database administrator.

Database administrator action: Correct the damage.

This is not an easy task. Since the time of the heuristic action, the data at LA might have been read or written by many applications. Correcting the damage can involve reversing the effects of these applications as well. The tools available are:

- DSN1LOGP. The summary report of this utility identifies the table spaces modified by the LUWID=16 thread.
- The statistics trace class 4 contains an IFCID 207. entry. This entry identifies the recovery log RBA for the LUWID=16 thread.

Notify the IBM Software Support about the problem.

Chapter 23. Recovery from BSDS or log failure during restart

Use this chapter when you have reason to believe that the bootstrap data set (BSDS) or part of the recovery log for DB2 is damaged or lost and that damage is preventing restart. If the problem is discovered at restart, begin with one of these recovery procedures:

"Active log failure recovery" on page 531
"Archive log failure recovery" on page 535
"BSDS failure recovery" on page 538

If the problem persists, return to the procedures in this chapter.

When DB2 recovery log damage terminates restart processing, DB2 issues messages to the console identifying the damage and giving an abend reason code. (The SVC dump title includes a more specific abend reason code to assist in problem diagnosis.) If the explanations in Part 2 of *DB2 Messages* indicate that restart failed because of some problem not related to a log error, refer to Part 2 of *DB2 Diagnosis Guide and Reference* and contact the IBM support center.

To minimize log problems during restart, the system requires two copies of the BSDS. Dual logging is also recommended.

Basic approaches to recovery: There are two basic approaches to recovery from problems with the log:

- Restart DB2, bypassing the inaccessible portion of the log and rendering some data inconsistent. Then recover the inconsistent objects by using the RECOVER utility, or re-create the data using REPAIR. Use the methods that are described following this procedure to recover the inconsistent data.
- Restore the entire DB2 subsystem to a prior point of consistency. The method requires that you have first prepared such a point; for suggestions, see "Preparing to recover to a prior point of consistency" on page 481. Methods of recovery are described under "Unresolvable BSDS or log data set problem during restart" on page 610.

Bypassing the damaged log: Even if the log is damaged, and DB2 is started by circumventing the damaged portion, the log is the most important source for determining what work was lost and what data is inconsistent. For information about data sharing considerations, see Chapter 5 of *DB2 Data Sharing: Planning and Administration*.

Bypassing a damaged portion of the log generally proceeds with the following steps:

1. DB2 restart fails. A problem exists on the log, and a message identifies the location of the error. The following abend reason codes, which appear only in the dump title, can be issued for this type of problem. This is not an exhaustive list; other codes might occur.

00D10261	00D10264	00D10267	00D1032A	00D1032C
00D10262	00D10265	00D10268	00D1032B	00E80084
00D10263	00D10266	00D10329		

Figure 57 on page 592 illustrates the general problem:

© Copyright IBM Corp. 1982, 2008

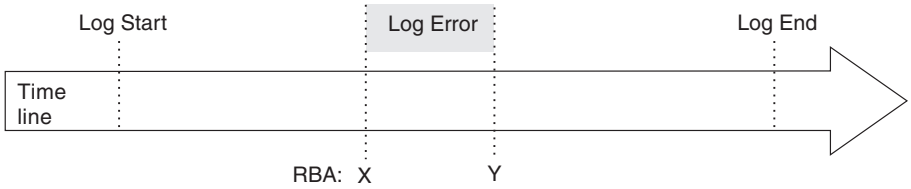


Figure 57. General problem of damaged DB2 log information

2. DB2 cannot skip over the damaged portion of the log and continue restart processing. Instead, you restrict processing to only a part of the log that is error free. For example, the damage shown in Figure 57 occurs in the log RBA range from X to Y. You can restrict restart to all of the log before X; then changes later than X are not made. Or you can restrict restart to all of the log after Y; then changes between X and Y are not made. In either case, some amount of data is inconsistent.
3. You identify the data that is made inconsistent by your restart decision. With the SUMMARY option, the DSN1LOGP utility scans the accessible portion of the log and identifies work that must be done at restart, namely, the units of recovery to be completed and the page sets that they modified. (For instructions on using DSN1LOGP, see Part 3 of *DB2 Utility Guide and Reference*.)

   Because a portion of the log is inaccessible, the summary information might not be complete. In some circumstances, your knowledge of work in progress is needed to identify potential inconsistencies.
4. You use the CHANGE LOG INVENTORY utility to identify the portion of the log to be used at restart, and to tell whether to bypass any phase of recovery. You can choose to do a cold start and bypass the entire log.
5. You restart DB2. Data that is unaffected by omitted portions of the log is available for immediate access.
6. Before you allow access to any data that is affected by the log damage, you resolve all data inconsistencies. That process is described under "Resolving inconsistencies resulting from a conditional restart" on page 616.

Where to start: The specific procedure depends on the phase of restart that was in control when the log problem was detected. On completion, each phase of restart writes a message to the console. You must find the last of those messages in the console log. The next phase after the one identified is the one that was in control when the log problem was detected. Accordingly, start at:
- "Log initialization or current status rebuild failure recovery" on page 593
- "Failure during forward log recovery" on page 602
- "Failure during backward log recovery" on page 607

As an alternative, determine which, if any, of the following messages was last received and follow the procedure for that message. Other DSN messages can be issued as well.

Message ID	Procedure to use
DSNJ001I	"Log initialization or current status rebuild failure recovery" on page 593
DSNJ100I	"Unresolvable BSDS or log data set problem during restart" on page 610
DSNJ107I	"Unresolvable BSDS or log data set problem during restart" on page 610
DSNJ1191	"Unresolvable BSDS or log data set problem during restart" on page 610
DSNR002I	None. Normal restart processing can be expected.
DSNR004I	"Failure during forward log recovery" on page 602

Message ID	Procedure to use
DSNR005I	“Failure during backward log recovery” on page 607
DSNR006I	None. Normal restart processing can be expected.
Other	“Log initialization or current status rebuild failure recovery”

Another scenario (“Failure resulting from total or excessive loss of log data” on page 613) provides information to use if you determine (by using Log initialization or current status rebuild failure recovery) that an excessive amount (or all) of DB2 log information (BSDS, active, and archive logs) has been lost.

The last scenario in this chapter (“Resolving inconsistencies resulting from a conditional restart” on page 616) can be used to resolve inconsistencies introduced while using one of the restart scenarios in this chapter. If you decide to use “Unresolvable BSDS or log data set problem during restart” on page 610, it is not necessary to use “Resolving inconsistencies resulting from a conditional restart” on page 616.

Because of the severity of the situations described, the scenarios identify “Operations management action”, rather than “Operator action”. Operations management might not be performing all the steps in the procedures, but they must be involved in making the decisions about the steps to be performed.

Log initialization or current status rebuild failure recovery

Problem: A failure occurred during the log initialization or current status rebuild phase of restart.

Symptom: An abend was issued indicating that restart failed. In addition, the last restart message received was a DSNJ001I message indicating a failure during current status rebuild, or none of the following messages was issued:

DSNJ001I
DSNR004I
DSNR005I

If none of the preceding messages were issued, the failure occurred during the log initialization phase of restart.

System action: The action depends on whether the failure occurred during log initialization or during current status rebuild.

- **Failure during log initialization:** DB2 terminates because a portion of the log is inaccessible, and DB2 cannot locate the end of the log during restart.
- **Failure during current status rebuild:** DB2 terminates because a portion of the log is inaccessible, and DB2 cannot determine the state of the subsystem (such as outstanding units of recovery, outstanding database writes, or exception database conditions) that existed at the prior DB2 termination.

Operations management action: To correct the problem, choose one of the following approaches:

- Correct the problem that has made the log inaccessible and start DB2 again. To determine if this approach is possible, refer to *DB2 Codes* for an explanation of the messages and codes received. The explanation will identify the corrective action that can be taken to resolve the problem. In this case, it is not necessary to read the scenarios in this chapter.

- Restore the DB2 log and all data to a prior consistent point and start DB2. This procedure is described in "Unresolvable BSDS or log data set problem during restart" on page 610.
- Start DB2 without completing some database changes. Using a combination of DB2 services and your own knowledge, determine what work will be lost by truncating the log. The procedure for determining the page sets that contain incomplete changes is described in "Restart by truncating the log" on page 595. In order to obtain a better idea of what the problem is, read one of the following sections, depending on when the failure occurred.

Description of failure during log initialization

Figure 58 illustrates the problem on the log.

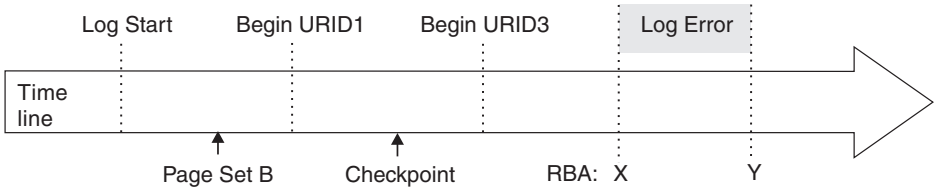


Figure 58. Failure during log initialization

The portion of the log between log RBAs X and Y is inaccessible. For failures that occur during the log initialization phase, the following activities occur:

1. DB2 allocates and opens each active log data set that is not in a stopped state.
2. DB2 reads the log until the last log record is located.
3. During this process, a problem with the log is encountered, preventing DB2 from locating the end of the log. DB2 terminates and issues one of the abend reason codes listed in Table 102 on page 596.

During its operations, DB2 periodically records in the BSDS the RBA of the last log record written. This value is displayed in the print log map report as follows:

```
HIGHEST RBA WRITTEN:    00000742989E
```

Because this field is updated frequently in the BSDS, the *highest RBA written* can be interpreted as an approximation of the end of the log. The field is updated in the BSDS when any one of a variety of internal events occurs. In the absence of these internal events, the field is updated each time a complete cycle of log buffers is written. A complete cycle of log buffers occurs when the number of log buffers written equals the value of the OUTPUT BUFFER field of installation panel DSNTIPL. The value in the BSDS is, therefore, relatively close to the end of the log.

To find the actual end of the log at restart, DB2 reads the log forward sequentially, starting at the log RBA that approximates the end of the log and continuing until the actual end of the log is located.

Because the end of the log is inaccessible in this case, some information has been lost. Units of recovery might have successfully committed or modified additional page sets past point X. Additional data might have been written, including those that are identified with writes pending in the accessible portion of the log. New units of recovery might have been created, and these might have modified data. Because of the log error, DB2 cannot perceive these events.

How to restart DB2 is described under "Restart by truncating the log" on page 595.

Description of failure during current status rebuild

Figure 59 illustrates the problem on the log.

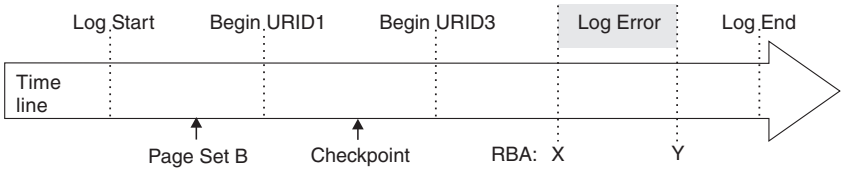


Figure 59. Failure during current status rebuild

The portion of the log between log RBAs X and Y is inaccessible. For failures that occur during the current status rebuild phase, the following activities occur:

1. Log initialization completes successfully.
2. DB2 locates the last checkpoint. (The BSDS contains a record of its location on the log.)
3. DB2 reads the log, beginning at the checkpoint and continuing to the end of the log.
4. DB2 reconstructs the subsystem's state as it existed at the prior termination of DB2.
5. During this process, a problem with the log is encountered, preventing DB2 from reading all required log information. DB2 terminates with one of the abend reason codes listed in Table 102 on page 596.

Because the end of the log is inaccessible in this case, some information has been lost. Units of recovery might have successfully committed or modified additional page sets past point X. Additional data might have been written, including those that are identified with writes pending in the accessible portion of the log. New units of recovery might have been created, and these might have modified data. Because of the log error, DB2 cannot perceive these events.

How to restart DB2 is described under "Restart by truncating the log."

Restart by truncating the log

When a portion of the log is inaccessible, during the log initialization or current status rebuild phases of restart, DB2 cannot identify precisely what units of recovery failed to complete, what page sets those modified, and what page sets have writes pending. This procedure tells how to gather that information and restart.

Step 1: Find the log RBA after the inaccessible part of the log

The log damage is illustrated in Figure 58 on page 594 and in Figure 59. The range of the log between RBAs X and Y is inaccessible to all DB2 processes.

Use the abend reason code accompanying the X'04E' abend and the message on the title of the accompanying dump at the operator's console, to find the name and location of a procedure in Table 102 on page 596. Use that procedure to find X and Y.

Table 102. Abend reason codes and messages

Abend reason code	Messag	Procedure	General error description
00D10261 00D10262 00D10263 00D10264 00D10265 00D10266 00D10267 00D10268	DSNJ012I	"RBA 1"	Log record is logically damaged
00D10329	DSNJ106I	"RBA 2"	I/O error occurred while log record was being read
00D1032A	DSNJ113E	"RBA 3" on page 597	Log RBA could not be found in BSDS
00D1032B	DSNJ103I	"RBA 4" on page 597	Allocation error occurred for an archive log data set
00D1032B	DSNJ007I	"RBA 5" on page 598	The operator canceled a request for archive mount
00D1032C	DSNJ104I	"RBA 4" on page 597	Open error occurred for an archive and active log data set
00E80084	DSNJ103I	"RBA 4" on page 597	Active log data set named in the BSDS could not be allocated during log initialization

Procedure RBA 1: The message accompanying the abend identifies the log RBA of the first inaccessible log record that DB2 detects. For example, the following message indicates a logical error in the log record at log RBA X'7429ABA'.

```
DSNJ012I ERROR D10265 READING RBA 000007429ABA
         IN DATA SET DSNCAT.LOGCOPY2.DS01
         CONNECTION-ID=DSN,
         CORRELATION-ID=DSN
```

Figure 151 on page 1117 shows that a given physical log record is actually a set of logical log records (the log records generally spoken of) and the log control interval definition (LCID). DB2 stores logical records in blocks of physical records to improve efficiency. When this type of an error on the log occurs during log initialization or current status rebuild, all log records within the physical log record are inaccessible. Therefore, the value of X is the log RBA that was reported in the message rounded down to a 4 KB boundary (X'7429000').

Continue with "Step 2: Identify lost work and inconsistent data" on page 598.

Procedure RBA 2: The message accompanying the abend identifies the log RBA of the first inaccessible log record that DB2 detects. For example, the following message indicates an I/O error in the log at RBA X'7429ABA'.

```
DSNJ106I LOG READ ERROR DSNAME=DSNCAT.LOGCOPY2.DS01,
         LOGRBA=000007429ABA,ERROR STATUS=0108320C
```

Figure 151 on page 1117 shows that a given physical log record is actually a set of logical log records (the log records generally spoken of) and the LCID. When this type of an error on the log occurs during log initialization or current status rebuild, all log records within the physical log record and beyond it to the end of the log data set are inaccessible to the log initialization or current status rebuild phase of

restart. Therefore, the value of X is the log RBA that was reported in the message, rounded down to a 4 KB boundary (X'7429000').

Continue with “Step 2: Identify lost work and inconsistent data” on page 598.

Procedure RBA 3: The message accompanying the abend identifies the log RBA of the inaccessible log record. This log RBA is not registered in the BSDS.

For example, the following message indicates that the log RBA X'7429ABA' is not registered in the BSDS:

```
DSNJ113E RBA 000007429ABA NOT IN ANY ACTIVE OR ARCHIVE
         LOG DATA SET.  CONNECTION-ID=DSN, CORRELATION-ID=DSN
```

The print log map utility can be used to list the contents of the BSDS. For an example of the output, see the description of print log map (DSNJU004) in Part 3 of *DB2 Utility Guide and Reference*.

Figure 151 on page 1117 shows that a given physical log record is actually a set of logical log records (the log records generally spoken of) and the LCID. When this type of an error on the log occurs during log initialization or current status rebuild, all log records within the physical log record are inaccessible.

Using the print log map output, locate the RBA closest to, but less than, X'7429ABA' for the value of X. If there is not an RBA that is less than X'7429ABA', a considerable amount of log information has been lost. If this is the case, continue with “Failure resulting from total or excessive loss of log data” on page 613.

If X has a value, continue with “Step 2: Identify lost work and inconsistent data” on page 598.

Procedure RBA 4: The message accompanying the abend identifies an entire data set that is inaccessible. For example, the following message indicates that the archive log data set DSNCAT.ARCHLOG1.A0000009 is not accessible, and the STATUS field identifies the code that is associated with the reason for the data set being inaccessible. For an explanation of the STATUS codes, see the explanation for the message in Part 2 of *DB2 Messages*.

```
DSNJ103I - csect-name LOG ALLOCATION ERROR
         DSNAME=DSNCAT.ARCHLOG1.A0000009,ERROR
         STATUS=04980004
         SMS REASON CODE=00000000
```

To determine the value of X, run the print log map utility to list the log inventory information. For an example of the output, see the description of print log map (DSNJU004) in Part 3 of *DB2 Utility Guide and Reference*. The output provides each log data set name and its associated log RBA range—the values of X and Y.

Verify the accuracy of the information in the print log map utility output for the active log data set with the lowest RBA range. For this active log data set only, the information in the BSDS is potentially inaccurate for the following reasons:

- When an active log data set is full, archiving is started. DB2 then selects another active log data set, usually the data set with the lowest RBA. This selection is made so that units of recovery do not have to wait for the archive operation to complete before logging can continue. However, if a data set has not been archived, nothing beyond it has been archived, and the procedure is ended.
- When logging has begun on a reusable data set, DB2 updates the BSDS with the new log RBA range for the active log data set, and marks it as *Not Reusable*. The

process of writing the new information to the BSDS can be delayed by other processing. It is therefore possible for a failure to occur between the time that logging to a new active log data set begins and the time that the BSDS is updated. In this case, the BSDS information is not correct.

The log RBA that appears for the active log data set with the lowest RBA range in the print log map utility output is valid, provided that the data set is marked *Not Reusable*. If the data set is marked *Reusable*, it can be assumed for the purposes of this restart that the starting log RBA (X) for this data set is one greater than the highest log RBA listed in the BSDS for all other active log data sets.

Continue with "Step 2: Identify lost work and inconsistent data."

Procedure RBA 5: The message accompanying the abend identifies an entire data set that is inaccessible. For example, the following message indicates that the archive log data set DSNCAT.ARCHLOG1.A0000009 is not accessible. The operator canceled a request for archive mount, resulting in the following message:

```
DSNJ007I OPERATOR CANCELED MOUNT OF ARCHIVE
         DSNCAT.ARCHLOG1.A0000009 VOLSER=5B225.
```

To determine the value of X, run the print log map utility to list the log inventory information. For an example of the output, see the description of print log map (DSNJU004) in Part 3 of *DB2 Utility Guide and Reference*. The output provides each log data set name and its associated log RBA range: the values of X and Y.

Continue with "Step 2: Identify lost work and inconsistent data."

Step 2: Identify lost work and inconsistent data

1. Obtain available information to help you determine the extent of the loss.

   It is impossible for DB2 to determine what units of recovery are not completed, what database state information is lost, or what data is inconsistent in this situation. The log contains all such information, but the information is not available. The following steps explain what to do to obtain the information that is available within DB2 to help determine the extent of the loss. The steps also explain how to start DB2 in this situation.

   After restart, data is inconsistent. Results of queries and any other operations on such data vary from incorrect results to abends. Abends that occur either identify an inconsistency in the data or incorrectly assume the existence of a problem in the DB2 internal algorithms. **If the inconsistent page sets cannot be identified and the problems in them cannot be resolved after starting DB2, there is a risk in following this procedure and allowing access to inconsistent data**.

   a. Execute the print log map utility. The report it produces includes a description of the last 100 checkpoints and provides, for each checkpoint:
      The location in the log of the checkpoint (begin and end RBA)
      The date and time of day that the checkpoint was performed.

   b. Locate the checkpoint on the log prior to the point of failure (X). Do that by finding the first checkpoint with an end RBA that is less than X.

      If you cannot find such a checkpoint, this means that a considerable amount of log has been lost. In this case, either follow the procedure under "Failure resulting from total or excessive loss of log data" on page 613 or the procedure under "Unresolvable BSDS or log data set problem during restart" on page 610.

If the checkpoint is found, look at the date and time that it was created. If the checkpoint is several days old (and DB2 was operational during the interim), either follow the procedure under "Failure resulting from total or excessive loss of log data" on page 613 or the procedure under "Unresolvable BSDS or log data set problem during restart" on page 610.

Otherwise, continue with the next step.

2. Determine what work is lost and what data is inconsistent.

   The portion of the log representing activity that occurred before the failure provides information about work that was in progress at that point. From this information, it might be possible to deduce the work that was in progress within the inaccessible portion of the log. If use of DB2 was limited at the time or if DB2 was dedicated to a small number of activities (such as batch jobs performing database loads or image copies), it might be possible to accurately identify the page sets that were made inconsistent. To make the identification, extract a summary of the log activity up to the point of damage in the log by using the DSN1LOGP utility described in Part 3 of *DB2 Utility Guide and Reference*.

   Use the DSN1LOGP utility to specify the "BEGIN CHECKPOINT" RBA prior to the point of failure, which was determined in the previous step as the RBASTART. End the DSN1LOGP scan prior to the point of failure on the log (X - 1) by using the RBAEND specification.

   Specifying the last complete checkpoint is very important for ensuring that complete information is obtained from DSN1LOGP.

   Specify the SUMMARY(ONLY) option to produce a summary report.

   Figure 60 is an example of a DSN1LOGP job to obtain summary information for the checkpoint discussed previously.

```
//ONE EXEC PGM=DSN1LOGP
//STEPLIB  DD DSN=prefix.SDSNLOAD,DISP=SHR
//SYSABEND DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//SYSSUMRY DD SYSOUT=A
//BSDS     DD DSN=DSNCAT.BSDS01,DISP=SHR
//SYSIN    DD *
   RBASTART (7425468) RBAEND (7428FFF) SUMMARY (ONLY)
/*
```

Figure 60. Sample JCL for obtaining DSN1LOGP summary output for restart

3. Analyze the DSN1LOGP utility output.

   The summary report that is placed in the SYSSUMRY file includes two sections of information: a summary of completed events (not shown here) and a restart summary shown in Figure 61 on page 600. A description of the sample output appears after the figure.

```
DSN1157I RESTART SUMMARY

DSN1153I DSN1LSIT CHECKPOINT
               STARTRBA=000007425468  ENDRBA=000007426C6C  STARTLRSN=AA527AA809DF  ENDLRSN=AA527AA829F4
               DATE=92.284  TIME=14:49:25

DSN1162I DSN1LPRT  UR  CONNID=BATCH     CORRID=PROGRAM2      AUTHID=ADMF001   PLAN=TCEU02
             START DATE=92.284 TIME=11:12:01  DISP=INFLIGHT    INFO=COMPLETE
             STARTRBA=0000063DA17B  STARTLRSN=A974FAFF27FF  NID=*
             LUWID=DB2NET.LUND0.A974FAFE6E77.0001  COORDINATOR=*
             PARTICIPANTS=*

             DATA MODIFIED:
                  DATABASE=0101=STVDB02   PAGESET=0002=STVTS02

DSN1162I DSN1LPRT  UR  CONNID=BATCH     CORRID=PROGRAM5      AUTHID=ADMF001   PLAN=TCEU02
             START DATE=92.284 TIME=11:21:02  DISP=INFLIGHT    INFO=COMPLETE
             STARTRBA=000006A57C57  STARTLRSN=A974FAFF2801  NID=*
             LUWID=DB2NET.LUND0.A974FAFE6FFF.0003  COORDINATOR=*
             PARTICIPANTS=*

             DATA MODIFIED:
                  DATABASE=0104=STVDB05   PAGESET=0002=STVTS05

DSN1162I DSN1LPRT  UR  CONNID=TEST0001  CORRID=CTHDCORID001  AUTHID=MULT002   PLAN=DONSQL1
             START DATE=92.278 TIME=06:49:33  DISP=INDOUBT     INFO=PARTIAL
             STARTRBA=000005FBCC4F  STARTLRSN=A974FBAF2302  NID=*
             LUWID=DB2NET.LUND0.B978FAFEFAB1.0000  COORDINATOR=*
             PARTICIPANTS=*

             NO DATA MODIFIED (BASED ON INCOMPLETE LOG INFORMATION)

DSN1162I UR  CONNID=BATCH     CORRID=PROGRAM2      AUTHID=ADMF001   PLAN=TCEU02
             START DATE=92.284 TIME=11:12:01  DISP=INFLIGHT    INFO=COMPLETE
             START=0000063DA17B

DSN1160I DATABASE WRITES PENDING:
          DATABASE=0001=DSNDB01    PAGESET=004F=SYSUTIL    START=000007425468
          DATABASE=0102   PAGESET=0015   START=000007425468
```

Figure 61. Partial sample of DSN1LOGP summary output

The following heading message is followed by messages that identify the units of recovery that have not yet completed and the page sets that they modified:

```
DSN1157I RESTART SUMMARY
```

Following the summary of outstanding units of recovery is a summary of page sets with database writes pending.

In each case (units of recovery or databases with pending writes), the earliest required log record is identified by the START information. In this context, START information is the log RBA of the earliest log record required in order to complete outstanding writes for this page set.

Those units of recovery with a START log RBA equal to, or prior to, the point Y cannot be completed at restart. All page sets modified by such units of recovery are inconsistent after completion of restart using this procedure.

All page sets identified in message DSN1160I with a START log RBA value equal to, or prior to, the point Y have database changes that cannot be written to disk. As in the case previously described, all such page sets are inconsistent after completion of restart using this procedure.

At this point, it is only necessary to identify the page sets in preparation for restart. After restart, the problems in the page sets that are inconsistent must be resolved.

Because the end of the log is inaccessible, some information has been lost, therefore, the information is inaccurate. Some of the units of recovery that appear to be inflight might have successfully committed, or they could have modified additional page sets beyond point X. Additional data could have been

written, including those page sets that are identified as having writes pending in the accessible portion of the log. New units of recovery could have been created, and these can have modified data. DB2 cannot detect that these events occurred.

From this and other information (such as system accounting information and console messages), it could be possible to determine what work was actually outstanding and which page sets will be inconsistent after starting DB2, because the record of each event contains the date and time to help determine how recent the information is. In addition, the information is displayed in chronological sequence.

Step 3: Determine what status information has been lost

Some amount of system status information might have been lost. In some cases, you will know what information has been lost (such as the case in which utilities are in progress). In other cases, messages about the loss of status information (such as in the cases of deferred restart pending or write error ranges) might be received. If system status information has been lost, you might be able to reconstruct this information from recent console displays, messages, and abends that alerted you to these conditions. These page sets contain inconsistencies that you must resolve.

Step 4: Truncate the log at the point of error

No DB2 process, including RECOVER, allows a gap in the log RBA sequence. You cannot process up to point X, skip over points X through Y, and continue after Y.

Use the change log inventory utility to create a conditional restart control record (CRCR) in the BSDS, identifying the end of the log (X) to use on a subsequent restart. The value is the RBA at which DB2 begins writing new log records. If point X is X'7429000', on the CRESTART control statement, specify ENDRBA=7429000.

At restart, DB2 discards the portion of the log beyond X'7429000' before processing the log for completing work (such as units of recovery and database writes). Unless otherwise directed, normal restart processing is performed within the scope of the log. Because log information has been lost, DB2 errors can occur. For example, a unit of recovery that has actually been committed can be rolled back. Also, some changes made by that unit of recovery might not be rolled back because information about data changes has been lost.

To minimize such errors, use this change log inventory control statement:

```
CRESTART  CREATE,ENDRBA=7429000,FORWARD=NO,BACKOUT=NO
```

Recovering and backing out units of recovery with lost information can introduce more inconsistencies than the incomplete units of recovery.

When you start DB2 after you run DSNJU003 with the FORWARD=NO and the BACKOUT=NO change log inventory options, DB2 performs the following actions (in “Step 5: Start DB2” on page 602):

1. Discards from the checkpoint queue any entries with RBAs beyond the ENDRBA value in the CRCR (which is X'7429000' in the previous example).
2. Reconstructs the system status up to the point of log truncation.
3. Performs pending database writes that the truncated log specifies that have not already applied to the data. You can use the DSN1LOGP utility to identify these writes. No forward recovery processing occurs for units of work in a FORWARD=NO conditional restart. All pending writes for in-commit and

indoubt units of recovery are applied to the data. The processing for the different unit of work states that is described in "Phase 3: Forward log recovery" on page 458 does not occur.

4. Marks all units of recovery that have committed or are indoubt as complete on the log.
5. Leaves inflight and in-abort units of recovery incomplete. Inconsistent data is left in tables modified by inflight or indoubt URs. When you specify a BACKOUT=NO conditional restart, inflight and in-abort units of recovery are not backed out.

   In a conditional restart that truncates the log, BACKOUT=NO minimizes DB2 errors for the following reasons:

   - Inflight units of recovery might have been committed in the portion of the log that the conditional restart discarded. If these units of recovery are backed out as "Phase 4: Backward log recovery" on page 459 describes, DB2 can back out database changes incompletely, which introduces additional errors.
   - Data modified by in-abort units of recovery could have been modified again after the point of damage on the log. For in-abort units of recovery DB2 could have written backout processing to disk after the point of log truncation. If these units of recovery are backed out as "Phase 4: Backward log recovery" on page 459 describes, DB2 can introduce additional data inconsistencies by backing out units of recovery that are already partially or fully backed out.

Step 5: Start DB2

At the end of restart, the conditional restart control record (CRCR) is marked *DEACTIVATED* to prevent its use on a later restart. Until the restart has completed successfully, the CRCR is in effect. Start DB2 with ACCESS (MAINT) until data is consistent or page sets are stopped.

Step 6: Resolve data inconsistency problems

After successfully restarting DB2, resolve all data inconsistency problems as described in "Resolving inconsistencies resulting from a conditional restart" on page 616.

Failure during forward log recovery

Problem: A failure occurred during the forward log recovery phase of restart.

Symptom: An abend was issued, indicating that restart had failed. In addition, the last restart message received was a DSNR004I message indicating that log initialization had completed and thus the failure occurred during forward log recovery.

System action: DB2 terminates because a portion of the log is inaccessible, and DB2 is therefore unable to guarantee the consistency of the data after restart.

Operations management action: To start DB2 successfully, choose one of the following approaches:

- Correct the problem that has made the log inaccessible and start DB2 again. To determine if this approach is possible, refer to *DB2 Codes* for an explanation of the messages and codes received. The explanation will identify any corrective action that can be taken to resolve the problem. In this case, it is not necessary to read the scenarios in this chapter.

- Restore the DB2 log and all data to a prior consistent point and start DB2. This procedure is described in "Unresolvable BSDS or log data set problem during restart" on page 610.
- Start DB2 without completing some database changes. The exact changes cannot be identified; all that can be determined is which page sets might have incomplete changes. The procedure for determining which page sets contain incomplete changes is described in "Starting DB2 by limiting restart processing." Continue reading this chapter to obtain a better idea of what the problem is.

Understanding forward log recovery failure

Figure 62 illustrates the problem on the log.

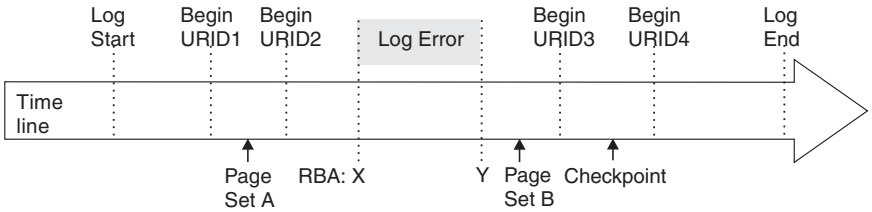


Figure 62. Illustration of failure during forward log recovery

The portion of the log between log RBA X and Y is inaccessible. The log initialization and current status rebuild phases of restart completed successfully. Restart processing was reading the log in a forward direction beginning at some point prior to X and continuing to the end of the log. Because of the inaccessibility of log data (between points X and Y), restart processing cannot guarantee the completion of any work that was outstanding at restart prior to point Y.

For purposes of discussion, assume the following work was outstanding at restart:

- The unit of recovery identified as URID1 was in-commit.
- The unit of recovery identified as URID2 was inflight.
- The unit of recovery identified as URID3 was in-commit.
- The unit of recovery identified as URID4 was inflight.
- Page set A had writes pending prior to the error on the log, continuing to the end of the log.
- Page set B had writes pending after the error on the log, continuing to the end of the log.

The earliest log record for each unit of recovery is identified on the log line in Figure 62. In order for DB2 to complete each unit of recovery, DB2 requires access to all log records from the beginning point for each unit of recovery to the end of the log.

The error on the log prevents DB2 from guaranteeing the completion of any outstanding work that began prior to point Y on the log. Consequently, database changes made by URID1 and URID2 might not be fully committed or backed out. Writes pending for page set A (from points in the log prior to Y) will be lost.

Starting DB2 by limiting restart processing

This procedure describes how to start DB2 when a portion of the log is inaccessible during forward recovery. It also describes how to identify the units of recovery for which database changes cannot be fully guaranteed (either committed or backed

out) and the page sets that these units of recovery changed. You must determine which page sets are involved because after this procedure is used, the page sets will contain inconsistencies that must be resolved. In addition, using this procedure results in the completion of all database writes that are pending. For a description of this process of writing database pages to disk, see “Tuning database buffer pools” on page 665.

Step 1: Find the log RBA after the inaccessible part of the log

The log damage is shown in Figure 62 on page 603. The range of the log between RBA X and RBA Y is inaccessible to all DB2 processes.

Use the abend reason code accompanying the X'04E' abend, and the message on the title of the accompanying dump at the operator’s console, to find the name and the location of a procedure in Table 103. Use that procedure to find X and Y.

Table 103. Abend reason codes and messages

Abend reason code	Message	Procedure	General error description
00D10261 00D10262 00D10263 00D10264 00D10265 00D10266 00D10267 00D10268	DSNJ012I	“RBA 1”	Log record is logically damaged
00D10329	DSNJ106I	“RBA 2” on page 605	I/O error occurred while log record was being read
00D1032A	DSNJ113E	“RBA 3” on page 605	Log RBA could not be found in BSDS
00D1032B	DSNJ103I	“RBA 4” on page 606	Allocation error occurred for an archive log data set
00D1032B	DSN007I	“RBA 5” on page 606	The operator canceled a request for archive mount
00D1032C	DSNJ104E	“RBA 4” on page 606	Open error occurred for an archive log data set
00E80084	DSNJ103I	“RBA 4” on page 606	Active log data set named in the BSDS could not be allocated during log initialization.

Procedure RBA 1: The message accompanying the abend identifies the log RBA of the first inaccessible log record that DB2 detects. For example, the following message indicates a logical error in the log record at log RBA X'7429ABA':

```
DSNJ012I ERROR D10265 READING RBA 000007429ABA
         IN DATA SET DSNCAT.LOGCOPY2.DS01
         CONNECTION-ID=DSN
         CORRELATION-ID=DSN
```

Figure 151 on page 1117 shows that a given physical log record is actually a set of logical log records (the log records generally spoken of) and the log control interval definition (LCID). When this type of an error on the log occurs during forward log recovery, all log records within the physical log record, as described, are inaccessible. Therefore, the value of X is the log RBA that was reported in the message, rounded down to a 4K boundary (that is, X'7429000').

For purposes of following the steps in this procedure, assume that the extent of damage is limited to the single physical log record. Therefore, calculate the value of Y as the log RBA that was reported in the message, rounded up to the end of the 4K boundary (that is, X'7429FFF').

Continue with “Step 2: Identify incomplete units of recovery and inconsistent page sets” on page 606.

Procedure RBA 2: The message accompanying the abend identifies the log RBA of the first inaccessible log record that DB2 detects. For example, the following message indicates an I/O error in the log at RBA X'7429ABA':

```
DSNJ106I LOG READ ERROR DSNAME=DSNCAT.LOGCOPY2.DS01,
         LOGRBA=000007429ABA, ERROR STATUS=0108320C
```

Figure 151 on page 1117 shows that a given physical log record is actually a set of logical log records (the log records generally spoken of) and the LCID. When this type of an error on the log occurs during forward log recovery, all log records within the physical log record and beyond it to the end of the log data set are inaccessible to the forward recovery phase of restart. Therefore, the value of X is the log RBA that was reported in the message, rounded down to a 4K boundary (that is, X'7429000').

To determine the value of Y, run the print log map utility to list the log inventory information. For an example of this output, see the description of print log map (DSNJU004) in Part 3 of *DB2 Utility Guide and Reference*. Locate the data set name and its associated log RBA range. The RBA of the end of the range is the value Y.

Continue with “Step 2: Identify incomplete units of recovery and inconsistent page sets” on page 606.

Procedure RBA 3: The message accompanying the abend identifies the log RBA of the inaccessible log record. This log RBA is not registered in the BSDS.

For example, the following message indicates that the log RBA X'7429ABA' is not registered in the BSDS:

```
DSNJ113E RBA 000007429ABA NOT IN ANY ACTIVE OR ARCHIVE
         LOG DATA SET.  CONNECTION-ID=DSN, CORRELATION-ID=DSN
```

Use the print log map utility to list the contents of the BSDS. For an example of this output, see the description of print log map (DSNJU004) in Part 3 of *DB2 Utility Guide and Reference*.

Figure 151 on page 1117 shows that a given physical log record is actually a set of logical log records (the log records generally spoken of) and the LCID. When this type of error on the log occurs during forward log recovery, all log records within the physical log record are inaccessible.

Using the print log map output, locate the RBA closest to, but less than, X'7429ABA'. This is the value of X. If an RBA less than X'7429ABA' cannot be found, the value of X is zero. Locate the RBA closest to, but greater than, X'7429ABA'. This is the value of Y.

Continue with “Step 2: Identify incomplete units of recovery and inconsistent page sets” on page 606.

Procedure RBA 4: The message accompanying the abend identifies an entire data set that is inaccessible. For example, the following message indicates that the archive log data set DSNCAT.ARCHLOG1.A0000009 is not accessible. The STATUS field identifies the code that is associated with the reason for the data set being inaccessible. For an explanation of the STATUS codes, see the explanation for the message in *DB2 Messages*.

```
DSNJ103I LOG ALLOCATION ERROR
         DSNAME=DSNCAT.ARCHLOG1.A0000009, ERROR
         STATUS=04980004
         SMS REASON CODE=00000000
```

To determine the values of X and Y, run the print log map utility to list the log inventory information. For an example of this output, see the description of print log map (DSNJU004) in Part 2 of *DB2 Utility Guide and Reference*. The output provides each log data set name and its associated log RBA range: the values of X and Y.

Continue with "Step 2: Identify incomplete units of recovery and inconsistent page sets."

Procedure RBA 5: The message accompanying the abend identifies an entire data set that is inaccessible. For example, the following message indicates that the archive log data set DSNCAT.ARCHLOG1.A0000009 is not accessible. The operator canceled a request for archive mount resulting in the following message:

```
DSNJ007I OPERATOR CANCELED MOUNT OF ARCHIVE
         DSNCAT.ARCHLOG1.A0000009 VOLSER=5B225.
```

To determine the values of X and Y, run the print log map utility to list the log inventory information. For an example of the output, see the description of print log map (DSNJU004) in Part 3 of *DB2 Utility Guide and Reference*. The output provides each log data set name and its associated log RBA range: the values of X and Y. Continue with "Step 2: Identify incomplete units of recovery and inconsistent page sets."

Step 2: Identify incomplete units of recovery and inconsistent page sets

Units of recovery that cannot be fully processed are considered *incomplete units of recovery*. Page sets that will be inconsistent after completion of restart are considered *inconsistent page sets*. Take the following steps to identify them:

1. Determine the location of the latest checkpoint on the log. Determine this by looking at one of the following sources, whichever is more convenient:
   - The operator's console contains the following message, identifying the location of the start of the last checkpoint on the log at log RBA X'876B355'.

     ```
     DSNR003I RESTART ... PRIOR CHECKPOINT
                          RBA=00007425468
     ```

   - The print log map utility output identifies the last checkpoint, including its BEGIN CHECKPOINT RBA.
2. Run the DSN1LOGP utility to obtain a report of the outstanding work that is to be completed at the next restart of DB2. When you run the DSN1LOGP utility, specify the checkpoint RBA as the STARTRBA and the SUMMARY(ONLY) option. It is very important that you include the last complete checkpoint from running DSN1LOGP in order to obtain complete information.

   Figure 60 on page 599 shows an example of the DSN1LOGP job submitted for the checkpoint that was reported in the DSNR003I message.

Analyze the output of the DSN1LOGP utility. The summary report that is placed in the SYSSUMRY file contains two sections of information. For an example of SUMMARY output, see Figure 61 on page 600; and for an example of the program that results in the output, see Figure 60 on page 599.

Step 3: Restrict restart processing to the part of the log after the damage

Use the change log inventory utility to create a conditional restart control record (CRCR) in the BSDS. Identify the accessible portion of the log beyond the damage by using the STARTRBA specification, which will be used at the next restart. Specify the value Y+1 (that is, if Y is X'7429FFF', specify STARTRBA=742A000). Restart will restrict its processing to the portion of the log beginning with the specified STARTRBA and continuing to the end of the log. A sample change log inventory utility control statement is:

```
CRESTART CREATE,STARTRBA=742A000
```

Step 4: Start DB2

At the end of restart, the CRCR is marked DEACTIVATED to prevent its use on a subsequent restart. Until the restart is complete, the CRCR will be in effect. Use START DB2 ACCESS(MAINT) until data is consistent or page sets are stopped.

Step 5: Resolve inconsistent data problems

After the successful start of DB2, all data inconsistency problems must be resolved. "Resolving inconsistencies resulting from a conditional restart" on page 616 describes how to do this. At this time, all other data can be made available for use.

Failure during backward log recovery

Problem: A failure occurred during the backward log recovery phase of restart.

Symptom: An abend was issued that indicated that restart failed because of a log problem. In addition, the last restart message received was a DSNR005I message, indicating that forward log recovery completed and thus the failure occurred during backward log recovery.

System action: DB2 terminates because a portion of the log that it needs is inaccessible, and DB2 is therefore unable to rollback some database changes during restart.

Operations management action: To start DB2, choose one of the following approaches:

- Correct the problem that has made the log inaccessible and start DB2 again. To determine whether this approach is possible, refer to *DB2 Codes* for an explanation of the messages and codes received. The explanation identifies the corrective action to take to resolve the problem. In this case, it is not necessary to read the scenarios in this chapter.
- Restore the DB2 log and all data to a prior consistent point and start DB2. This procedure is described in "Unresolvable BSDS or log data set problem during restart" on page 610.
- Start DB2 without rolling back some database changes. The exact database changes cannot be identified. All that can be determined is which page sets contain incomplete changes and which units of recovery made modifications to those page sets. The procedure for determining which page sets contain incomplete changes and which units of recovery made the modifications is

described in “Bypassing backout before restarting.” Continue reading this chapter to obtain a better idea of how to fix the problem.

Understanding backward log recovery failure

Figure 63 illustrates the problem on the log.

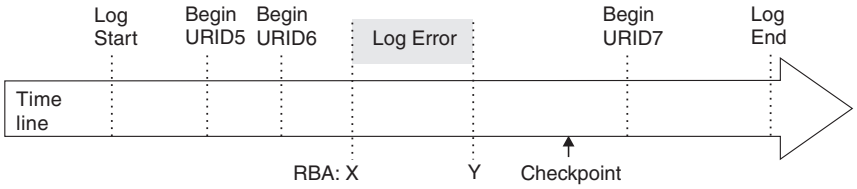


Figure 63. Illustration of failure during backward log recovery

The portion of the log between log RBA X and Y is inaccessible. Restart was reading the log in a backward direction beginning at the end of the log and continuing backward to the point marked by Begin URID5 in order to back out the changes made by URID5, URID6, and URID7. You can assume that DB2 determined that these units of recovery were inflight or in-abort. The portion of the log from point Y to the end has been processed. However, the portion of the log from Begin URID5 to point Y has not been processed and cannot be processed by restart. Consequently, database changes made by URID5 and URID6 might not be fully backed out. All database changes made by URID7 have been fully backed out, but these database changes might not have been written to disk. A subsequent restart of DB2 causes these changes to be written to disk during forward recovery.

Bypassing backout before restarting

This procedure describes how to start DB2 when a portion of the log is inaccessible during backward recovery. It also describes how to identify the units of recovery that cannot be fully backed out and the page sets that are inconsistent because they were changed by the units of recovery that did not complete.

1. Determine the units of recovery that cannot be backed out and the page sets that will be inconsistent after the completion of restart. To do this, take the following steps:
   a. Determine the location of the latest checkpoint on the log. This can be determined by looking at one of the following sources, whichever is more convenient:
      - The operator’s console contains message DSNR003I, which identifies the location of the start of the last checkpoint on the log at log RBA X'7425468'.

        ```
        DSNR003I RESTART ... PRIOR CHECKPOINT
                             RBA=00007425468
        ```

      - Print log map utility output identifies the last checkpoint, including its BEGIN CHECKPOINT RBA.
   b. Execute the DSN1LOGP utility to obtain a report of the outstanding work that is to be completed at the next restart of DB2. When you run DSN1LOGP, specify the checkpoint RBA as the RBASTART and the SUMMARY(ONLY) option. Include the last complete checkpoint in the execution of DSN1LOGP in order to obtain complete information.

      Figure 61 on page 600 shows an example of the DSN1LOGP job submitted for the checkpoint that was reported in the DSNR003I message.

Analyze the output of the DSN1LOGP utility. The summary report that is placed in the SYSSUMRY file contains two sections of information. The sample report output shown in Figure 61 on page 600 resulted from the invocation shown in Figure 60 on page 599. The following description refers to that sample output.

The first section is headed by the following message:

```
DSN1150I SUMMARY OF COMPLETED EVENTS
```

That message is followed by others that identify completed events, such as completed units of recovery. That section does not apply to this procedure.

The second section is headed by this message:

```
DSN1157I RESTART SUMMARY
```

That message is followed by others that identify units of recovery that are not yet completed and the page sets that they modified. An example of the DSN1162I messages is shown in Figure 61 on page 600.

After the summary of outstanding units of recovery is a summary of page sets with database writes pending. An example of the DSN1160I message is shown in Figure 61 on page 600.

The restart processing that failed was able to complete all units of recovery processing within the accessible scope of the log after point Y. Database writes for these units of recovery are completed during the forward recovery phase of restart on the next restart. Therefore, do not bypass the forward recovery phase. All units of recovery that can be backed out have been backed out.

All remaining units of recovery to be backed out (DISP=INFLIGHT or DISP=IN-ABORT) are bypassed on the next restart because their STARTRBA values are less than the RBA of point Y. Therefore, all page sets modified by those units of recovery are inconsistent after restart. This means that some changes to data might not be backed out. At this point, it is only necessary to identify the page sets in preparation for restart.

2. Direct restart to bypass backward recovery processing. Use the change log inventory utility to create a conditional restart control record (CRCR) in the BSDS. Direct restart to bypass backward recovery processing during the subsequent restart by using the BACKOUT specification. At restart, all units of recovery requiring backout are declared complete by DB2, and log records are generated to note the end of the unit of recovery. The change log inventory utility control statement is:

   ```
   CRESTART CREATE,BACKOUT=NO
   ```

3. Start DB2. At the end of restart, the CRCR is marked DEACTIVATED to prevent its use on a subsequent restart. Until the restart is complete, the CRCR is in effect. Use START DB2 ACCESS(MAINT) until data is consistent or page sets are stopped.
4. Resolve all inconsistent data problems. After the successful start of DB2, all data inconsistency problems must be resolved. “Resolving inconsistencies resulting from a conditional restart” on page 616 describes how to do this. At this time, all other data can be made available for use.

Failure during a log RBA read request

Problem: The BSDS is wrapping around too frequently when log RBA read requests are submitted; when the last archive log data sets were added to the BSDS, the maximum allowable number of log data sets in the BSDS was exceeded. This caused the earliest data sets in the BSDS to be displaced by the new entry. Subsequently, the requested RBA containing the dropped log data set cannot be read after the wrap occurs.

Symptom: Abend code 00D1032A and message DSNJ113E are displayed:

```
DSNJ113E RBA log-rba NOT IN ANY ACTIVE OR ARCHIVE
         LOG DATA SET.  CONNECTION-ID=aaaaaaaa, CORRELATION-ID=aaaaaaaa
```

System programmer action:

1. Stop DB2 with the STOP DB2 command, if it has not already been stopped automatically as a result of the problem.
2. Check any other messages and reason codes displayed and correct the errors indicated. Locate the output from an old print log map run, and identify the data set that contains the missing RBA. If the data set has not been reused, run the change log inventory utility to add this data set back into the inventory of log data sets.
3. Increase the maximum number of archive log volumes that can be recorded in the BSDS. To do this, update the MAXARCH system parameter value as follows:
   a. Start the installation CLIST.
   b. On panel DSNTIPA1, select UPDATE mode.
   c. On panel DSNTIPT, change any data set names that are not correct.
   d. On panel DSNTIPB, select the ARCHIVE LOG DATA SET PARAMETERS option.
   e. On panel DSNTIPA, increase the value of RECORDING MAX.
   f. When the installation CLIST editing completes, rerun job DSNTIJUZ to recompile the system parameters.
4. Start DB2 with the START DB2 command.

For more information about updating DB2 system parameters, see Part 2 of *DB2 Installation Guide*.

For instructions about adding an old archive data set, refer to "Changing the BSDS log inventory" on page 438. Also, see Part 3 of *DB2 Utility Guide and Reference* for additional information about the change log inventory utility.

Unresolvable BSDS or log data set problem during restart

Use dual logging (active logs, archive logs, and bootstrap data sets) to reduce your efforts in resolving the problem described in this section.

Problem: During restart of DB2, serious problems with the BSDS or log data sets were detected and cannot be resolved.

Symptom: The following messages are issued:
DSNJ100I
DSNJ107I
DSNJ119I

Any of the following problems could be involved:

- A log data set is physically damaged.
- Both copies of a log data set could be physically damaged in the case of dual logging mode.
- A log data set could be lost.
- An archive log volume could have been reused even though it was still needed.
- A log data set could contain records that are not recognized by DB2 because they are logically broken.

System action: DB2 cannot be restarted unless the following procedure is used:

Operations management action: In serious cases such as this, it can be necessary to fall back to a prior shutdown level. If this procedure is used, all database changes between the shutdown point and the present will be lost, but all the data retained will be consistent within DB2.

If it is necessary to fall back, read "Preparing to recover to a prior point of consistency" on page 481.

If too much log information has been lost, use the alternative approach described in "Failure resulting from total or excessive loss of log data" on page 613.

Preparing for recovery or restart

See "Preparing to recover to a prior point of consistency" on page 481 for preparation procedures.

Performing fall back to a prior shutdown point

1. When a failure occurs and you decide to fall back, use the print log map utility against the most current copy of the BSDS. Even if you are not able to do this, continue with the next step. (If you are unable to do this, an error message will be issued.)
2. Use access method services IMPORT to restore the backed-up versions of the BSDS and active log data sets.
3. Use the print log map utility against the copy of the BSDS with which DB2 is to be restarted.
4. Determine whether any archive log data sets must be deleted.
   - If you have a copy of the most current BSDS, compare it to the BSDS with which DB2 is to be restarted. Delete and uncatalog any archive log data sets that are listed in the most current BSDS but are not listed in the previous one. These archive log data sets are normal physical sequential (SAM) data sets. If you are able to do this step, continue with step 5.
   - If you were not able to print a copy of the most current BSDS and the archive logs are cataloged, use access method services LISTCAT to check for archive logs with a higher sequence number than the last archive log shown in the BSDS being used to restart DB2.
     - If no archive log data sets with a higher sequence number exist, you do not have to delete or uncatalog any data sets, and you can continue with step 5.
     - Delete and uncatalog all archive log data sets that have a higher sequence number than the last archive log data set in the BSDS being used to restart DB2. These archive log data sets are SAM data sets. Continue with the next step.

     If the archive logs are not cataloged, you do not need to uncatalog them.
5. Issue the command START DB2. Use START DB2 ACCESS(MAINT) until data is consistent or page sets are stopped. If DDL is required, the creator might not be the same.
6. Now, determine what data needs to be recovered, what data needs to be dropped, what data can remain unchanged, and what data needs to be recovered to the prior shutdown point.

- For table spaces and indexes that might have been changed after the shutdown point, use the DB2 RECOVER utility to recover these table spaces and indexes. They must be recovered in the order indicated in Part 2 of *DB2 Utility Guide and Reference*.
- For data that has not been changed after the shutdown point (data used with RO access), it is not necessary to use RECOVER or DROP.
- For table spaces that were deleted after the shutdown point, issue the DROP statement. These table spaces will not be recovered.
- Any objects created after the shutdown point should be re-created.

You must recover all data that has potentially been modified after the shutdown point. If the RECOVER utility is not used to recover modified data, serious problems can occur because of data inconsistency.

If an attempt is made to access data that is inconsistent, any of the following events can occur (and the list is not comprehensive):

- It is possible to successfully access the correct data.
- Data can be accessed without DB2 recognizing any problem, but it might not be the data you want (the index might be pointing to the wrong data).
- DB2 might recognize that a page is logically incorrect and, as a result, abend the subsystem with an X'04E' abend completion code and an abend reason code of X'00C90102'.
- DB2 might notice that a page was updated after the shutdown point and, as a result, abend the requester with an X'04E' abend completion code and an abend reason code of X'00C200C1'.

7. Analyze the CICS log and the IMS log to determine the work that must be redone (work that was lost because of shutdown at the previous point). Inform all TSO users, QMF users, and batch users for which no transaction log tracking has been performed, about the decision to fall back to a previous point.
8. When DB2 is started after being shut down, indoubt units of recovery can exist. This occurs if transactions are indoubt when the command STOP DB2 MODE (QUIESCE) is given. When DB2 is started again, these transactions will still be indoubt to DB2. IMS and CICS cannot know the disposition of these units of recovery.

   To resolve these indoubt units of recovery, use the command RECOVER INDOUBT.
9. If a table space was dropped and re-created after the shutdown point, it should be dropped and re-created again after DB2 is restarted. To do this, use SQL DROP and SQL CREATE statements.

   Do not use the RECOVER utility to accomplish this, because it will result in the old version (which can contain inconsistent data) being recovered.
10. If any table spaces and indexes were created after the shutdown point, these must be re-created after DB2 is restarted. There are two ways to accomplish this:
    - For data sets defined in DB2 storage groups, use the CREATE TABLESPACE statement and specify the appropriate storage group names. DB2 automatically deletes the old data set and redefines a new one.
    - For user-defined data sets, use access method services DELETE to delete the old data sets. After these data sets have been deleted, use access method services DEFINE to redefine them; then use the CREATE TABLESPACE statement.

Failure resulting from total or excessive loss of log data

Problem: Either all copies of the BSDS and logs have been destroyed or lost, or an excessive amount of the active log has been destroyed or lost.

Symptom: Any messages or abends indicating that all or an excessive amount of log information has been lost.

System action: None.

Operations management action: Restart DB2 without any log data by using either the procedure in "Total loss of the log" or "Excessive loss of data in the active log" on page 614.

Total loss of the log

Even if all copies of the BSDS and either the active or archive log or both have been destroyed or lost, DB2 can still be restarted, and data that belongs to that DB2 subsystem can still be accessed, provided that all system and user table spaces have remained intact and you have a recent copy of the BSDS. However, you must rely on your own sources to determine what data is inconsistent, because DB2 cannot provide any hints of inconsistencies. Other VSAM clusters on disk, such as the system databases DSNDB01, DSNDB04, and DSNB06, as well as user databases are assumed to exist.

For example, you might know that DB2 was dedicated to a few processes (such as utilities) during the DB2 session, and you might be able to identify the page sets they modified. If you cannot identify the page sets that are inconsistent, you must decide whether you are willing to assume the risk involved in restarting DB2 under those conditions. If you decide to restart, take the following steps:

1. Define and initialize the BSDSs. See "Recovering the BSDS from a backup copy" on page 539.
2. Define the active log data sets using the access method services DEFINE function. Run utility DSNJLOGF to initialize the new active log data sets.
3. Prepare to restart DB2 using no log data. See "Deferring restart processing" on page 450.

   Each data and index page contains the log RBA of the last log record applied against the page. Safeguards within DB2 disallow a modification to a page that contains a log RBA that is higher than the current end of the log. There are two choices.

   a. Run the DSN1COPY utility specifying the RESET option to reset the log RBA in every data and index page. Depending on the amount of data in the subsystem, this process can take quite a long time. Because the BSDS has been redefined and reinitialized, logging begins at log RBA 0 when DB2 starts.

      If the BSDS is not reinitialized, logging can be forced to begin at log RBA 0 by constructing a conditional restart control record (CRCR) that specifies a STARTRBA and ENDRBA that are both equal to 0, as the following command shows:

      ```
      CRESTART CREATE,STARTRBA=0,ENDRBA=0
      ```

      Continue with step 4.

   b. Determine the highest possible log RBA of the prior log. From previous console logs written when DB2 was operational, locate the last DSNJ001I message. When DB2 switches to a new active log data set, this message is

written to the console, identifying the data set name and the highest potential log RBA that can be written for that data set. Assume that this is the value X'8BFFF'. Add one to this value (X'8C000'), and create a conditional restart control record specifying the following change log inventory control statement:

```
CRESTART CREATE,STARTRBA=8C000,ENDRBA=8C000
```

When DB2 starts, all phases of restart are bypassed and logging begins at log RBA X'8C000'. If this method is chosen, it is not necessary to use the DSN1COPY RESET option and a lot of time is saved.

4. Start DB2. Use START DB2 ACCESS(MAINT) until data is consistent or page sets are stopped.
5. After restart, resolve all inconsistent data as described in "Resolving inconsistencies resulting from a conditional restart" on page 616.

Excessive loss of data in the active log

By studying "Total loss of the log" on page 613, a procedure can be developed for restarting that meets the requirements of the situation. Specifically, when an excessive amount of the active log has been lost, the procedure can be adapted to fit the situation, as described in "Total loss of the log" on page 613. *Do not delete and redefine the BSDS*. Instead, proceed as follows:

1. Use the print log map utility (DSNJU004) against the copy of the BSDS with which DB2 is to be restarted.
2. Use the print log map output to obtain the data set names of all active log data sets. Use access method services LISTCAT to determine which active log data sets are no longer available or usable.
3. Use access method services DELETE to delete all active log data sets that are no longer usable.
4. Use access method services DEFINE to define new active log data sets. Run utility DSNJLOGF to initialize the new active log data sets. One active log data set must be defined for each one found to be no longer available or usable in step 2. Use the active log data set name found in the BSDS as the data set name for the access method services DEFINE statement.
5. Use the print log map utility (DSNJU004) output, and note whether an archive log data set exists that contains the RBA range of the redefined active log data set. To do this, note the starting and ending RBA values for the active log data set that was newly redefined, and look for an archive log data set with the same starting and ending RBA values.

   If no such archive log data sets exist:

   a. Use the change log inventory utility (DSNJU003) DELETE statement to delete the newly redefined active log data sets from the BSDS active log data set inventory.
   b. Use the change log inventory utility (DSNJU003) NEWLOG statement to add the active log data set back into the BSDS active log data set inventory. Do not specify RBA ranges on the NEWLOG statement.

   If the corresponding archive log data sets exist, then there are two courses of action:

   - If you want to minimize the number of potential read operations against the archive log data sets, then use access method services REPRO to copy the data from each archive log data set into the corresponding active log data set. Make certain you copy the proper RBA range into the active log data set.

     Be sure that the active log data set is big enough to hold all the data from the archive log data set. When DB2 does an archive operation, it copies the

log data from the active log data set to the archive log data set, then pads the archive log data set with binary zeroes to fill a block. In order for the access method services REPRO command to be able to copy all of the data from the archive log data set to a newly defined active log data set, the new active log data set might need to be bigger than the original one.

For example, if the block size of the archive log data set is 28 KB, and the active log data set contains 80 KB of data, DB2 copies the 80 KB and pads the archive log data set with 4 KB of nulls to fill the last block. Thus, the archive log data set now contains 84 KB of data instead of 80 KB. In order for the access method services REPRO command to complete successfully, the active log data set must be able to hold 84 KB, rather than just 80 KB of data.

- If you are not concerned about read operations against the archive log data sets, complete the two steps that appear in the preceding list (as though the archive data sets did not exist).

6. Choose the appropriate point for DB2 to start logging (X'8C000') as described in "Total loss of the log" on page 613.
7. To restart DB2 without using any log data, create a CRCR, as described for the change log inventory utility (DSNJU003) in Part 3 of *DB2 Utility Guide and Reference*.
8. Start DB2. Use START DB2 ACCESS(MAINT) until data is consistent or page sets are stopped.
9. After restart, resolve all inconsistent data as described in "Resolving inconsistencies resulting from a conditional restart" on page 616.

This procedure will cause all phases of restart to be bypassed and logging to begin at log RBA X'8C000'. It will create a gap in the log between the highest RBA kept in the BSDS and X'8C000', and that portion of the log will be inaccessible.

No DB2 process can tolerate a gap, including RECOVER. Therefore, all data must be image copied after a cold start. Even data that is known to be consistent must be image copied again when a gap is created in the log.

There is another approach to doing a cold start that does not create a gap in the log. This is only a method for eliminating the gap in the physical record. It does not mean that you can use a cold start to resolve the logical inconsistencies. The procedure is as follows:

1. Locate the last valid log record by using DSN1LOGP to scan the log. (Message DSN1213I identifies the last valid log RBA.)
2. Begin at an RBA that is known to be valid. If message DSN1213I indicated that the last valid log RBA is at X'89158', round this value up to the next 4K boundary (X'8A000').
3. Create a CRCR similar to the CRCR that the following command specifies:

   ```
   CRESTART CREATE,STARTRBA=8A000,ENDRBA=8A000
   ```

4. Use START DB2 ACCESS(MAINT) until data is consistent or page sets are stopped.
5. Now, take image copies of all data for which data modifications were recorded beyond log RBA X'8A000'. If you do not know what data was modified, take image copies of all data.

   If image copies are not taken of data that has been modified beyond the log RBA used in the CRESTART statement, future RECOVER operations can fail or result in inconsistent data.

After restart, resolve all inconsistent data as described in Resolving inconsistencies resulting from a conditional restart.

Resolving inconsistencies resulting from a conditional restart

When a conditional restart of the DB2 subsystem is done, the following problems can occur:

- Some amount of work is left incomplete.
- Some data is left inconsistent.
- Information about the status of the DB2 subsystem is made unusable.

Inconsistencies in a distributed environment

In a distributed environment, when a DB2 system restarts, it indicates its restart status and the name of its recovery log to the systems with which it communicates. There are two possible conditions for restart status, *warm* and *cold*.

A cold status for restart means that the DB2 system has no memory of previous connections with its partner, and therefore has no memory of indoubt logical units of work. This includes all postponed aborted URs that end without resolution and any restart pending page sets and partitions are removed from restart pending state.

The partner accepts the cold start connection and remembers the recovery log name of the cold starting DB2. If the partner has indoubt thread resolution requirements with the cold starting DB2, those requirements cannot be achieved. The partner terminates its indoubt resolution responsibility with the cold starting DB2. However, as a participant, the partner has indoubt logical units of work that must be resolved manually.

The DB2 system has an incomplete record of resolution responsibilities. It attempts to reconstruct as much resynchronization information as possible and displays the information in one or more DSNL438 or DSNL439 messages. The displayed information is then forgotten.

A warm status for restart means the DB2 system does have memory of previous connections with the partner and therefore does have memory of indoubt logical units of work. The exchange of recovery log names validates that the correct recovery logs are being used for indoubt resolution. Each partner indicates its recovery log name and the recovery log name it believes to be the one the other partner is using. A warm start connection where one system specifies a recovery log name that is different than the name remembered by the other system is rejected if indoubt resolution is required between the two partners.

Procedures for resolving inconsistencies

The following section explains what must be done to resolve any inconsistencies that exist. Before reading this section, the procedures in the other sectionss in this chapter must be considered. Each one provides important steps that must be followed before using the information in this section.

The following three methods describe one or more steps that must be taken to resolve inconsistencies in the DB2 subsystem. Before using these methods, however, complete the following procedure:

1. Obtain image copies of all DB2 table spaces. You will need these image copies if any of the following conditions apply:
   - You did a cold start.

- You did a conditional restart that altered or truncated the log.
- The log is damaged.
- Part of the log is no longer accessible.

The first thing to do after a conditional restart is to take image copies of all DB2 table spaces, except those that are inconsistent. For those table spaces suspected of being inconsistent, resolve the inconsistencies and then obtain image copies of them.

A cold start might cause down-level page set errors. Some of these errors cause message DSNB232I to be displayed during DB2 restart. After you restart DB2, check the console log for down-level page set messages. If any of those messages exist, correct the errors before you take image copies of the affected data sets. Other down-level page set errors are not detected by DB2 during restart. If you use the COPY utility with the SHRLEVEL REFERENCE option to make image copies, the COPY utility will issue message DSNB232I when it encounters down-level page sets. Correct those errors and continue making image copies. If you use some other method to make image copies, you will find out about down-level errors during normal DB2 operation. "Recovery from down-level page sets" on page 543 describes methods for correcting down-level page set errors.

Pay particular attention to DB2 subsystem table spaces. If any are inconsistent, recover all of them in the order shown in the discussion on recovering catalog and directory objects in Part 2 of *DB2 Utility Guide and Reference*.

When a portion of the DB2 recovery log becomes inaccessible, all DB2 recovery processes have difficulty operating successfully, including restart, RECOVER, and deferred restart processing. Conditional restart allowed circumvention of the problem during the restart process. To ensure that RECOVER does not attempt to access the inaccessible portions of the log, secure a copy (either full or incremental) that does not require such access. A failure occurs any time a DB2 process (such as the RECOVER utility) attempts to access an inaccessible portion of the log. You cannot be sure which DB2 processes must use that portion of the recovery log, and, therefore, you must assume that all data recovery requires that portion of the log.

2. Resolve database inconsistencies. If you determine that the existing inconsistencies involve indexes only (not data), use the utility RECOVER INDEX. If the inconsistencies involve data (either user data or DB2 subsystem data), continue reading this section.

   Inconsistencies in DB2 subsystem databases DSNDB01 and DSNDB06 must be resolved before inconsistencies in other databases can be resolved. This is necessary because the subsystem databases describe all other databases, and access to other databases requires information from DSNDB01 and DSNDB06.

   If the table space that cannot be recovered (and is thus inconsistent) is being dropped, either all rows are being deleted or the table is not necessary. In either case, drop the table when DB2 is restarted, and do not bother to resolve the inconsistencies before restarting DB2.

Any one of the following three procedures can be used to resolve data inconsistencies. However, using one of the first two procedures is advisable because of the complexity of the third procedure.

Method 1. Recover to a prior point of consistency

See "Recovering data to a prior point of consistency" on page 495 for a description of how to successfully prepare for and do data recovery to a prior point of consistency.

Method 2. Re-create the table space

Take the following steps to drop the table space and reconstruct the data using the CREATE statement. This procedure is simple relative to "Method 3. Use the REPAIR utility on the data." However, if you want to use this procedure, you need to plan ahead, because, when a table space is dropped, all tables in that table space, as well as related indexes, authorities, and views, are implicitly dropped. Be prepared to reestablish indexes, views, and authorizations, as well as the data content itself.

DB2 subsystem tables, such as the catalog and directory, cannot be dropped. Follow either "Method 1. Recover to a prior point of consistency" on page 617 or "Method 3. Use the REPAIR utility on the data" for these tables.

1. Issue an SQL DROP TABLESPACE statement for all table spaces that are suspected of being involved in the problem.
2. Re-create the table spaces, tables, indexes, synonyms, and views using SQL CREATE statements.
3. Grant access to these objects as it was granted prior to the time of the error.
4. Reconstruct the data in the tables.
5. Run the RUNSTATS utility on the data.
6. Use COPY to acquire a full image copy of all data.
7. Use the REBIND process on all plans that use the tables or views involved in this activity.

Method 3. Use the REPAIR utility on the data

The third method for resolving data inconsistencies involves the use of the REPAIR utility. This method of resolving inconsistencies is not recommended unless the inconsistency is limited to a small number of data or index pages for the following reasons.

- For extensive data inconsistency, this method can be fairly time consuming and complex, making the procedure more error prone than the two methods described previously.
- DSN1LOGP can identify page sets that contain inconsistencies, but it cannot identify the specific data modifications involved in the inconsistencies within a given page set.
- DB2 provides no mechanism to tell users whether data is physically consistent or damaged. If the data is damaged physically, you might learn this when you attempt to use SQL to access the data and find that the data is inaccessible.

If you decide to use this method to resolve data inconsistencies, be sure to read the following section carefully, because it contains information that is important to the successful resolution of the inconsistencies.

Considerations for using the REPAIR method:

- Any pages that are on the logical page list (perhaps caused by this restart) cannot be accessed via the REPAIR utility. Because you have decided to use the REPAIR utility to resolve the inconsistency, issue the command START DATABASE (*dbase*) SPACENAM (*space*) ACCESS(FORCE), where *space* names the table space involved. That allows access to the data.
- As noted in "Recovering data to a prior point of consistency" on page 495, DB2 subsystem data (in the catalog and directory) exists in interrelated tables and table spaces. Data in DB2 subsystem databases cannot be modified via SQL, so the REPAIR utility must be used to resolve the inconsistencies that are identified.

- For a description of stored data and index formats, refer to Part 6 of *DB2 Diagnosis Guide and Reference*.
- DB2 stores data in data pages. The structure of data in a data page must conform to a set of rules for DB2 to be able to process the data accurately. Using a conditional restart process does not cause violations to this set of rules; but, if violations existed prior to conditional restart, they will continue to exist after conditional restart. Therefore, use DSN1COPY with the CHECK option.
- DB2 uses several types of pointers in accessing data. Each type (indexes, hashes, and links) is described in Part 6 of *DB2 Diagnosis Guide and Reference*. Look for these pointers and manually ensure their consistency.

  Hash and link pointers exist in the DB2 directory database; link pointers also exist in the catalog database. DB2 uses these pointers to access data. During a conditional restart, it is possible for data pages to be modified without update of the corresponding pointers. When this occurs, one of the following actions can occur:

  – If a pointer addresses data that is nonexistent or incorrect, DB2 abends the request. If SQL is used to access the data, a message identifying the condition and the page in question is issued.

  – If data exists but no pointer addresses it, that data is virtually invisible to all functions that attempt to access it via the damaged hash or link pointer. The data might, however, be visible and accessible by some functions, such as SQL functions that use some other pointer that was not damaged. As might be expected, this situation can result in inconsistencies.

  If a row containing a varying-length field is updated, it can increase in size. If the page in which the row is stored does not contain enough available space to store the additional data, the row is placed in another data page, and a pointer to the new data page is stored in the original data page. After a conditional restart, one of the following conditions can exist.

  – The row of data exists, but the pointer to that row does not exist. In this case, the row is invisible and the data cannot be accessed.

  – The pointer to the row exists, but the row itself no longer exists. DB2 abends the requester when any operation (for instance, a SELECT) attempts to access the data. If termination occurs, one or more messages will be received that identify the condition and the page containing the pointer.

  When these inconsistencies are encountered, use the REPAIR utility to resolve them, as described in Part 2 of *DB2 Utility Guide and Reference*.
- If the log has been truncated, there can be problems changing data via the REPAIR utility. Each data and index page contains the log RBA of the last recovery log record that was applied against the page. DB2 does not allow modification of a page containing a log RBA that is higher than the current end of the log. If the log has been truncated and you choose to use the REPAIR utility rather than recovering to a prior point of consistency, the DSN1COPY RESET option must be used to reset the log RBA in every data and index page set to be corrected with this procedure.
- **This last step is imperative.** When all known inconsistencies have been resolved, full image copies of all modified table spaces must be taken, in order to use the RECOVER utility to recover from any future problems.

Part 5. Performance monitoring and tuning

Chapter 24. Planning your performance strategy 627
Managing performance in general 627
Setting reasonable performance objectives 628
Defining the workload 628
Initial performance planning 629
Translating resource requirements into objectives. 630
External design. 630
Internal design 630
Coding and testing 631
Post-development review 631
Planning for performance monitoring 631
Continuous performance monitoring 632
Periodic performance monitoring. 633
Detailed performance monitoring. 633
Exception performance monitoring 634
A performance monitoring strategy 634
Reviewing performance data 634
Typical review questions 635
Are your performance objectives reasonable? 636

Chapter 25. Analyzing performance data . . . 639
Investigating the problem overall. 639
Looking at the entire system 639
Beginning to look at DB2 639
Reading accounting reports from OMEGAMON 640
The accounting report (short format) 640
The accounting report (long format). 641
A general approach to problem analysis in DB2 646

Chapter 26. Improving response time and throughput 651
Reducing I/O operations 651
Using RUNSTATS to keep access path statistics current 651
Reserving free space in table spaces and indexes 652
Specifying free space on pages 652
Determining pages of free space 653
Recommendations for allocating free space 653
Making buffer pools large enough for the workload. 654
Reducing the time needed to perform I/O operations 654
Distributing data sets efficiently 654
Putting frequently used data sets on fast devices 654
Distributing the I/O 655
Creating additional work file table spaces . . . 657
Managing space for I/O performance 658
Formatting early and speed up formatting 658
Avoiding excessive extents 659
Reducing processor resource consumption. . . . 659
Reusing threads for your high-volume transactions 660
Minimizing the use of DB2 traces 660
Global trace 660
Accounting and statistics traces 660
Audit trace 661
Performance trace 661
Using fixed-length records 661
Response time reporting. 661

Chapter 27. Tuning DB2 buffer, EDM, RID, and sort pools 665
Tuning database buffer pools 665
Terminology: Types of buffer pool pages . . . 666
Read operations 666
Write operations 666
Assigning a table space or index to a buffer pool 667
Assigning data to default buffer pools . . . 667
Assigning data to particular buffer pools . . 667
Buffer pool thresholds 667
Fixed thresholds 668
Thresholds you can change. 668
Guidelines for setting buffer pool thresholds 670
Determining size and number of buffer pools 671
Buffer pool sizes 671
The buffer-pool hit ratio. 671
Buffer pool size guidelines 672
Advantages of large buffer pools 673
Choosing one or many buffer pools 673
Choosing a page-stealing algorithm 674
| Long-term page fix option for buffer pools . . 675
Monitoring and tuning buffer pools using online commands 675
Using OMEGAMON to monitor buffer pool statistics 677
| Tuning EDM storage 679
| EDM storage space handling 680
Implications for database design 680
Monitoring and EDM storage 681
Tips for managing EDM storage 682
Use packages 682
Use RELEASE(COMMIT) when appropriate 683
| Be aware of large DBDs 683
Understand the impact of using DEGREE(ANY). 683
Increasing RID pool size. 683
Controlling sort pool size and sort processing . . 684
Estimating the maximum size of the sort pool 684
How sort work files are allocated. 684
Improving the performance of sort processing 685

Chapter 28. Improving resource utilization. . . 687
Managing the opening and closing of data sets . . 687
Determining the maximum number of open data sets 687
How DB2 determines DSMAX. 688
Modifying DSMAX 688
Recommendations. 689

© Copyright IBM Corp. 1982, 2008

Understanding the CLOSE YES and CLOSE NO options . . . 690
The process of closing . . . 690
When the data sets are closed . . . 690
Switching to read-only for infrequently updated and infrequently accessed page sets . . . 691
Planning the placement of DB2 data sets . . . 692
Estimating concurrent I/O requests . . . 692
Crucial DB2 data sets. . . . 692
Changing catalog and directory size and location . . . 693
Monitoring I/O activity of data sets . . . 693
Work file data sets. . . . 693
DB2 logging . . . 694
Logging performance issues and recommendations . . . 694
Log writes . . . 694
Log reads . . . 696
Log capacity. . . . 697
Total capacity and the number of logs . . . 698
Choosing a checkpoint frequency. . . . 698
Calculating average log record size . . . 698
Increasing the number of active log data sets 698
Tips on setting the size of active log data sets 698
Controlling the amount of log data . . . 699
Utilities . . . 699
SQL . . . 700
Improving disk utilization: space and device utilization . . . 701
Allocating and extending data sets . . . 701
Compressing your data . . . 702
Deciding whether to compress. . . . 702
Tuning recommendation. . . . 704
Determining the effectiveness of compression 704
Evaluating your indexes. . . . 705
Eliminating unnecessary partitioning indexes 705
Dropping indexes created to avoid sorts . . . 705
Using nonpadded indexes . . . 706
Improving real storage utilization . . . 706
Performance and storage . . . 708
Real storage . . . 708
Storage devices. . . . 709
Planning considerations for storage device types . . . 709
Storage servers . . . 709
Older storage device types . . . 710
z/OS performance options for DB2 . . . 711
Determining z/OS workload management velocity goals . . . 711
How DB2 assigns I/O priorities . . . 712
IBM System z9 Integrated Information Processor
usage monitoring . . . 713
Controlling resource usage . . . 714
Prioritizing resources . . . 715
Limiting resources for each job . . . 715
Limiting resources for TSO sessions . . . 715
Limiting resources for IMS and CICS . . . 716
Limiting resources for a stored procedure . . . 716
Resource limit facility (governor) . . . 716
Using resource limit tables (RLSTs) . . . 717
Creating an RLST . . . 717
Descriptions of the RLST columns . . . 719
Governing dynamic queries . . . 722
Qualifying rows in the RLST . . . 722
Predictive governing . . . 724
Combining reactive and predictive governing 725
Governing statements from a remote site . . . 725
Calculating service units. . . . 726
Restricting bind operations . . . 726
Example . . . 727
Restricting parallelism modes . . . 727

Chapter 29. Managing DB2 threads . . . 729
Setting thread limits . . . 729
Allied thread allocation . . . 730
Step 1: Thread creation . . . 730
Step 2: Resource allocation . . . 731
Step 3: SQL statement execution . . . 731
Step 4: Commit and thread termination. . . . 732
Variations on thread management . . . 733
TSO and call attachment facility differences 733
Thread management for Resource Recovery Services attachment facility (RRSAF) . . . 733
Differences for SQL under DB2 QMF . . . 733
Providing for thread reuse . . . 733
Bind options for thread reuse . . . 734
Using reports to tell when threads were reused. . . . 734
Database access threads . . . 735
Using allied threads and database access threads 735
Setting thread limits for database access threads 735
Using threads in INACTIVE MODE for DRDA-only connections . . . 736
Understanding the advantages of database access threads in INACTIVE MODE. . . . 737
Enabling threads to be pooled . . . 737
Timing out idle active threads . . . 737
Using threads with private-protocol connections 738
Reusing threads for remote connections . . . 739
Using z/OS workload management to set performance objectives . . . 739
Classifying DDF threads. . . . 739
Establishing performance periods for DDF threads . . . 741
Basic procedure for establishing performance objectives. . . . 742
CICS design options . . . 742
IMS design options . . . 742
TSO design options . . . 743
DB2 QMF design options . . . 744

Chapter 30. Tuning your queries . . . 745
General tips and questions . . . 745
Is the query coded as simply as possible? . . . 745
Are all predicates coded correctly? . . . 745
Are there subqueries in your query?. . . . 746
Does your query involve aggregate functions? 747
Do you have an input variable in the predicate of an SQL query? . . . 748
Do you have a problem with column correlation? . . . 748

Can your query be written to use a noncolumn expression? . . . 748
Can materialized query tables help your query performance? . . . 748
Does the query contain encrypted data? . . . 749
Writing efficient predicates . . . 749
Properties of predicates . . . 749
Predicate types . . . 750
Indexable and nonindexable predicates . . . 751
Stage 1 and stage 2 predicates . . . 751
Boolean term (BT) predicates . . . 752
Predicates in the ON clause . . . 752
General rules about predicate evaluation . . . 753
Order of evaluating predicates. . . . 753
Summary of predicate processing. . . . 754
Examples of predicate properties . . . 759
Predicate filter factors . . . 760
Default filter factors for simple predicates 761
Filter factors for uniform distributions . . . 761
Interpolation formulas . . . 762
Filter factors for all distributions . . . 763
Using multiple filter factors to determine the cost of a query . . . 765
Column correlation . . . 766
How to detect column correlation . . . 766
Impacts of column correlation . . . 767
What to do about column correlation . . . 769
DB2 predicate manipulation . . . 770
Predicate modifications for IN-list predicates 770
When DB2 simplifies join operations . . . 770
Predicates generated through transitive closure . . . 771
Predicates with encrypted data . . . 773
Using host variables efficiently . . . 774
Changing the access path at run time . . . 774
The REOPT(ALWAYS) bind option . . . 774
The REOPT(ONCE) bind option . . . 775
The REOPT(NONE) bind option . . . 776
Rewriting queries to influence access path selection . . . 777
Writing efficient subqueries. . . . 780
Correlated subqueries . . . 780
Noncorrelated subqueries . . . 781
Single-value subqueries . . . 781
Multiple-value subqueries . . . 782
Conditions for DB2 to transform a subquery into a join . . . 782
Subquery tuning . . . 784
Using scrollable cursors efficiently . . . 785
Writing efficient queries on tables with data-partitioned secondary indexes . . . 786
Special techniques to influence access path selection . . . 788
Obtaining information about access paths . . . 789
Fetching a limited number of rows: FETCH FIRST *n* ROWS ONLY . . . 789
Minimizing overhead for retrieving few rows: OPTIMIZE FOR *n* ROWS . . . 790
Favoring index access . . . 792
Using a subsystem parameter to control outer join processing . . . 793
Using the CARDINALITY clause to improve the performance of queries with user-defined table function references . . . 793
Reducing the number of matching columns . . . 794
Creating indexes for efficient star-join processing . . . 796
Recommendations for creating indexes for star-join queries . . . 796
Determining the order of columns in an index for a star schema design . . . 797
Rearranging the order of tables in a FROM clause . . . 798
Updating catalog statistics . . . 799
Using a subsystem parameter . . . 800
Using a subsystem parameter to favor matching index access . . . 800
Using a subsystem parameter to optimize queries with IN-list predicates. . . . 801
Giving optimization hints to DB2. . . . 801
Planning to use optimization hints . . . 801
Enabling optimization hints for the subsystem . . . 802
Scenario: Preventing a change at rebind . . . 802
Scenario: Modifying an existing access path 803
Reasons to use the QUERYNO clause . . . 805
How DB2 locates the PLAN_TABLE rows for a hint . . . 805
How DB2 validates the hint . . . 805
Limitations on optimization hints. . . . 807

Chapter 31. Improving concurrency . . . 809
Definitions of concurrency and locks . . . 810
Effects of DB2 locks . . . 810
Suspension . . . 810
Timeout . . . 811
Deadlock . . . 811
Basic recommendations to promote concurrency 812
Recommendations for system and subsystem options . . . 812
Recommendations for database design . . . 813
Recommendations for application design . . . 814
Aspects of transaction locks . . . 817
The size of a lock . . . 817
Definition . . . 817
Hierarchy of lock sizes . . . 817
General effects of size . . . 818
Effects of table spaces of different types . . . 818
Differences between simple and segmented table spaces . . . 819
The duration of a lock . . . 820
Effects. . . . 820
The mode of a lock . . . 821
Modes of page and row locks . . . 821
Modes of table, partition, and table space locks . . . 822
Lock mode compatibility . . . 823
The object of a lock . . . 823
Definition and examples. . . . 823
Indexes and data-only locking. . . . 824
Locks on the DB2 catalog . . . 824

Locks on the skeleton tables (SKCT and SKPT) . . . 825
Locks on the database descriptors (DBDs) 825
DB2 choices of lock types . . . 826
Modes of locks acquired for SQL statements 826
Lock promotion . . . 829
Lock escalation . . . 829
Modes of transaction locks for various processes . . . 831
Options for tuning locks. . . . 832
IRLM startup procedure options . . . 832
Estimating the storage needed for locks . . 833
Installation options for wait times . . . 833
DEADLOCK TIME on installation panel DSNTIPJ . . . 833
RESOURCE TIMEOUT on installation panel DSNTIPI . . . 833
Wait time for transaction locks . . . 833
IDLE THREAD TIMEOUT on installation panel DSNTIPR . . . 835
UTILITY TIMEOUT on installation panel DSNTIPI . . . 836
Wait time for drains . . . 836
Other options that affect locking . . . 837
LOCKS PER USER field of installation panel DSNTIPJ . . . 837
LOCKSIZE clause of CREATE and ALTER TABLESPACE . . . 838
LOCKMAX clause of CREATE and ALTER TABLESPACE . . . 839
LOCKS PER TABLE(SPACE) field of installation panel DSNTIPJ . . . 840
The option U LOCK FOR RR/RS. . . . 840
Option to release locks for cursors defined WITH HOLD . . . 841
Option XLOCK for searched updates/deletes 841
Option to avoid locks during predicate evaluation . . . 842
Option to disregard uncommitted inserts . . 842
Bind options. . . . 843
The ACQUIRE and RELEASE options . . . 843
Advantages and disadvantages of the combinations . . . 845
The ISOLATION option . . . 846
Advantages and disadvantages of the isolation values. . . . 848
The CURRENTDATA option . . . 853
When plan and package options differ . . . 856
The effect of WITH HOLD for a cursor. . . 857
Isolation overriding with SQL statements . . . 857
The LOCK TABLE statement . . . 859
The purpose of LOCK TABLE . . . 859
When to use LOCK TABLE. . . . 859
The effect of LOCK TABLE. . . . 859
LOB locks . . . 860
Relationship between transaction locks and LOB locks . . . 860
Hierarchy of LOB locks . . . 861
LOB and LOB table space lock modes . . . 861
Modes of LOB locks . . . 862
Modes of LOB table space locks . . . 862
LOB lock and LOB table space lock duration 862
The duration of LOB locks . . . 862
The duration of LOB table space locks . . . 863
Instances when LOB table space locks are not taken . . . 863
Control of the number of LOB locks. . . . 863
Controlling the number of LOB locks that are acquired for a user . . . 863
Controlling LOB lock escalation . . . 863
The LOCK TABLE statement for LOBs . . . 864
LOCKSIZE clause for LOB table spaces. . . . 864
Claims and drains for concurrency control . . . 864
Objects that are subject to takeover . . . 864
Claims . . . 865
Drains. . . . 865
Usage of drain locks . . . 866
Utility locks on the catalog and directory . . . 866
Compatibility of utilities. . . . 867
Concurrency during REORG . . . 868
Utility operations with nonpartitioned indexes 869
Monitoring of DB2 locking . . . 869
Using EXPLAIN to tell which locks DB2 chooses 870
Using the statistics and accounting traces to monitor locking . . . 871
Scenario for analyzing concurrency . . . 872
Scenario description . . . 872
Accounting report . . . 873
Lock suspension report . . . 874
Lockout report . . . 875
Lockout trace . . . 876
Corrective decisions . . . 876
OMEGAMON online locking conflict display 877
Deadlock detection scenarios . . . 877
Scenario 1: Two-way deadlock with two resources . . . 877
Scenario 2: Three-way deadlock with three resources . . . 879

Chapter 32. Using materialized query tables 881
Introduction to materialized query tables and automatic query rewrite . . . 881
Example of automatic query rewrite using a materialized query table. . . . 882
Steps for using automatic query rewrite . . . 882
Defining a materialized query table . . . 883
Creating a new materialized query table . . . 883
Registering an existing table as a materialized query table . . . 884
Altering a materialized query table . . . 886
Populating and maintaining materialized query tables . . . 886
Using the REFRESH TABLE statement . . . 887
Using INSERT, UPDATE, DELETE, and LOAD for materialized query tables . . . 887
Collecting statistics on materialized query tables 888
Using multilevel security with materialized query tables . . . 888
Creating a materialized query table . . . 888
Altering a source table . . . 889
Refreshing a materialized query table . . . 889
Exploiting automatic query rewrite . . . 889

Making materialized query tables eligible . . . 889
Query requirements and the rewrite process . . 891
Query rewrite analysis 891
Examples of automatic query rewrite . . . 892
Determining if query rewrite occurred . . . 896
Recommendations for materialized query table and base table design. 897
Materialized query table design 897
Base table design 898
Materialized query table examples shipped with DB2 898

Chapter 33. Maintaining statistics in the catalog 899
Statistics used for access path selection 899
Filter factors and catalog statistics 906
Statistics for partitioned table spaces 908
Setting default statistics for created temporary tables 908
History statistics 909
Gathering monitor statistics and update statistics 912
Updating the catalog 913
Correlations in the catalog 913
Recommendation for COLCARDF and FIRSTKEYCARDF 914
Recommendation for HIGH2KEY and LOW2KEY 915
Statistics for distributions 915
Recommendation for using the TIMESTAMP column 915
Querying the catalog for statistics 916
Improving index and table space access 916
How clustering affects access path selection . . 917
What other statistics provide index costs . . . 919
When to reorganize indexes and table spaces 920
Reorganizing Indexes. 920
Reorganizing table spaces 922
Reorganizing LOB table spaces 923
Whether to rebind after gathering statistics . . 923
Modeling your production system 923

Chapter 34. Using EXPLAIN to improve SQL performance 927
Obtaining PLAN_TABLE information from EXPLAIN 928
Creating PLAN_TABLE 929
Populating and maintaining a plan table . . . 936
Executing the SQL statement EXPLAIN . . 937
Binding with the option EXPLAIN(YES) . . 937
Executing EXPLAIN under DB2 QMF . . . 937
Maintaining a plan table. 939
Reordering rows from a plan table 939
Retrieving rows for a plan 939
Retrieving rows for a package 939
Asking questions about data access 939
Is access through an index? (ACCESSTYPE is I, I1, N or MX) 940
Is access through more than one index? (ACCESSTYPE=M) 940
How many columns of the index are used in matching? (MATCHCOLS=n) 941
Is the query satisfied using only the index? (INDEXONLY=Y) 942
Is direct row access possible? (PRIMARY_ACCESSTYPE = D) 942
Which predicates qualify for direct row access? 942
Reverting to ACCESSTYPE 943
Using direct row access and other access methods 944
Is a view or nested table expression materialized? 944
Was a scan limited to certain partitions? (PAGE_RANGE=Y) 944
What kind of prefetching is expected? (PREFETCH = L, S, D, or blank) 945
Is data accessed or processed in parallel? (PARALLELISM_MODE is I, C, or X) 945
Are sorts performed? 945
Is a subquery transformed into a join? 946
When are aggregate functions evaluated? (COLUMN_FN_EVAL) 946
How many index screening columns are used? 946
Is a complex trigger WHEN clause used? (QBLOCKTYPE=TRIGGR) 946
Interpreting access to a single table 947
Table space scans (ACCESSTYPE=R PREFETCH=S) 947
Table space scans of nonsegmented table spaces 947
Table space scans of segmented table spaces 948
Table space scans of partitioned table spaces 948
Table space scans and sequential prefetch 948
Overview of index access 948
Using indexes to avoid sorts 948
Costs of indexes 949
Index access paths. 950
Matching index scan (MATCHCOLS>0) . . 950
Index screening. 951
Nonmatching index scan (ACCESSTYPE=I and MATCHCOLS=0) 951
IN-list index scan (ACCESSTYPE=N) . . . 951
Multiple index access (ACCESSTYPE is M, MX, MI, or MU) 952
One-fetch access (ACCESSTYPE=I1) 953
Index-only access (INDEXONLY=Y) 954
Equal unique index (MATCHCOLS=number of index columns) 954
UPDATE using an index 954
Interpreting access to two or more tables (join) . . 955
Definitions and examples of join operations . . 955
Nested loop join (METHOD=1) 957
Method of joining 957
Performance considerations. 958
When nested loop join is used. 958
Merge scan join (METHOD=2) 959
Method of joining 959
Performance considerations. 960
When merge scan join is used 960
Hybrid join (METHOD=4) 961
Method of joining 962

Possible results from EXPLAIN for hybrid join . . . 962
Performance considerations. . . . 962
When hybrid join is used . . . 962
Star join (JOIN_TYPE='S') . . . 963
Example of a star schema . . . 963
When star join is used . . . 964
Dedicated virtual memory pool for star join operations . . . 967
Interpreting data prefetch . . . 969
Sequential prefetch (PREFETCH=S) . . . 969
Dynamic prefetch (PREFETCH=D) . . . 970
List prefetch (PREFETCH=L) . . . 970
The access method . . . 970
When list prefetch is used . . . 971
Bind time and execution time thresholds . . . 971
Sequential detection at execution time . . . 972
When sequential detection is used . . . 972
How to tell whether sequential detection was used . . . 972
How to tell if sequential detection might be used . . . 972
Determining sort activity . . . 973
Sorts of data. . . . 973
Sorts for group by and order by . . . 973
Sorts to remove duplicates . . . 974
Sorts used in join processing . . . 974
Sorts needed for subquery processing . . . 974
Sorts of RIDs . . . 974
The effect of sorts on OPEN CURSOR . . . 974
Processing for views and nested table expressions 975
Merge . . . 975
Materialization . . . 976
Two steps of materialization . . . 976
When views or table expressions are materialized . . . 976
Using EXPLAIN to determine when materialization occurs . . . 978
Using EXPLAIN to determine UNION activity and query rewrite . . . 979
Performance of merge versus materialization 981
Estimating a statement's cost . . . 981
Creating a statement table . . . 982
Populating and maintaining a statement table 984
Retrieving rows from a statement table . . . 984
The implications of cost categories . . . 985

Chapter 35. Parallel operations and query performance . . . 987
Comparing the methods of parallelism . . . 988
Partitioning for optimal parallel performance . . . 990
Determining if a query is I/O- or processor-intensive . . . 991
Determining the number of partitions . . . 991
Working with a table space that is already partitioned . . . 992
Making the partitions the same size . . . 992
Working with partitioned indexes . . . 993
Enabling parallel processing . . . 993
When parallelism is not used . . . 994
Interpreting EXPLAIN output . . . 995
A method for examining PLAN_TABLE columns for parallelism . . . 995
PLAN_TABLE examples showing parallelism 995
Monitoring parallel operations . . . 997
Using DISPLAY BUFFERPOOL . . . 998
Using DISPLAY THREAD . . . 998
Using DB2 trace . . . 998
Accounting trace . . . 998
Performance trace . . . 999
Tuning parallel processing. . . . 1000
Disabling query parallelism . . . 1001

Chapter 36. Tuning and monitoring in a distributed environment . . . 1003
Remote access types: DRDA and private protocol 1003
Characteristics of DRDA . . . 1003
Characteristics of DB2 private protocol . . . 1003
Tuning distributed applications . . . 1004
Application and requesting systems . . . 1004
BIND options . . . 1004
SQL statement options . . . 1004
Block fetching result sets . . . 1005
Optimizing for very large results sets for DRDA . . . 1010
Optimizing for small results sets for DRDA 1011
Data encryption security options . . . 1012
Serving system . . . 1012
Monitoring DB2 in a distributed environment 1013
The DISPLAY command . . . 1013
Tracing distributed events . . . 1013
Reporting server-elapsed time . . . 1017
Monitoring distributed processing with RMF . . . 1017
Duration of an enclave . . . 1017
RMF records for enclaves . . . 1018

Chapter 37. Monitoring and tuning stored procedures and user-defined functions . . . 1019
Controlling address space storage . . . 1019
Assigning procedures and functions to WLM application environments . . . 1020
Providing DB2 cost information for accessing user-defined table functions . . . 1021
Monitoring stored procedures with the accounting trace . . . 1022
Accounting for nested activities . . . 1024
Comparing the types of stored procedure address spaces . . . 1025

Chapter 24. Planning your performance strategy

The first step toward improving performance is planning, which is the subject of this chapter.

For the elements of planning a strategy, see the following sections:
- "Managing performance in general"
- "Setting reasonable performance objectives" on page 628
- "Planning for performance monitoring" on page 631
- "Reviewing performance data" on page 634

Throughout the information about performance monitoring and tuning, remember the following information:
- DB2 is only a part of your overall system. Any change to zSeries hardware, disk subsystems, z/OS, IMS, CICS, TCP/IP, VTAM, the network, WebSphere, or distributed application platforms (such as Windows, UNIX, or Linux) that share your enterprise IT infrastructure can affect how DB2 and its applications run.
- The recommendations for performance monitoring and tuning are based on current knowledge of DB2 performance for "normal" circumstances and "typical" systems. Therefore, that this book provides the best or most appropriate advice for any specific site cannot be guaranteed. In particular, the advice in this sectionon performance often approaches situations from a performance viewpoint only. At some sites, other factors of higher priority may make some performance recommendations in this section inappropriate.
- The recommendations are general. Performance measurements are highly dependent on workload and system characteristics external to DB2.

Information on performance monitoring and tuning in a data sharing environment is presented in *DB2 Data Sharing: Planning and Administration*.

Managing performance in general

Managing the performance of any system involves the following steps:
1. Establish performance objectives.
2. Plan how to monitor performance.
3. Carry out the plan.
4. Analyze performance reports to decide whether the objectives have been met.
5. If performance is thoroughly satisfactory, use one of the following options:
   - Monitor less, because monitoring itself uses resources.
   - Continue monitoring to generate a history of performance to compare with future results.
6. If performance has not been satisfactory, take the following actions:
   a. Determine the major constraints in the system.
   b. Decide where you can afford to make trade-offs and which resources can bear an additional load. Nearly all tuning involves trade-offs among system resources.
   c. Tune your system by adjusting its characteristics to improve performance.
   d. Return to step 3 and continue to monitor the system.

© Copyright IBM Corp. 1982, 2008

Periodically, or after significant changes to your system or workload, return to step 1, reexamine your objectives, and refine your monitoring and tuning strategy accordingly.

Setting reasonable performance objectives

How you define good performance for your DB2 subsystem depends on your particular data processing needs and their priority. Performance objectives should be realistic, in line with your budget, understandable, and measurable.

Common objectives include values for:

- Acceptable response time (a duration within which some percentage of all applications have completed)
- Average throughput (the total number of transactions or queries that complete within a given time)
- System availability, including mean time to failure and the durations of down times

Objectives such as these define the workload for the system and determine the requirements for resources, such as processor speed, amount of storage, additional software, and so on. Often, however, available resources limit the maximum acceptable workload, which requires revising the objectives.

Service-level agreements: Presumably, your users have a say in your performance objectives. A mutual agreement on acceptable performance, between the data processing and user groups in an organization, is often formalized and called a *service-level agreement*. Service-level agreements can include expectations of query response time, the workload throughput per day, hour, or minute, and windows provided for batch jobs (including utilities). These agreements list criteria for determining whether or not the system is performing adequately.

Example: A service-level agreement might require that 90% of all response times sampled on a local network in the prime shift be under 2 seconds, or that the average response time not exceed 6 seconds even during peak periods. (For a network of remote terminals, consider substantially higher response times.)

Performance objectives must reflect not only elapsed time, but also the amount of processing expected. Consider whether to define your criteria in terms of the average, the ninetieth percentile, or even the worst-case response time. Your choice can depend on your site's audit controls and the nature of the workloads.

z/OS workload management (WLM) can manage to the performance objectives in the service-level agreement and provide performance reporting analysis. The terms used in the service-level agreement and the WLM service policy are similar.

Defining the workload

To define the workload of the system, begin by determining the type of workload. For each type, describe a preliminary workload profile that includes the following information:

- A definition of the workload type in terms of its function and its volume. You are likely to have many workloads that perform the same general function (for example, order entry through CICS or IMS transaction managers) and have an identifiable workload profile. Other workload types include SPUFI and QMF

queries, transactions, utilities, and batch jobs. For the volume of a workload that is already processed by DB2, use the summary of its volumes from the DB2 statistics trace.
- The relative priority of the type, including periods during which the priorities change.
- The resources that are required to do the work, including physical resources that are managed by the operating system (such as real storage, disk I/O, and terminal I/O) and logical resources managed by the subsystem (such as control blocks and buffers).

Before installing DB2, gather design data during the phases of initial planning, external design, internal design, and coding and testing. Keep reevaluating your performance objectives with that information.

Initial performance planning

Begin establishing resource requirements by estimating the following quantities. Make an estimate even if these quantities are uncertain at this stage.

For transactions:
- Availability of transaction managers, such as IMS, CICS, or WebSphere
- Number of message pairs for each user function, either to and from a terminal or to and from a distributed application
- Network bandwidth and network device latencies
- Average and maximum number of concurrent users, either terminal operators or distributed application requesters
- Maximum rate of workloads per second, minute, hour, day, or week
- Number of disk I/O operations per user workload
- Average and maximum processor usage per workload type and total workload
- Size of tables
- Effects of objectives on operations and system programming

For query use:
- Time required to key in user data
- Online query processing load
- Limits to be set for the query environment or preformatted queries
- Size of tables
- Effects of objectives on operations and system programming

For batch processing:
- Batch windows for data reorganization, utilities, data definition activities, and BIND processing
- Batch processing load
- Length of batch window
- Number of records to process, data reorganization activity, use of utilities, and data definition activity
- Size of tables and complexity of the queries
- Effects of objectives on operations and system programming

Look at the base estimate to find ways to reduce the workload. Changes in design at this stage, before contention with other programs, are likely to be the most effective. Later, you can compare the actual production profile against the base.

Translating resource requirements into objectives

For each workload type, convert the estimates of resource requirements into measurable objectives. Include statements about the throughput rates to be supported (including any peak periods) and the internal response time profiles to be achieved. Make assumptions about I/O rates, paging rates, and workloads. Include the following factors:

- System response time. You cannot guarantee requested response times before any of the design has been done. Therefore, plan to review your performance targets when you design and implement the system.

  Response times can vary for many reasons. Therefore, include acceptable tolerances in your descriptions of targets. Remember that distributed data processing adds overhead at both the local and remote locations.

  Exclude from the targets any unusual applications that have exceptionally heavy requirements for processing or database access, or establish individual targets for those applications.
- Network response time. Responses in the processor are likely to be in microseconds, whereas responses in the network with appropriate facilities can be about a millisecond. This difference in response times means that an overloaded network can impact the delivery of server responses to user terminals or distributed applications regardless of the speed of the processor.
- Disk response time. I/O operations are generally responsible for much internal processing time. Consider all I/O operations that affect a workload.
- Existing workload. Consider the effects of additional work on existing applications. In planning the capacity of the system, consider the total load on each major resource, not just the load for the new application.
- Business factors. When calculating performance estimates, concentrate on the expected peak throughput rate. Allow for daily peaks (for example, after receipt of mail), weekly peaks (for example, a Monday peak after weekend mail), and seasonal peaks as appropriate to the business. Also allow for peaks of work after planned interruptions, such as preventive maintenance periods and public holidays. Remember that the availability of input data is one of the constraints on throughput.

External design

During the external design phase, you must:

1. Estimate the network, Web server, application server, processor, and disk subsystem workload.
2. Refine your estimates of logical disk accesses. Ignore physical accesses at this stage; one of the major difficulties will be determining the number of I/Os per statement.

Internal design

During the internal design phase, you must:

1. Refine your estimated workload against the actual workload.
2. Refine disk access estimates against database design. After internal design, you can define physical data accesses for application-oriented processes and estimate buffer hit ratios.
3. Add the accesses for DB2 work file database, DB2 log, program library, and DB2 sorts.
4. Consider whether additional processor loads will cause a significant constraint.
5. Refine estimates of processor usage.

6. Estimate the internal response time as the sum of processor time and synchronous I/O time or as asynchronous I/O time, whichever is larger.
7. Prototype your DB2 system. Before committing resources to writing code, you can create a small database, update the statistics stored in the DB2 catalog tables, run SQL statements and examine the results.
8. Use DB2 estimation formulas to develop estimates for processor resource consumption and I/O costs for application processes that are high volume or complex.

Coding and testing

During this phase:

1. Refine the internal design estimates of disk and processing resources.
2. Run the monitoring tools you have selected and check the results against the estimates. You might use a terminal network simulator such as TeleProcessing Network Simulator (TPNS) or other tools to test the system and simulate load conditions.

Post-development review

When you are ready to test the complete system, review its performance in detail. Take the following steps to complete your performance review:

1. Validate system performance and response times against the objectives.
2. Identify resources whose usage requires regular monitoring.
3. Incorporate the observed figures into future estimates. This step requires:
   a. Identifying discrepancies from the estimated resource usage
   b. Identifying the cause of the discrepancies
   c. Assigning priorities to remedial actions
   d. Identifying resources that are consistently heavily used
   e. Setting up utilities to provide graphic representation of those resources
   f. Projecting the processor usage against the planned future system growth to ensure that adequate capacity will be available
   g. Updating the design document with the observed performance figures
   h. Modifying the estimation procedures to reflect what you have learned

You need feedback from users and might have to solicit it. Establish reporting procedures and teach your users how to use them. Consider logging incidents such as these:

- System, line and transaction or query failures
- System unavailable time
- Response times that are outside the specified limits
- Incidents that imply performance constraints, such as deadlocks, deadlock abends, and insufficient storage
- Situations, such as recoveries, that use additional system resources

The data logged should include the time, date, location, duration, cause (if it can be determined), and the action taken to resolve the problem.

Planning for performance monitoring

Your plan for monitoring DB2 performance should include:

- A master schedule of monitoring. Large batch jobs or utility runs can cause activity peaks. Coordinate monitoring with other operations so that it need not conflict with unusual peaks, unless that is what you want to monitor.
- The kinds of analysis to be performed and the tools to be used. Document the data that is extracted from the monitoring output.

  Some of the performance reports are derived from the products that are described in Appendix F, "Using tools to monitor performance," on page 1185. These reports can be produced using Tivoli® Decision Support for OS/390, OMEGAMON XE for DB2 Performance Monitor (DB2 PM), OMEGAMON (which includes the function of DB2 PM), other reporting tools, manual reduction, or a program of your own that extracts information from standard reports.
- A list of people who should review the results. The results of monitoring and the conclusions based on them should be available to the user support group and to system performance specialists.
- A strategy for tuning DB2. Describe how often changes are permitted and standards for testing their effects. Include the tuning strategy in regular system management procedures.

  Tuning recommendations could include generic database and application design changes. You should update development standards and guidelines to reflect your experience and to avoid repeating mistakes.

Typically, your plan provides for four levels of monitoring:
- "Continuous performance monitoring"
- "Periodic performance monitoring" on page 633
- "Detailed performance monitoring" on page 633
- "Exception performance monitoring" on page 634

"A performance monitoring strategy" on page 634 describes a plan that includes all of these levels.

Continuous performance monitoring

For monitoring the basic load of the system, try continually running classes 1, 3, 4, and 6 of the DB2 statistics trace and classes 1 and 3 of the DB2 accounting trace. In the data you collect, look for statistics or counts that differ from past records. Pay special attention to peak periods of activity, both of any new application and of the system as a whole.

Running accounting class 2 as well as class 1 allows you to separate DB2 times from application times.

Running with CICS without the open transaction environment (OTE), entails less need to run with accounting class 2. Application and non-DB2 processing take place under the CICS main TCB. Because SQL activity takes place under the SQL TCB, the class 1 and class 2 times are generally close. The CICS attachment work is spread across class 1, class 2, and time spent processing outside of DB2. Class 1 time thus reports on the SQL TCB time and some of the CICS attachment. If you are concerned about class 2 overhead and you use CICS, you can generally run without turning on accounting class 2.

Statistics and accounting information can be very helpful for application and database designers. Consider putting this information into a performance warehouse so that the data can be analyzed more easily by all the personnel who

need the information. OMEGAMON includes a performance warehouse that allows you to define, schedule, and run processes that:

- Automate the creation of reports
- Automate the conversion and loading of these reports and other data into a performance warehouse
- Analyze the data that is loaded into the performance warehouse using suggested rules or rules that you define yourself

The data in the performance warehouse can be accessed by any member of the DB2 Universal Database family or by any product that supports Distributed Relational Database Architecture (DRDA).

Periodic performance monitoring

A typical periodic monitoring interval of about ten minutes provides information on the workload achieved, resources used, and significant changes to the system. In effect, you are taking "snapshots" at peak loads and under normal conditions.

Recommendation: Monitor peak periods when constraints and response-time problems are more pronounced.

The current peak is also a good indicator of the future average. You might have to monitor more frequently at first to confirm that expected peaks correspond with actual ones. Do not base conclusions on one or two monitoring periods, but on data from several days representing different periods.

Both continuous and periodic monitoring serve to check system throughput, utilized resources (processor, I/Os, and storage), changes to the system, and significant exceptions that might affect system performance. You might notice that subsystem response is becoming increasingly sluggish, or that more applications fail from lack of resources (such as from locking contention or concurrency limits). You also might notice an increase in the processor time DB2 is using, even though subsystem responses seem normal. In any case, if the subsystem continues to perform acceptably and you are not having any problems, DB2 might not need additional tuning.

For periodic monitoring, gather information from z/OS, the transaction manager (IMS, CICS, or WebSphere), the network, the distributed application platforms (such as Windows, UNIX, or Linux), and DB2 itself. To compare the different results from each source, monitor each for the same period of time. Because the monitoring tools require resources, you need to consider the overhead for using these tools. See "Minimizing the use of DB2 traces" on page 660 for information on DB2 trace overhead.

Detailed performance monitoring

Add detailed monitoring to periodic monitoring when you discover or suspect a problem. Use it also to investigate areas not covered periodically. To keep the cost of the detailed monitoring low, limit the information to the specific application and data as much as possible.

If you have a performance problem, first verify that it is not caused by faulty design of an application or database. If you suspect a problem in application design, consult Part 4 of *DB2 Application Programming and SQL Guide*; for information about database design, see Part 2, "Designing a database: advanced topics," on page 23.

If you believe that the problem is caused by the choice of system parameters, I/O device assignments, or other factors, begin monitoring DB2 to collect data about its internal activity. Appendix F, "Using tools to monitor performance," on page 1185 suggests various techniques and methods.

If you have access path problems, refer to Chapter 34, "Using EXPLAIN to improve SQL performance," on page 927 for information.

Exception performance monitoring

Exception monitoring looks for specific exceptional values or events, such as very high response times or deadlocks. Perform exception monitoring for response-time and concurrency problems. For an example, see "Scenario for analyzing concurrency" on page 872.

With OMEGAMON, exception monitoring is available in both batch reporting and Online Monitor. For information on how to use exception processing, set exception threshold limits in the threshold limit data set, and use the exception profiling function of OMEGAMON, see:
- *Using IBM Tivoli OMEGAMON XE on z/OS*
- *OMEGAMON Report Reference*
- *OMEGAMON Monitoring Performance from ISPF*
- *OMEGAMON Monitoring Performance from Performance Expert Client*

A performance monitoring strategy

Consider the following cost factors when planning for performance monitoring and tuning:
- Trace overhead
- Trace data reduction and reporting times
- Time spent on report analysis and tuning action

"Minimizing the use of DB2 traces" on page 660 discusses overhead for global, accounting, statistics, audit, and performance traces.

Reviewing performance data

Inspect your performance data to determine whether performance has been satisfactory, to identify problems, and to evaluate the monitoring process. When establishing requirements and planning to monitor performance, also plan how to review the results of monitoring.

Plan to review the performance data systematically. Review daily data weekly and weekly data monthly; review data more often if a report raises questions that require checking. Depending on your system, the weekly review might require about an hour, particularly after you have had some experience with the process and are able to locate quickly any items that require special attention. The monthly review might take half a day at first, less time later on. But when new applications are installed, workload volumes increased, or terminals added, allow more time for review.

Review the data on a gross level, looking for problem areas. Review details only if a problem arises or if you need to verify measurements.

When reviewing performance data, try to identify the basic pattern in the workload, and then identify variations of the pattern. After a certain period, discard most of the data you have collected, but keep a representative sample. For

example, save the report from the last week of a month for three months; at the end of the year, discard all but the last week of each quarter. Similarly, keep a representative selection of daily and monthly figures. Because of the potential volume of data, consider using Tivoli Decision Support for OS/390, Application Monitor for z/OS and OS/390, or a similar tool to track historical data in a manageable form.

Typical review questions

Use the following questions as a basis for your own checklist. They are not limited strictly to performance items, but your historical data can provide most of their answers. If the performance data is for modeled workloads or changed workloads, the first question to ask for each category is "What changed?" Pointers to more information are also listed.

How often was each transaction and SQL statement used?

1. Considering variations in the workload mix over time, are the monitoring times appropriate?
2. Should monitoring be done more frequently during the day, week, or month to verify this?
3. How many SELECT, INSERT, UPDATE, DELETE, PREPARE, DESCRIBE, DESCRIBE TABLE, PREPARE, OPEN, FETCH, and CLOSE statements are issued per second and per commit?
4. How many IRLM and buffer pool requests are issued per second and per commit?

See "Accounting trace" on page 1190.

How were processor and I/O resources used?

1. Has usage increased for functions that run at a higher priority than DB2 tasks? Examine CICS, IMS, z/OS, JES, TCP/IP, VTAM, WebSphere Application Server, WebSphere MQ (formerly called MQSeries), other subsystems, and key applications.
2. Is the report of processor usage consistent with previous observations?
3. Are scheduled batch jobs able to run successfully?
4. Do any incident reports show that the first invocation of a function takes much longer than later ones? This increased time can happen when programs have to open data sets.
5. What is the CPU time and where is it accumulated? Separate CPU time into accounting TCB and SRB time, and distinguish non-nested, stored procedure, user-defined function, and trigger CPU times. Note the times for DB2 address spaces, DBM1, MSTR, IRLM, and DDF.
6. In a data sharing environment, how are the coupling facility (CF) lock, group buffer pool, and SCA structures performing? What is the peak CF CPU utilization?

See "Monitoring system resources" on page 1188, "Using z/OS, CICS, and IMS tools" on page 1187, and "Statistics trace" on page 1190.

How much real storage was used, and how effective is storage?

1. Is the paging rate increasing? Adequate real storage is very important for DB2 performance.
2. What are the hit ratios for the buffer pools, the EDM pool (the cursor tables and package tables), the EDM DBD cache (the DBDs), and the dynamic statement cache?

See "Monitoring system resources" on page 1188.

To what degree was disk used?

Is the number of I/O requests increasing? DB2 records both physical and logical requests. The number of physical I/Os depend on the configuration of indexes, the data records per control interval, and the buffer allocations.

See "Monitoring system resources" on page 1188 and "Statistics trace" on page 1190.

To what extent were DB2 log resources used?

1. Is the log subject to undue contention from other data sets? In particular, is the log on the same drive as any resource whose updates are logged?

   Recommendation: Do not put a recoverable (updated) resource and a log on the same drive. If that drive fails, you lose both the resource and the log, and you are unable to carry out forward recovery.
2. What is the I/O rate for requests and physical blocks on the log? What is the logging rate for one log in MB per second?

See "Statistics trace" on page 1190.

Do any figures indicate design, coding, or operational errors?

1. Are disk, I/O, log, or processor resources heavily used? If so, was that heavy use expected at design time? If not, can the heavy use be explained in terms of heavier use of workloads?
2. Is the heavy usage associated with a particular application? If so, is there evidence of planned growth or peak periods?
3. What are your needs for concurrent read/write and query activity?
4. How often do locking contentions occur?
5. Are there any disk, channel, or path problems?
6. Are there any abends or dumps?

See "Monitoring system resources" on page 1188, "Statistics trace" on page 1190, and "Accounting trace" on page 1190.

What are the effects of DB2 locks?

1. What are the incidents of deadlocks and timeouts?
2. What percentage of elapsed time is due to lock suspensions? How much lock or latch contention was encountered? Check the contention rate per second by class.
3. How effective is lock avoidance?

See "Monitoring of DB2 locking" on page 869

Were there any bottlenecks?

1. Were any critical thresholds reached?
2. Are any resources approaching high utilization?

See "Monitoring system resources" on page 1188 and "Accounting trace" on page 1190.

Are your performance objectives reasonable?

After beginning to monitor, you need to find out if the objectives themselves are reasonable. Are they achievable, given the hardware available? Are they based upon actual measurements of the workload?

When you measure performance against initial objectives and report the results to users, identify any systematic differences between the measured data and what the user sees. This means investigating the differences between internal response time (seen by DB2) and external response time (seen by the end user). If the measurements differ greatly from the estimates, revise response-time objectives for

the application, upgrade your system, or plan a reduced application workload. If the difference is not too large, however, you can begin tuning the entire system.

Chapter 25. Analyzing performance data

This chapter includes the following topics:

- "Investigating the problem overall"
- "Reading accounting reports from OMEGAMON" on page 640
- "A general approach to problem analysis in DB2" on page 646

Investigating the problem overall

When analyzing performance data, keep in mind that almost all symptoms of poor performance are magnified when contention occurs. For example, if there is a slowdown in disk operations:
- Transactions can pile up, waiting for data set activity.
- Transactions can wait for I/O and locks.
- Paging can be delayed.

In addition, more transactions in the system means greater processor overhead, greater real-storage demand, greater I/O demand, and greater locking demand. This increased overhead and demand can have a large impact.

In such situations, the system shows heavy use of **all** its resources. However, it is actually experiencing typical system stress, with a constraint that is yet to be found.

Looking at the entire system

Start by looking at the overall system before you decide that you have a problem in DB2. In general, look in some detail to see why application processes are progressing slowly, or why a given resource is being heavily used. The best tool for that is the resource measurement facility (RMF™) of z/OS. If the application processes are running on distributed platforms, you must look at the performance on those platforms and the network connecting them with z/OS.

Beginning to look at DB2

Before you analyze DB2 for performance problems, look at the overall system. If your initial analysis suggests that the performance problem is within DB2, the problem might be poor response time, an unexpected and unexplained high use of resources, or locking conflicts. Check factors such as total processor usage, disk activity, and paging.

First, turn on classes 1, 2, and 3 of the accounting trace to get a picture of task activity and determine whether the performance problem is within DB2. Although accounting trace class 2 has a high overhead cost for fetch-intensive applications, it is generally recommended that you keep it on in all cases. For other applications, accounting trace class 2 has an overhead of less than 5%. Accounting trace class 3 can help you determine which resources DB2 is waiting on.

Next, focus on particular activities, such as specific application processes or a specific time interval. You might see problems such as these:
- Slow response time. You could look at detailed traces of one slow task, a problem for which there could be several reasons. For instance, the users could be trying to do too much work with certain applications, work that clearly takes time, and the system simply cannot do all the work that they want done.

© Copyright IBM Corp. 1982, 2008

- Real storage constraints. Applications progress more slowly than expected because of paging interrupts. The constraints show as delays between successive requests recorded in the DB2 trace.
- Contention for a particular function. For example, there might be a wait on a particular data set, or a certain application might cause many application processes to put the same item in their queues. Use the DB2 performance trace to distinguish most of these cases.

For information about packages or DBRMs, run accounting trace classes 7 and 8. To determine which packages are consuming excessive resources, compare accounting trace classes 7 and 8 to the elapsed time for the whole plan on accounting classes 1 and 2.

A number greater than 1 in the QXMAXDEG field of the accounting trace indicates that parallelism was used. There are special considerations for interpreting such records, as described in "Monitoring parallel operations" on page 997.

The easiest way to read and interpret the trace data is through the reports produced by OMEGAMON. If you do not have OMEGAMON or an equivalent program, refer to Appendix D, "Interpreting DB2 trace output," on page 1135 for information about the format of data from DB2 traces.

You can also use the tools for performance measurement described in Appendix F, "Using tools to monitor performance," on page 1185 to diagnose system problems. Also see Appendix F, "Using tools to monitor performance," on page 1185 for information on analyzing the DB2 catalog and directory.

Reading accounting reports from OMEGAMON

You can obtain OMEGAMON reports of accounting data in long or short format and in various levels of detail. The examples of OMEGAMON reports in this book are based on the default formats, which might have been modified for your installation. Furthermore, the OMEGAMON reports have been reformatted or modified for this publication. Refer to *OMEGAMON Report Reference* for an exact description of each report. See "Accounting for nested activities" on page 1024 for information on time results for triggers, stored procedures, and user-defined functions.

Although this section talks about OMEGAMON reports, you can also use DB2 PM to obtain the same reports.

The accounting report (short format)

General capabilities: The OMEGAMON accounting report, short format, allows you to monitor application distribution, resources used by each major group of applications, and the average DB2 elapsed time for each major group. The report summarizes application-related performance data and orders the data by selected DB2 identifiers.

Monitoring application distribution helps you to identify the most frequently used transactions or queries, and is intended to cover the 20% of the transactions or queries that represent about 80% of the total work load. The TOP list function of OMEGAMON lets you identify the report entries that represent the largest user of a given resource.

To get an overall picture of the system work load, you can use the OMEGAMON GROUP command to group several DB2 plans together.

You can use the accounting report, short format, to:

- Monitor the effect of each application or group on the total work load
- Monitor, in each application or group:
  - DB2 response time (elapsed time)
  - Resources used (processor, I/Os)
  - Lock suspensions
  - Application changes (SQL used)
  - Usage of packages and DBRMs
  - Processor, I/O wait, and lock wait time for each package

An accounting report in the short format can list results in order by package. Thus you can summarize package or DBRM activity independently of the plan under which the package or DBRM executed.

Only class 1 of the accounting trace is needed for a report of information by plan. Classes 2 and 3 are recommended for additional information. Classes 7 and 8 are needed to give information by package or DBRM.

The accounting report (long format)

Use the OMEGAMON accounting report, short format, to monitor your applications. Use the OMEGAMON accounting report, long format, when an application seems to have a problem, and you need a more detailed analysis. For a partial example of an accounting report, long format, see Figure 64 on page 642.

```
AVERAGE       APPL(CL.1)  DB2 (CL.2)  IFI (CL.5)    CLASS 3 SUSPENSIONS   AVERAGE TIME  AV.EVENT    HIGHLIGHTS
------------  ----------  ----------  ----------    --------------------  ------------  --------    --------------------------
ELAPSED TIME    0.072929    0.029443         N/P    LOCK/LATCH(DB2+IRLM)  A  0.000011      0.04    #OCCURRENCES    :      193
 NONNESTED      0.072929    0.029443         N/A    SYNCHRON. I/O  B      0.010170      9.16    #ALLIEDS        :        0
 STORED PROC    0.000000    0.000000         N/A     DATABASE I/O         0.006325      8.16    #ALLIEDS DISTRIB:        0
 UDF            0.000000    0.000000         N/A     LOG WRITE I/O        0.003845      1.00    #DBATS          :      193
 TRIGGER        0.000000    0.000000         N/A    OTHER READ I/O  D     0.000000      0.00    #DBATS DISTRIB. :        0
                                                     OTHER WRTE I/O  E     0.000148      0.04    #NO PROGRAM DATA:        0
CPU TIME        0.026978 C  0.018994         N/P    SER.TASK SWTCH  F     0.000115      0.04    #NORMAL TERMINAT:      193
 AGENT          0.026978    0.018994         N/A     UPDATE COMMIT        0.000000      0.00    #ABNORMAL TERMIN:        0
  NONNESTED     0.026978    0.018994         N/P     OPEN/CLOSE           0.000000      0.00    #CP/X PARALLEL. :        0
  STORED PRC    0.000000    0.000000         N/A     SYSLGRNG REC         0.000115      0.04    #IO PARALLELISM :        0
  UDF           0.000000    0.000000         N/A     EXT/DEL/DEF          0.000000      0.00    #INCREMENT. BIND:        0
  TRIGGER       0.000000    0.000000         N/A     OTHER SERVICE        0.000000      0.00    #COMMITS        :      193
 PAR.TASKS      0.000000    0.000000         N/A    ARC.LOG(QUIES)  G     0.000000      0.00    #ROLLBACKS      :        0
                                                     ARC.LOG READ  H       0.000000      0.00    #SVPT REQUESTS  :        0
SUSPEND TIME    0.000000    0.010444         N/A    DRAIN LOCK  I         0.000000      0.00    #SVPT RELEASE   :        0
 AGENT               N/A    0.010444         N/A    CLAIM RELEASE  J      0.000000      0.00    #SVPT ROLLBACK  :        0
 PAR.TASKS           N/A    0.000000         N/A    PAGE LATCH  K         0.000000      0.00    MAX SQL CASC LVL:        0
 STORED PROC    0.000000         N/A         N/A    NOTIFY MSGS           0.000000      0.00    UPDATE/COMMIT   :    40.00
 UDF            0.000000         N/A         N/A    GLOBAL CONTENTION     0.000000      0.00    SYNCH I/O AVG.  : 0.001110
                                                     COMMIT PH1 WRITE I/O  0.000000      0.00
NOT ACCOUNT.  L     N/A    0.000004         N/A    ASYNCH CF REQUESTS    0.000000      0.00
DB2 ENT/EXIT        N/A      182.00         N/A    TOTAL CLASS 3         0.010444      9.28
EN/EX-STPROC        N/A        0.00         N/A
EN/EX-UDF           N/A        0.00         N/A
DCAPT.DESCR.        N/A         N/A         N/P
LOG EXTRACT.        N/A         N/A         N/P

SQL DML   AVERAGE    TOTAL    SQL DCL           TOTAL    SQL DDL    CREATE   DROP  ALTER    LOCKING                  AVERAGE    TOTAL
-------- -------- --------    -------------- --------    ---------- ------ ------ ------    ---------------------- -------- --------
SELECT      20.00     3860    LOCK TABLE            0    TABLE           0      0      0    TIMEOUTS                   0.00        0
INSERT       0.00        0    GRANT                 0    CRT TTABLE      0    N/A    N/A    DEADLOCKS                  0.00        0
UPDATE      30.00     5790    REVOKE                0    DCL TTABLE      0    N/A    N/A    ESCAL.(SHARED)             0.00        0
DELETE      10.00     1930    SET CURR.SQLID        0    AUX TABLE       0    N/A    N/A    ESCAL.(EXCLUS)             0.00        0
                              SET HOST VAR.         0    INDEX           0      0      0    MAX PG/ROW LOCKS HELD     43.34       47
DESCRIBE     0.00        0    SET CUR.DEGREE        0    TABLESPACE      0      0      0    LOCK REQUEST              63.82    12318
DESC.TBL     0.00        0    SET RULES             0    DATABASE        0      0      0    UNLOCK REQUEST            14.48     2794
PREPARE      0.00        0    SET CURR.PATH         0    STOGROUP        0      0      0    QUERY REQUEST              0.00        0
OPEN        10.00     1930    SET CURR.PREC.        0    SYNONYM         0      0    N/A    CHANGE REQUEST            33.35     6436
FETCH       10.00     1930    CONNECT TYPE 1        0    VIEW            0      0    N/A    OTHER REQUEST              0.00        0
CLOSE       10.00     1930    CONNECT TYPE 2        0    ALIAS           0      0    N/A    LOCK SUSPENSIONS           0.00        0
                              SET CONNECTION        0    PACKAGE       N/A      0    N/A    IRLM LATCH SUSPENSIONS     0.03        5
                              RELEASE               0    PROCEDURE       0      0      0    OTHER SUSPENSIONS          0.00        0
DML-ALL     90.00    17370    CALL                  0    FUNCTION        0      0      0    TOTAL SUSPENSIONS          0.03        5
                              ASSOC LOCATORS        0    TRIGGER         0      0    N/A
                              ALLOC CURSOR          0    DIST TYPE       0      0    N/A
                              HOLD LOCATOR          0    SEQUENCE        0      0      0
                              FREE LOCATOR          0
                              DCL-ALL               0    TOTAL           0      0      0
                                                         RENAME TBL      0
                                                         COMMENT ON      0
                                                         LABEL ON        0
```

Figure 64. Partial accounting report (long format)

In analyzing a detailed accounting report, consider the following components of response time. (Fields of the report that are referred to are labeled in Figure 64.)

Class 1 elapsed time: Compare this with the CICS, IMS, WebSphere, or distributed application transit times:

- In CICS, you can use CICS Performance Analyzer to find the attach and detach times; use this time as the transit time.
- In IMS, use the PROGRAM EXECUTION time reported in IMS Performance Analyzer.

Differences between the CICS, IMS, WebSphere, or distributed application times and the DB2 accounting times arise mainly because the DB2 times do not include:

- Time before the first SQL statement, which for a distributed application includes the inbound network delivery time
- DB2 create thread
- DB2 terminate thread

- For a distributed application, the time to deliver the response to a commit if database access thread pooling is used

Differences can also arise from thread reuse in CICS, IMS, WebSphere, or distributed application processing or through multiple commits in CICS. If the class 1 elapsed time is significantly less than the CICS or IMS time, check the report from Tivoli Decision Support for OS/390, IMS Performance Analyzer, or an equivalent reporting tool to find out why. Elapsed time can occur:

- In DB2, during sign-on, create, or terminate thread
- Outside DB2, during CICS or IMS processing

Not-in-DB2 time: This is time calculated as the difference between the class 1 and the class 2 elapsed time. It is time spent outside of DB2, but within the DB2 accounting interval. A lengthy time can be caused by thread reuse, which can increase class 1 elapsed time, or a problem in the application program, CICS, IMS, or the overall system.

For a distributed application, not-in-DB2 time is calculated with this formula:

```
Not_in_DB2 time = A - (B + C + (D - E))
```

Where the variables have the following values:

A Class 1 elapsed time

B Class 2 non-tested elapsed time

C Class 1 non-tested time of any stored procedures, user-defined functions, or triggers

D Class 1 non-tested CPU time

E Class 2 non-tested CPU time

The calculated not-in-DB2 time might be zero. Furthermore, this time calculation is only an estimate. A primary factor that is not included in the equation is the amount of time that requests wait for CPU resources while executing within the DDF address space. To determine how long requests wait for CPU resources, look at the NOT ACCOUNT field. The NOT ACCOUNT field shows the time that requests wait for CPU resources while a distributed task is inside DB2.

Lock/latch suspension time: This shows contention for DB2 and IRLM resources. If contention is high, check the locking summary section of the report, and then proceed with the locking reports. For more information, see "Scenario for analyzing concurrency" on page 872.

In the OMEGAMON accounting report, see the field LOCK/LATCH(DB2+IRLM) (**A**).

Synchronous I/O suspension time: This is the total application wait time for synchronous I/Os. It is the total of database I/O and log write I/O. In the OMEGAMON accounting report, check the number reported for SYNCHRON. I/O (**B**).

If the number of synchronous read or write I/Os is higher than expected, check for:

- A change in the access path to data. If you have data from accounting trace class 8, the number of synchronous and asynchronous read I/Os is available for individual packages. Determine which package or packages have unacceptable

counts for synchronous and asynchronous read I/Os. Activate the necessary performance trace classes for the OMEGAMON SQL activity reports to identify the SQL statement or cursor that is causing the problem. If you suspect that your application has an access path problem, see Chapter 34, "Using EXPLAIN to improve SQL performance," on page 927.

- A lower than expected buffer pool hit ratio. You can improve the ratio by tuning the buffer pool. Look at the number of synchronous reads in the buffer pool that are associated with the plan, and look at the related buffer pool hit ratio. If the buffer pool size and the buffer pool hit ratio for random reads is small, consider increasing the buffer pool size. By increasing the buffer pool size, you might reduce the amount of synchronous database I/O and reduce the synchronous I/O suspension time.
- A change to the catalog statistics, or statistics that no longer match the data.
- Changes in the application. Check the "SQL ACTIVITY" section and compare with previous data. There might have been some inserts that changed the amount of data. Also, check the names of the packages or DBRMs being executed to determine if the pattern of programs being executed has changed.
- Pages might be out of order so that sequential detection is not used, or data might have been moved to other pages. Run the REORG utility in these situations.
- A system-wide problem in the database buffer pool. Refer to "Using OMEGAMON to monitor buffer pool statistics" on page 677. You can also use OMEGAMON Buffer Pool Analyzer User's Guide or OMEGAMON, which contains the function of DB2 BPA, to manage and optimize buffer pool activity.
- A RID pool failure. Refer to "Increasing RID pool size" on page 683.
- A system-wide problem in the EDM pool. Refer to "Tuning EDM storage" on page 679.

If I/O time is greater than expected, and not caused by more read I/Os, check for:

- Synchronous write I/Os. See "Using OMEGAMON to monitor buffer pool statistics" on page 677. (You can also use OMEGAMON, which contains the function of DB2 BPA, to manage and buffer pool activity.)
- I/O contention. In general, each synchronous read I/O typically takes from 5 to 20 milliseconds, depending on the disk device. This estimate assumes that there are no prefetch or deferred write I/Os on the same device as the synchronous I/Os. Refer to "Monitoring I/O activity of data sets" on page 693.
- An increase in the number of users, or an increase in the amount of data. Also, drastic changes to the distribution of the data can cause the problem.

Processor resource consumption: The problem might be caused by DB2 or IRLM traces, a change in access paths, or an increase in the number of users. In the OMEGAMON accounting report, DB2 processor resource consumption is indicated in the field for class 2 CPU TIME (**C**).

Other read suspensions: The accumulated wait time for read I/O done under a thread other than this one. It includes time for:

- Sequential prefetch
- List prefetch
- Sequential detection
- Synchronous read I/O performed by a thread other than the one being reported

Generally, an asynchronous read I/O for sequential prefetch or sequential detection takes about 0.1 to 1 milliseconds per page. List prefetch takes about 0.2 to 2 milliseconds per page.

In the OMEGAMON accounting report, other read suspensions are reported in the field OTHER READ I/O (**D**).

Other write suspensions: The accumulated wait time for write I/O done under a thread other than this one. It includes time for:
- Asynchronous write I/O
- Synchronous write I/O performed by a thread other than the one being reported

As a guideline, an asynchronous write I/O takes 0.1 to 2 milliseconds per page.

In theOMEGAMON accounting report, other write suspensions are reported in the field OTHER WRTE I/O (**E**).

Service task suspensions: The accumulated wait time from switching synchronous execution units, by which DB2 switches from one execution unit to another. The most common contributors to service task suspensions are:
- Wait for phase 2 commit processing for updates, inserts, and deletes (UPDATE COMMIT). You can reduce this wait time by allocating the DB2 primary log on a faster disk. You can also help to reduce the wait time by reducing the number of commits per unit of work.
- Wait for OPEN/CLOSE service task (including HSM recall). You can minimize this wait time by using two strategies. If DSMAX is frequently reached, increase DSMAX. If DSMAX is not frequently reached, change CLOSE YES to CLOSE NO on data sets that are used by critical applications.
- Wait for SYSLGRNG recording service task.
- Wait for data set extend/delete/define service task (EXT/DEL/DEF). You can minimize this wait time by defining larger primary and secondary disk space allocation for the table space.
- Wait for other service tasks (OTHER SERVICE).

In the OMEGAMON accounting report, the total of this information is reported in the field SER.TASK SWTCH (**F**). The field is the total of the five fields that follow it. If several types of suspensions overlap, the sum of their wait times can exceed the total clock time that DB2 spends waiting. Therefore, when service task suspensions overlap other types, the wait time for the other types of suspensions is not counted.

Archive log mode (QUIESCE): The accumulated time the thread was suspended while processing ARCHIVE LOG MODE(QUIESCE). In the OMEGAMON accounting report, this information is reported in the field ARCH.LOG (QUIES) (**G**).

Archive log read suspension: This is the accumulated wait time the thread was suspended while waiting for a read from an archive log on tape. In the OMEGAMON accounting report, this information is reported in the field ARCHIVE LOG READ (**H**).

Drain lock suspension: The accumulated wait time the thread was suspended while waiting for a drain lock. If this value is high, see “Installation options for

wait times" on page 833, and consider running the OMEGAMON locking reports for additional detail. In the OMEGAMON accounting report, this information is reported in the field DRAIN LOCK (**I**).

Claim release suspension: The accumulated wait time the drainer was suspended while waiting for all claim holders to release the object. If this value is high, see "Installation options for wait times" on page 833, and consider running the OMEGAMON locking reports for additional details.

In the OMEGAMON accounting report, this information is reported in the field CLAIM RELEASE (**J**).

Page latch suspension: This field shows the accumulated wait time because of page latch contention. As an example, when the RUNSTATS and COPY utilities are run with the SHRLEVEL(CHANGE) option, they use a page latch to serialize the collection of statistics or the copying of a page. The page latch is a short duration "lock". If this value is high, the OMEGAMON locking reports can provide additional data to help you determine which object is the source of the contention.

In a data sharing environment, high page latch contention could occur in a multithreaded application that runs on multiple members and requires many inserts. The OMEGAMON lock suspension report shows this suspension for page latch contention in the "other" category. If the suspension is on the index leaf page, use one of the following strategies:

- Make the inserts random
- Drop the index
- Perform the inserts from a single member

If the page latch suspension is on a space map page, use the member cluster option for the table space.

In the OMEGAMON accounting report, this information is reported in the field PAGE LATCH (**K**).

Not-accounted-for DB2 time: The DB2 accounting class 2 elapsed time that is not recorded as class 2 CPU time or class 3 suspensions. The most common contributors to this category are:

- z/OS paging
- Processor wait time
- On DB2 requester systems, the amount of time waiting for requests to be returned from either VTAM or TCP/IP, including time spent on the network and time spent handling the request in the target or server systems
- Time spent waiting for parallel tasks to complete (when query parallelism is used for the query)
- Some online performance monitoring

In the OMEGAMON accounting report, this information is reported in the field NOT ACCOUNT (**L**).

A general approach to problem analysis in DB2

The following is a suggested sequence for investigating a response-time problem:

1. If the problem is inside DB2, determine which plan has the longest response time. If the plan can potentially allocate many different packages or DBRMs, determine which packages or DBRMs have the longest response time. Or, if you have a record of past history, determine which transactions show the largest increases.

   Compare class 2 CPU time, class 3 time, and not accounted time. If your performance monitoring tool does not specify times other than class 2 and class 3, then you can determine the not accounted for time with the following formula:

   ```
   Not accounted time = Class 2 elapsed time - Class 2 CPU time - Total class 3 time
   ```

2. If the class 2 CPU time is high, investigate by doing the following:
   - Check to see whether unnecessary trace options are enabled. Excessive performance tracing can be the reason for a large increase in class 2 CPU time.
   - Check the SQL statement count, the getpage count, and the buffer update count on the OMEGAMON accounting report. If the profile of the SQL statements has changed significantly, review the application. If the getpage counts or the buffer update counts change significantly, check for changes to the access path and significant increases in the number of rows in the tables that are accessed.
   - Use the statistics report to check buffer pool activity, including the buffer pool thresholds. If buffer pool activity has increased, be sure that your buffer pools are properly tuned. For more information on buffer pools, see "Tuning database buffer pools" on page 665.
   - Use EXPLAIN to check the efficiency of the access paths for your application. Based on the EXPLAIN results:
     - Use package-level accounting reports to determine which package or DBRM has a long elapsed time. In addition, use the class 7 CPU time for packages to determine which package or DBRM has the largest CPU time or the greatest increase in CPU time.
     - Use the OMEGAMON SQL activity report to analyze specific SQL statements. You can also use OMEGAMON to analyze specific SQL statements, including the currently running SQL statement.
     - If you have a history of the performance of the affected application, compare current EXPLAIN output to previous access paths and costs.
     - Check that RUNSTATS statistics are current.
     - Check that databases have been reorganized using the REORG utility.
     - Check which indexes are used and how many columns are accessed. Has your application used an alternative access path because an index was dropped?
     - Examine joins and subqueries for efficiency.

     See Chapter 34, "Using EXPLAIN to improve SQL performance," on page 927 for help in understanding access path selection and analyzing access path problems. DB2 Visual Explain can give you a graphic display on your workstation of your EXPLAIN output.
   - Check the counts in the locking section of the OMEGAMON accounting report. If locking activity has increased, see Chapter 31, "Improving concurrency," on page 809. For a more detailed analysis, use the deadlock or timeout traces from statistics trace class 3 and the lock suspension report or trace.

3. If class 3 time is high, check the individual types of suspensions in the "Class 3 Suspensions" section of the OMEGAMON accounting report. (The fields referred to here are in Figure 64 on page 642).
   - If LOCK/LATCH (**A**), DRAIN LOCK (**I**), or CLAIM RELEASE (**J**) time is high, see Chapter 31, "Improving concurrency," on page 809.
   - If SYNCHRON. I/O (**B**) time is high, see "Synchronous I/O suspension time" on page 643.
   - If OTHER READ I/O (**D**) time is high, check prefetch I/O operations, disk contention and the tuning of your buffer pools.
   - If OTHER WRITE I/O (**E**) time is high, check the I/O path, disk contention, and the tuning of your buffer pools.
   - If SER.TASK SWTCH (**F**) is high, check open and close activity, as well as commit activity. A high value could also be caused by:
     – SYSLGRNG recording service
     – Data set extend/delete/define service

     Consider also, the possibility that DB2 is waiting for Hierarchical Storage Manager (HSM) to recall data sets that had been migrated to tape. The amount of time that DB2 waits during the recall is specified on the RECALL DELAY parameter on installation panel DSNTIPO.

   If accounting class 8 trace was active, each of these suspension times is available on a per-package or per-DBRM basis in the package block of the OMEGAMON accounting report.
4. If NOT ACCOUNT. (**L**) time is high, check for paging activity, processor wait time, return wait time for requests to be returned from VTAM or TCP/IP, wait time for completion of parallel tasks, and the use of online performance monitors. Turn off or reduce the intensity of online monitoring to eliminate or significantly reduce a high NOT ACCOUNT time that is caused by some monitors. A high NOT ACCOUNT time is acceptable if it is caused by wait time for completion of parallel tasks.
   - Use RMF reports to analyze paging, CPU utilization, and other workload activities.
   - Check the SER.TASK SWTCH field in the "Class 3 Suspensions" section of the OMEGAMON accounting reports.

Figure 65 on page 649 shows which reports you might use, depending on the nature of the problem, and the order in which to look at them.

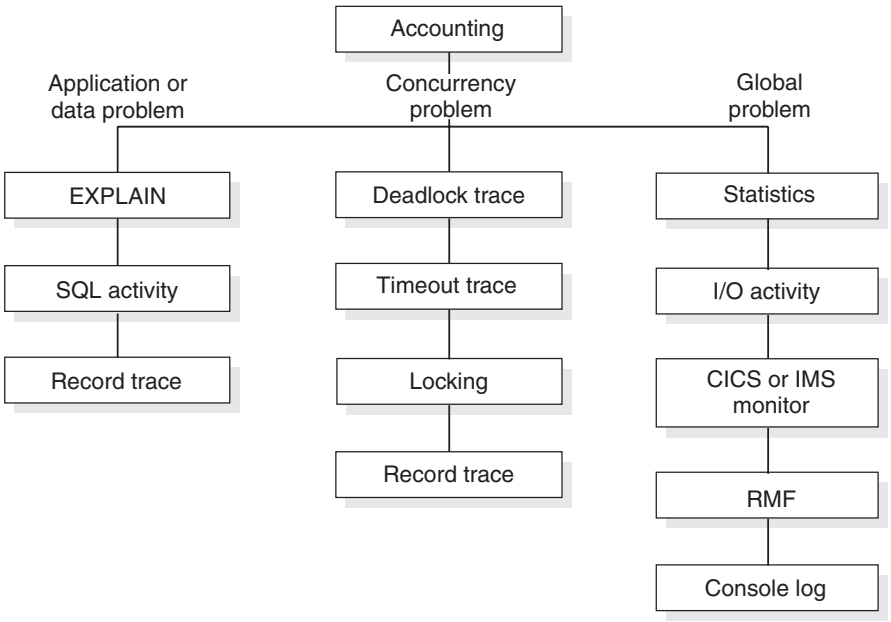


Figure 65. OMEGAMON reports used for problem analysis

If you suspect that the problem is in DB2, it is often possible to discover its general nature from the accounting reports. You can then analyze the problem in detail based on one of the branches shown in Figure 65:

- Follow the first branch, **Application or data problem**, when you suspect that the problem is in the application itself or in the related data. Also use this path for a further breakdown of the response time when no reason can be identified.
- The second branch, **Concurrency problem**, shows the reports required to investigate a lock contention problem. This is illustrated in "Scenario for analyzing concurrency" on page 872.
- Follow the third branch for a **Global problem**, such as an excessive average elapsed time per I/O. A wide variety of transactions could suffer similar problems.

Before starting the analysis in any of the branches, start the DB2 trace to support the corresponding reports. When starting the DB2 trace:

- Refer to *OMEGAMON Report Reference* for the types and classes needed for each report.
- To make the trace data available as soon as an experiment has been carried out, and to avoid flooding the SMF data sets with trace data, use a GTF data set as the destination for DB2 performance trace data.

  Alternatively, use the Collect Report Data function in OMEGAMON to collect performance data. You specify only the report set, not the DB2 trace types or classes you need for a specific report. Collect Report Data lets you collect data in a TSO data set that is readily available for further processing. No SMF or GTF handling is required.
- To limit the amount of trace data collected, you can restrict the trace to particular plans or users in the reports for SQL activity or locking. However, you cannot so restrict the records for performance class 4, which traces asynchronous I/O for specific page sets. You might want to consider turning on selective traces and be aware of the added costs incurred by tracing.

If the problem is not in DB2, check the appropriate reports from a CICS or IMS reporting tool.

When CICS or IMS reports identify a commit, the time stamp can help you locate the corresponding OMEGAMON accounting trace report.

You can match DB2 accounting records with CICS accounting records. If you specify ACCOUNTREC(UOW) or ACCOUNTREC(TASK) on the DB2ENTRY RDO definition, the CICS LU 6.2 token is included in the DB2 trace records, in field QWHCTOKN of the correlation header. To help match CICS and DB2 accounting records, specify ACCOUNTREC(UOW) or ACCOUNTREC(TASK) in the DB2ENTRY definition. That writes a DB2 accounting record after every transaction. As an alternative, you can produce OMEGAMON accounting reports that summarize accounting records by CICS transaction ID. Use the OMEGAMON function Correlation Translation to select the subfield containing the CICS transaction ID for reporting.

You can synchronize the statistics recording interval with the RMF reporting interval, using the STATIME and SYNCVAL subsystem parameters. STATIME specifies the length of the statistics interval, and SYNCVAL specifies that the recording interval is synchronized with some part of the hour.

Example: If the RMF reporting interval is 15 minutes, you can set the STATIME to 15 minutes and SYNCVAL to 0 to synchronize with RMF at the beginning of the hour. These values cause DB2 statistics to be recorded at 15, 30, 45, and 60 minutes past the hour, matching the RMF report interval. Alternatively, because the RMF reporting interval is typically in multiples of 5 minutes, you could specify 0 for STATIME and 5 for SYNCVAL to synchronize the reporting times. These values produce synchronized statistics reports with more granularity and better synchronization.

Synchronizing the statistics recording interval across data sharing members and with the RMF reporting interval is helpful because having the DB2 statistics, RMF, and CF data for identical time periods makes the problem analysis more accurate.

Chapter 26. Improving response time and throughput

Response time consists of the following three components:

- Processor resource consumption, which is shown on Figure 64 on page 642 as "CPU TIME".
- Wait time traced in accounting class 3, which includes:
  - I/O wait time (synchronous and asynchronous)
  - Lock and latch wait time
- Other time

In general, you can improve the response time and throughput of your DB2 applications and queries by:

- "Reducing I/O operations"
- "Reducing the time needed to perform I/O operations" on page 654
- "Reducing processor resource consumption" on page 659
- "Response time reporting" on page 661

The chapter concludes with an overview of how various DB2 response times are reported.

Parallel processing, which is described in Chapter 35, "Parallel operations and query performance," on page 987, can also improve response times. And, DB2 data sharing is also a possible solution for increasing throughput in your system, as well as an opportunity for an improved price for performance ratio. For more information about data sharing, see *DB2 Data Sharing: Planning and Administration*.

Reducing I/O operations

Reducing the number of I/O operations is one way to improve the response time of your applications and queries. This section describes the following ways you can minimize I/O operations:

- "Using RUNSTATS to keep access path statistics current"
- "Reserving free space in table spaces and indexes" on page 652
- "Making buffer pools large enough for the workload" on page 654
- "Formatting early and speed up formatting" on page 658

Using indexes can also minimize I/O operations. For information on indexes and access path selection see "Overview of index access" on page 948.

Using RUNSTATS to keep access path statistics current

The RUNSTATS utility collects statistics about DB2 objects. These statistics can be stored in the DB2 catalog and are used during the bind process to choose the path in accessing data. If you never use RUNSTATS and subsequently rebind your packages or plans, DB2 will not have the information it needs to choose the most efficient access path. This can result in unnecessary I/O operations and excessive processor consumption. See "Gathering monitor statistics and update statistics" on page 912 for more information on using RUNSTATS.

Run RUNSTATS at least once against each table and its associated indexes. How often you rerun the utility depends on how current you need the catalog data to

© Copyright IBM Corp. 1982, 2008

be. If data characteristics of the table vary significantly over time, you should keep the catalog current with those changes. RUNSTATS is most beneficial for the following:

- Table spaces that contain frequently accessed tables
- Tables involved in a sort
- Tables with many rows
- Tables against which SELECT statements having many search arguments are performed

For some tables, you will find no good time to run RUNSTATS. For example, you might use some tables for work that is in process. The tables might have only a few rows in the evening when it is convenient to run RUNSTATS, but they might have thousands or millions of rows in them during the day. For such tables, consider these possible approaches:

- Set the statistics to a relatively high number and hope your estimates are appropriate.
- Use volatile tables. For information about defining a table as volatile, see *DB2 SQL Reference*.

Whichever approach that you choose, monitor the tables because optimization is adversely affected by incorrect information.

Reserving free space in table spaces and indexes

General-use Programming Interface

You can use the PCTFREE and FREEPAGE clauses of the CREATE and ALTER TABLESPACE statements and CREATE and ALTER INDEX statements to improve the performance of INSERT and UPDATE operations. The table spaces and indexes for the DB2 catalog can also be altered to modify FREEPAGE and PCTFREE. These options are not applicable for LOB table spaces.

You can change the values of PCTFREE and FREEPAGE for existing indexes and table spaces using the ALTER INDEX and ALTER TABLESPACE statements, but the change has no effect until you load or reorganize the index or table space.

When you specify a sufficient amount of free space, the advantages during normal processing are:

Better clustering of rows (giving faster access)
Fewer overflows
Less frequent reorganizations needed
Less information locked by a page lock
Fewer index page splits

The disadvantages are:

More disk space occupied
Less information transferred per I/O
More pages to scan
Possibly more index levels
Less efficient use of buffer pools and storage controller cache

Specifying free space on pages

The PCTFREE clause specifies what percentage of each page in a table space or index is left free when loading or reorganizing the data. DB2 uses the free space later on when you insert or update your data; when no free space is available, DB2

holds your additional data on another page. When several records are physically located out of sequence, performance suffers.

The default for PCTFREE for table spaces is 5 (5% of the page is free). If you have previously used a large PCTFREE to force one row per page, you should instead use MAXROWS 1 on the CREATE or ALTER TABLESPACE statement. MAXROWS has the advantage of maintaining the free space even when new data is inserted.

The default for indexes is 10. The maximum amount of space that is left free in index nonleaf pages is 10%, even if you specify a value higher than 10 for PCTFREE.

To determine the amount of free space currently on a page, run the RUNSTATS utility and examine the PERCACTIVE column of SYSIBM.SYSTABLEPART. See Part 2 of *DB2 Utility Guide and Reference* for information about using RUNSTATS.

Determining pages of free space

The FREEPAGE clause specifies how often DB2 leaves a full page of free space when loading data or when reorganizing data or indexes. DB2 uses the free space later on when you insert or update your data. For example, if you specify 10 for FREEPAGE, DB2 leaves every 10th page free.

The maximum value you can specify for FREEPAGE is 255; however, in a segmented table space, the maximum value is 1 less than the number of pages specified for SEGSIZE.

Recommendations for allocating free space

The goal for allocating free space is to maintain the physical clustering of the data and to reduce the need to frequently reorganize table spaces and indexes. However, you do not want to allocate too much disk space.

Use of PCTFREE or FREEPAGE depends on the type of SQL and the distribution of that activity across the table space or index. When deciding whether to allocate free space consider the data and each index separately and assess the insert and update activity on the data and indexes.

When not to use free space: Free space is not necessary if:

- The object is read-only.

  If you do not plan to insert or update data in a table, there is no need to leave free space for either the table or its indexes.
- The object is not read-only, inserts are at the end, and updates that lengthen varying-length columns are few.

  For example, if inserts are in ascending order by key of the clustering index or are caused by LOAD RESUME SHRLEVEL NONE and update activity is only on fixed-length columns with non-compressed data, the free space for both the table and clustering index should be zero. Generally, free space is beneficial for a non-clustering index because inserts are usually random. However, if the non-clustering index contains a column with a timestamp value that causes the inserts into the index to be in sequence, the free space should be zero.

When to use PCTFREE: When free space is needed, using PCTFREE is recommended over using FREEPAGE in most situations.

If update activity on compressed data, which often results in longer rows, is heavy or insert volume is heavy, use a PCTFREE value greater than the default.

When to use FREEPAGE: Use FREEPAGE rather than PCTFREE if MAXROWS is 1 or rows are larger than half a page because you cannot insert a second row on a page.

Additional recommendations:

- For concurrency, use MAXROWS or larger PCTFREE values for small tables and shared table spaces that use page locking. This reduces the number of rows per page, thus reducing the frequency that any given page is accessed.
- For the DB2 catalog table spaces and indexes, use the defaults for PCTFREE. If additional free space is needed, use FREEPAGE.

End of General-use Programming Interface

Making buffer pools large enough for the workload

Make buffer pools as large as you can afford for the following reasons:

- Using larger buffer pools might mean fewer I/O operations and therefore faster access to your data.
- Using larger buffer pools can reduce I/O contention for the most frequently used tables and indexes.
- Using larger buffer pools can speed sorting by reducing I/O contention for work files.

However, many factors affect how you determine the number of buffer pools to have and how big they should be. See "Determining size and number of buffer pools" on page 671 for more information.

Reducing the time needed to perform I/O operations

You can reduce the time needed to perform individual I/O operations in several ways:

- "Distributing data sets efficiently"
- "Creating additional work file table spaces" on page 657
- "Managing space for I/O performance" on page 658

For information on parallel operations, see Chapter 35, "Parallel operations and query performance," on page 987.

For information on I/O scheduling priority, see "z/OS performance options for DB2" on page 711.

Distributing data sets efficiently

Avoid I/O contention and increase throughput through the I/O subsystem by placing frequently used data sets on fast disk devices and by distributing I/O activity. Distributing I/O activity is less important when you use disk devices with parallel access volumes (PAV) support and multiple allegiance support. (For more information, see "Storage servers and advanced features" on page 710)

Putting frequently used data sets on fast devices

Assign the most frequently used data sets to the faster disk devices at your disposal. For partitioned table spaces, you might choose to have some partitions on faster devices than other partitions. Placing frequently used data sets on fast disk

devices also improves performance for nonpartitioned table spaces. You might consider partitioning any nonpartitioned table spaces that have excessive I/O contention at the data set level.

Distributing the I/O

Allocate frequently used data sets or partitions across your available disk volumes so that I/O operations are distributed. Even with RAID devices, in which the data set is spread across the physical disks in an array, it is important that is accessed at the same time on separate logical volumes to reduce the chance of an I/O request being queued in z/OS.

Consider isolating data sets with characteristics that do not complement other data sets. For example, do not put high volume transaction work that uses synchronous reads on the same volume as something of lower importance that uses list prefetch.

Considering the partitioning scheme of partitioned tablespaces: If the partitions of your partitioned table spaces must be of relatively the same size (which can be a great benefit for query parallelism), consider using a ROWID column as all or part of the partitioning key. For partitions that are of such unequal size that performance is negatively affected, alter the limit key values to set new partition boundaries and then reorganize the affected partitions to rebalance the data. Alternatively, you can use the REORG utility with the REBALANCE keyword to set new partition boundaries. REBALANCE causes DB2 to change the limit key values such that the rows in the range of partitions being reorganized will be distributed across those partitions as evenly as possible.

Increasing the number of data sets for an index:

General-use Programming Interface

If you increase the number of data sets that are used for an index and spread those data sets across the available I/O paths, you can reduce the physical contention on the index. Using data-partitioned secondary indexes or making the piece size of a nonpartitioned index smaller increases the number of data sets that are used for the index.

Using data-partitioned secondary indexes (DPSIs): A secondary index on a partitioned table space can be partitioned. When you partition an index, the partitioning scheme is that of the data in the underlying table, and each index partition has its own data set. Although data-partitioned secondary indexes promote partition independence and can provide performance advantages, they do not always improve query performance. Before using a data-partitioned secondary index, understand their advantages and disadvantages as explained in "Data-partitioned secondary index (DPSI)" on page 56.

Decreasing the piece size of nonpartitioned indexes: If I/O contention on a nonpartitioned index has prevented you from running batch update jobs in parallel, use the PIECESIZE option of CREATE or ALTER INDEX to indicate how large DB2 should make the data sets that make up the nonpartitioned index. As the specification of the maximum addressability of a data set, the piece size of a nonpartitioned index limits how much data DB2 puts into a data set before it is broken into multiple pieces (data sets). By making the piece size smaller than the default value, for example, you can end up with many more data sets.

- *Choosing a value for PIECESIZE.* To choose a PIECESIZE value, divide the size of the nonpartitioned index by the number of data sets that you want. For

example, to ensure that you have 5 data sets for the nonpartitioned index, and your nonpartitioned index is 10 MB (and not likely to grow much), specify PIECESIZE 2M. If your nonpartitioned index is likely to grow, choose a larger value.

When choosing a value, remember that the maximum partition size of the table space determines the maximum number of data sets that the index can use. If the underlying table space is defined with a DSSIZE of 4G or greater (or with LARGE), the limit is 254 pieces for table spaces with 254 parts or less and 4096 pieces for table spaces with more than 254 parts; otherwise, the limit is 32 pieces. Nonpartitioned indexes that were created on LARGE table spaces in Version 5 can have only 128 pieces. If an attempt is made to allocate more data sets than the limit, an abend occurs.

Keep your PIECESIZE value in mind when you are choosing values for primary and secondary quantities. Ideally, although PIECESIZE has no effect on primary and secondary space allocation, the value of your primary quantity and the secondary quantities should be evenly divisible into PIECESIZE to avoid wasting space. Because the underlying data sets are always allocated at the size of PRIQTY and extended, when possible, with the size of SECQTY, understand the implications of their values with the PIECESIZE value:

- If PRIQTY is larger than PIECESIZE, a new data set is allocated and used when the file size exceeds PIECESIZE. Thus, part of the allocated primary storage goes unused, and no secondary extents are created.
- If PRIQTY is smaller than PIECESIZE and SECQTY is not zero, secondary extents are created until the total file size equals or exceeds PIECESIZE. After the allocation of a secondary extent causes the total file size to meet or exceed PIECESIZE, a new data set is allocated and used. When the total file size exceeds PIECESIZE, the part of secondary storage that is allocated beyond PIECESIZE goes unused.
- If PRIQTY is smaller than PIECESIZE and SECQTY is zero, an "unavailable resource" message is returned when the data set fills up. No secondary extents are created nor are additional data sets allocated.

- *Identifying suitable nonpartitioned indexes.* If a nonpartitioned index that has a lot of I/O and a high IOS queue time, consider using the Parallel Access Volumes (PAV) feature and the multiple allegiance feature. For more information about these features, see "Storage servers and channel subsystems" on page 709. Also, consider breaking up the index into smaller pieces. Use the statistics trace to identify I/O intensive data sets. IFCID 199 contains information about every data set that averages more than one I/O per second during the statistics interval. IOS queue time that is 2 or 3 times higher than connect time is considered high. The RMF (Resource Measurement Facility) Device Activity report provides IOS time and CONN time.
- *Determining the number of pieces a nonpartitioned index is using.* You can use one of the following techniques to determine the number of pieces that a nonpartitioned index uses:
  - For DB2-managed data sets, use access method services LISTCAT to check the number of data sets that have been created.
  - For user-managed data sets, examine the high-used RBA (HURBA) for each data set.

End of General-use Programming Interface

Creating additional work file table spaces

If your applications require any of the following, allocate additional work file table spaces on separate disk volumes in a work file database (database DSNDB07 in a non data-sharing environment) to help minimize I/O contention:

- Large concurrent sorts or a single large sort
- Created temporary tables
- Some merge, star, and outer joins
- Non-correlated subqueries
- Materialized views
- Materialized nested table expressions
- Triggers with transition variables

For a single query, the recommended number of work file disk volumes to have is one-fifth the maximum number of data partitions, with 5 as a minimum and 50 as a maximum. For concurrently running queries, multiply this value by the number of concurrent queries.

In addition, in a query parallelism environment, the number of work file disk volumes should be at least equal to the maximum number of parallel operations that is seen for queries in the given workload.

Place these volumes on different channel or control unit paths. Monitor the I/O activity for the work file table spaces, because you might need to further separate this work file activity to avoid contention. As the amount of work file activity increases, consider increasing the size of the buffer pool for work files to support concurrent activities more efficiently. The general recommendation for the work file buffer pool is to increase the size to minimize the following buffer pool statistics:

- MERGE PASSES DEGRADED, which should be less than 1% of MERGE PASS REQUESTED
- WORKFILE REQUESTS REJECTED, which should be less than 1% of WORKFILE REQUEST ALL MERGE PASSES
- Synchronous read I/O, which should be less than 1% of pages read by prefetch
- Prefetch quantity of 4 or less, which should be near 8

During the installation or migration process, you allocated table spaces for 4-KB, 8-KB, 16-KB, and 32-KB buffering.

Steps to create an additional work file table space: Use the following steps to create a new work file table space, *xyz*. (If you are using DB2-managed data sets, omit the step to create the data sets.)

1. Define the required data sets using the VSAM DEFINE CLUSTER statement. You might want to use the definitions in the edited, installation job DSNTIJTM as a model. For more information about job DSNTIJTM and the number of work files, see *DB2 Installation Guide*.
2. Create the work file table space by entering the following SQL statement:

```
CREATE TABLESPACE xyz IN DSNDB07
   BUFFERPOOL BP7
   CLOSE NO
   USING VCAT DSNC810;
```

Managing space for I/O performance

Managing space effectively, by speeding up preformatting and avoiding excessive extents, can improve I/O performance.

Formatting early and speed up formatting

This section describes a general way to speed up preformatting of data by allocating in cylinders, and a more specific way you can preformat a table space before inserting data.

Allocate space in cylinders or in large primary and secondary quantities: Specify your space allocation amounts to ensure allocation by CYLINDER. If you use record allocation for more than a cylinder, cylinder allocation is used. Cylinder allocation can reduce the time required to do SQL mass inserts and to perform LOGONLY recovery; it does not affect the time required to recover a table space from an image copy or to run the REBUILD utility.

When inserting records, DB2 preformats space within a page set as needed. The allocation amount, which is either CYLINDER or TRACK, determines the amount of space that is preformatted at any one time. See "Formatting early and speed up formatting" for a way you can preformat data using LOAD or REORG.

Because less space is preformatted at one time for the TRACK allocation amount, a mass insert can take longer when the allocation amount is TRACK than the same insert when the allocation amount is CYLINDER. However, smart secondary space allocation minimizes the difference between TRACK and CYLINDER. Refer to "Secondary space allocation" on page 32 for information about smart secondary space allocation.

The allocation amount is dependent on device type and the number of bytes you specify for PRIQTY and SECQTY when you define table spaces and indexes. The default SECQTY is 10% of the PRIQTY, or 3 times the page size, whichever is larger. This default quantity is an efficient use of storage allocation. Choosing a SECQTY value that is too small in relation to the PRIQTY value results in track allocation.

For more information about how space allocation amounts are determined, see the description of the DEFINE CLUSTER command in *DFSMS/MVS: Access Method Services for the Integrated Catalog*.

Preformatting during LOAD or REORG: When DB2's preformatting delays impact the performance or execution time consistency of applications that do heavy insert processing, and if the table size can be predicted for a business processing cycle, consider using the PREFORMAT option of LOAD and REORG. If you preformat during LOAD or REORG, DB2 does not have to preformat new pages during execution. When the preformatted space is used and when DB2 has to extend the table space, normal data set extending and preformatting occurs.

Consider preformatting only if preformatting is causing a measurable delay with the insert processing or causing inconsistent elapsed times for insert applications. For more information about the PREFORMAT option, see Part 2 of *DB2 Utility Guide and Reference*.

Recommendation: Quantify the results of preformatting in your environment by assessing the performance both before and after using preformatting.

Avoiding excessive extents

Data set extent size affects performance because excessively small extents can degrade performance during a sequential database scan.

Example: Suppose that the sequential data transfer speed is 100 MB per second and that the extent size is 10 MB. The sequential scan must move to a new extent every 100 milliseconds.

Recommendation: You should maintain extent sizes that are large enough to avoid excessively frequent extent moving during scans.

A linear data set is limited to 123 extents on a volume and 251 total extents on all volumes. If a data set grows and extents are not monitored, jobs will eventually fail due to these extent limitations.

Recommendation: Monitor the number of extents to avoid reaching the maximum number of extents on a volume and the maximum number of extents on all volumes.

Specifying sufficient primary and secondary allocations for frequently used data sets minimizes I/O time, because the data is not located at different places on the disks. Listing the catalog or VTOC occasionally to determine the number of secondary allocations that have been made for your more frequently used data sets can also be helpful. Alternatively, you can use IFCID 0258 in the statistics class 3 trace to monitor data set extensions. OMEGAMON monitors IFCID 0258. You can define a threshold for the number of extents, and receive an alert when a data set exceeds that number of extents.

If you discover that the data sets backing frequently used table spaces or indexes have an excessive number of extents, and if the data sets are user-defined, you can use access method services to reallocate the affected data sets using a larger primary allocation quantity. If the data sets were created using STOGROUPs, you can use the procedure for modifying the definition of table spaces presented in "Altering table spaces" on page 69.

Specify primary quantity for nonpartitioned indexes: To prevent wasted space for nonpartitioned indexes, do one of the following things:

- Let DB2 use the default primary quantity and calculate the secondary quantities. Do this by specifying 0 for the IXQTY subsystem parameter, and by omitting a PRIQTY and SECQTY value in the CREATE INDEX statement or ALTER INDEX statement. If a primary and secondary quantity were previously specified for an index, you can specify PRIQTY -1 and SECQTY -1 to change to the default primary quantity and calculated secondary quantity.
- If the MGEXTSZ subsystem parameter is set to NO, so that you control secondary space allocations, make sure that the value of PRIQTY + (N × SECQTY) is a value that evenly divides into PIECESIZE. For more information about PIECESIZE, see Chapter 5 of *DB2 SQL Reference*.

Reducing processor resource consumption

Many factors affect the amount of processor resources that DB2 consumes. This section describes some ways to reduce DB2 consumption of these resources. It contains the following topics:

- "Reusing threads for your high-volume transactions" on page 660
- "Minimizing the use of DB2 traces" on page 660

- "Using fixed-length records" on page 661

Other ways to reduce DB2 consumption of processor resources include:

- Consider caching authorizations for plans, packages, and routines (user-defined functions and stored procedures). See "Caching authorization IDs for best performance" on page 152 for more information.
- Use an isolation level of cursor stability and CURRENTDATA(NO) to allow lock avoidance. See "The ISOLATION option" on page 846 for more information.
- Avoid using excess granularity for locking, such as row-level locking. See "LOCKSIZE clause of CREATE and ALTER TABLESPACE" on page 838.
- Reorganize indexes and table spaces. Reorganizing can improve the performance of access paths. Reorganizing indexes and table spaces is also important after table definition changes, such as changing the data type of a column, so that the data is converted to its new definition. Otherwise, DB2 must track the data and apply the changes as the data is accessed. See "When to reorganize indexes and table spaces" on page 920.
- Ensure that your access paths are effective. Have only the indexes that you need. Update statistics and rebind as necessary. See "Gathering monitor statistics and update statistics" on page 912 and "Whether to rebind after gathering statistics" on page 923.

Reusing threads for your high-volume transactions

For high-volume transactions, reusing threads can help performance significantly.

- For IMS, process multiple input messages in one scheduling of the IMS processing program by setting PROCLIM to a value greater than 1 and using class priority scheduling. This shares the cost of thread creation and termination among more than one transaction. Alternatively, you can reuse threads with wait for input (WFI), or the IMS fast path and class scheduling. See Part 2 of *DB2 Installation Guide* for more information.
- For CICS, you can enhance thread reuse through specifications for pool and entry threads in the CICS resource definition online (RDO).
- If you are using the Resource Recovery Services attachment facility, see the RRSAF chapter of Part 6 of *DB2 Application Programming and SQL Guide* for more information about reusing threads.

Minimizing the use of DB2 traces

Using the DB2 trace facility, particularly performance and global trace, can consume a large amount of processing resources. Suppressing these trace options significantly reduces additional processing costs.

Global trace

Global trace requires 2% to 100% additional processor utilization. If conditions permit at your site, the DB2 global trace should be turned off. You can do this by specifying NO for the field TRACE AUTO START on panel DSNTIPN at installation. Then, if the global trace is needed for serviceability, you can start it using the START TRACE command.

Accounting and statistics traces

Enabling accounting class 2 along with accounting classes 1 and 3 provides additional detail relating directly to the accounting record IFCID 0003, as well as recording thread level entry into and exit from DB2. This allows you to separate DB2 times from application times. Running accounting class 2 does add to the cost of processing. How much overhead occurs depends on how much SQL the

application issues. Typically, an online transaction incurs an additional 2.5% when running with accounting class 2. A typical batch query application, which accesses DB2 more often, incurs about 10% overhead when running with accounting class 2. If most of your work is through CICS, you most likely do not need to run with class 2, because the class 1 and class 2 times are very close.

Exception: If you are using CICS Transaction Server for z/OS 2.2 with the Open Transaction Environment (OTE), activate and run class 2.

If you have very light DB2 usage and you are using Measured Usage, then you need the SMF 89 records. In other situations, be sure that SMF 89 records are not recorded to avoid this overhead.

Audit trace

The performance impact of auditing is directly dependent on the amount of audit data produced. When the audit trace is active, the more tables that are audited and the more transactions that access them, the greater the performance impact. The overhead of audit trace is typically less than 5%.

When estimating the performance impact of the audit trace, consider the frequency of certain events. For example, security violations are not as frequent as table accesses. The frequency of utility runs is likely to be measured in executions per day. Alternatively, authorization changes can be numerous in a transaction environment.

Performance trace

The combined overhead of all performance classes runs from about 20% to 100%. The overhead for performance trace classes 1 through 3 is typically in the range of 5% to 30%. Therefore, turn on only the performance trace classes required to address a specific performance problem and qualify the trace as much as possible to limit the that is data gathered to only the data that you need. For example, qualify the trace by the plan name and IFCID.

Suppressing other trace options, such as TSO, IRLM, z/OS, IMS, CICS, and other trace options can also reduce overhead.

Using fixed-length records

Use fixed-length columns rather than varying-length columns, particularly in tables that contain many short columns. This can reduce processor use, but is offset by the need for more disk space. If you must use varying-length columns, see Table 109 on page 701 for recommendations about where to place those columns for the best performance and to reduce logging.

If you use ALTER to add a fixed-length column to a table, that column is treated as variable-length until the table has been reorganized.

Response time reporting

To correctly monitor response time, you must understand how it is reported. Response time can be measured in several different ways. Figure 66 on page 663 shows how some of the main measures relate to the flow of a transaction.

Figure 66 on page 663 shows the following times:

End user response time

This is the time from the moment the end user presses the enter key until he or she receives the first response back at the terminal.

DB2 accounting elapsed times

These times are collected in the records from the accounting trace and can be found in the OMEGAMON accounting reports. They are taken over the accounting interval between the point where DB2 starts to execute the first SQL statement, and the point preceding thread termination or reuse by a different user (sign-on).

This interval excludes the time spent creating a thread, and it includes a portion of the time spent terminating a thread.

For parallelism, there are special considerations for doing accounting. See "Monitoring parallel operations" on page 997 for more information.

Elapsed times for stored procedures or user-defined functions separate the time spent in the allied address space and the time spent in the stored procedures address space.

There are two elapsed times:

- Class 1 elapsed time

  This time is always presented in the accounting record and shows the duration of the accounting interval. It includes time spent in DB2 as well as time spent in the front end. In the accounting reports, it is referred to as "application time."

- Class 2 elapsed time

  Class 2 elapsed time, produced only if the accounting class 2 is active, counts only the time spent in the DB2 address space during the accounting interval. It represents the sum of the times from any entry into DB2 until the corresponding exit from DB2. It is also referred to as the time spent in DB2. If class 2 is not active for the duration of the thread, the class 2 elapsed time does not reflect the entire DB2 time for the thread, but only the time when the class was active.

DB2 total transit time

In the particular case of an SQL transaction or query, the "total transit time" is the elapsed time from the beginning of create thread, or sign-on of another authorization ID when reusing the thread, until either the end of the thread termination, or the sign-on of another authorization ID.

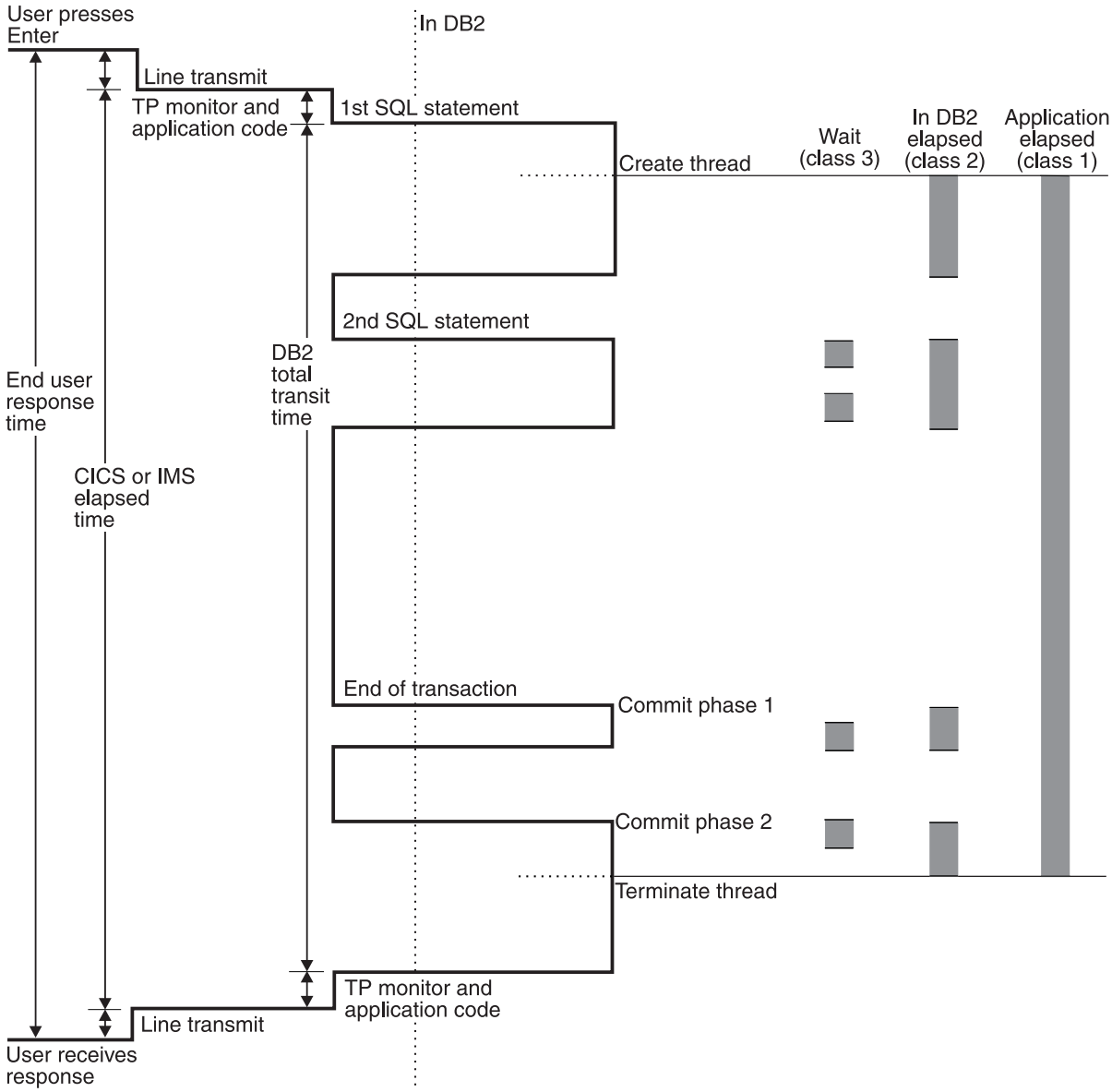


Figure 66. Transaction response times. Class 1 is standard accounting data. Class 2 is elapsed and processor time in DB2. Class 3 is elapsed wait time in DB2. Standard accounting data is provided in IFCID 0003, which is turned on with accounting class 1. When accounting classes 2 and 3 are turned on as well, IFCID 0003 contains additional information about DB2 times and wait times.

Chapter 27. Tuning DB2 buffer, EDM, RID, and sort pools

Proper tuning of your buffer pools, EDM pools, RID pools, and sort pools can improve the response time and throughput for your applications and provide optimum resource utilization. Using data compression can also improve buffer-pool hit ratios and reduce table space I/O rates. For more information on compression, see "Compressing your data" on page 702. This chapter covers the following topics:

- "Tuning database buffer pools"
- "Tuning EDM storage" on page 679
- "Increasing RID pool size" on page 683
- "Controlling sort pool size and sort processing" on page 684

Tuning database buffer pools

Buffer pools are areas of virtual storage that temporarily store pages of table spaces or indexes. When an application program accesses a row of a table, DB2 places the page that contains that row in a buffer. If the requested data is already in a buffer, the application program does not have to wait for it to be retrieved from disk. Avoiding the need to retrieve data from disk results in faster performance.

If the row is changed, the data in the buffer must be written back to disk eventually. But that write operation might be delayed until DB2 takes a checkpoint, or until one of the related write thresholds is reached. (In a data sharing environment, however, the writing mechanism is somewhat different. See Chapter 6 of *DB2 Data Sharing: Planning and Administration* for more information.)

The data remains in the buffer until DB2 decides to use the space for another page. Until that time, the data can be read or changed without a disk I/O operation.

DB2 allows you to use up to 50 buffer pools that contain 4-KB buffers and up to 10 buffer pools each for 8-KB, 16-KB, and 32-KB buffers. You can set the size of each of those buffer pools separately when installing DB2. You can change the sizes and other characteristics of a buffer pool at any time while DB2 is running, by using the ALTER BUFFERPOOL command.

This section includes the following information:

- "Terminology: Types of buffer pool pages" on page 666
- "Read operations" on page 666
- "Write operations" on page 666
- "Assigning a table space or index to a buffer pool" on page 667
- "Buffer pool thresholds" on page 667
- "Determining size and number of buffer pools" on page 671
- "Choosing a page-stealing algorithm" on page 674
- "Long-term page fix option for buffer pools" on page 675
- "Monitoring and tuning buffer pools using online commands" on page 675
- "Using OMEGAMON to monitor buffer pool statistics" on page 677

Buffer Pool Analyzer: You can use the Buffer Pool Analyzer for z/OS to recommend buffer pool allocation changes and to do "what if" analysis of your buffer pools.

© Copyright IBM Corp. 1982, 2008

Terminology: Types of buffer pool pages

At any moment, a database buffer pool can have three types of pages:

In-use pages: These are pages that are currently being read or updated. The data they contain is available for use by other applications.

Updated pages: These are pages whose data has been changed but have not yet been written to disk.

Available pages: These pages can be considered for new use, to be overwritten by an incoming page of new data. Both in-use pages and updated pages are *unavailable* in this sense; they are not considered for new use.

Read operations

DB2 uses three read mechanisms: *normal read*, *sequential prefetch*, and *list sequential prefetch*.

Normal read: Normal read is used when just one or a few consecutive pages are retrieved. The unit of transfer for a normal read is one page.

Sequential prefetch: Sequential prefetch is performed concurrently with other operations of the originating application program. It brings pages into the buffer pool before they are required and reads several pages with a single I/O operation.

Sequential prefetch can be used to read data pages, by table space scans or index scans with clustered data reference. It can also be used to read index pages in an index scan. Sequential prefetch allows CP and I/O operations to be overlapped.

See "Sequential prefetch (PREFETCH=S)" on page 969 for a complete description of sequential prefetch.

List sequential prefetch: List sequential prefetch is used to prefetch data pages that are not contiguous (such as through non-clustered indexes). List prefetch can also be used by incremental image copy. For a complete description of the mechanism, see "List prefetch (PREFETCH=L)" on page 970.

Write operations

Write operations are usually performed concurrently with user requests. Updated pages are queued by data set until they are written when:

- A checkpoint is taken.
- The percentage of updated pages in a buffer pool for a single data set exceeds a preset limit called the vertical deferred write threshold (VDWQT). For more information on this threshold, see "Buffer pool thresholds" on page 667.
- The percentage of unavailable pages in a buffer pool exceeds a preset limit called the deferred write threshold (DWQT). For more information on this threshold, see "Buffer pool thresholds" on page 667.

Table 104 lists how many pages DB2 can write in a single I/O operation.

Table 104. Number of pages that DB2 can write in a single I/O operation

Page size	Number of pages
2 KB	64

Table 104. Number of pages that DB2 can write in a single I/O operation (continued)

Page size	Number of pages
4 KB	32
8 KB	16
16 KB	8
32 KB	4

Assigning a table space or index to a buffer pool

How you assign data to buffer pools can have a significant impact on performance. See "Reasons to choose more than one buffer pool" on page 674 for guidance in choosing a scheme for assigning data and indexes to buffer pools.

Assigning data to default buffer pools

Installation panel DSNTIP1 lets you set one default buffer pool for user data and one for user indexes. Choosing a value other than the default value of BP0 for both of these options is a good idea, because BP0 must be used by the DB2 catalog and directory. BP0 is much more difficult to monitor and tune if user data and indexes also use that buffer pool.

Assigning data to particular buffer pools

You assign a table space or an index to a particular buffer pool by a clause of the following SQL statements: CREATE TABLESPACE, ALTER TABLESPACE, CREATE INDEX, ALTER INDEX. The buffer pool is actually allocated the first time a table space or index assigned to it is opened.

You cannot use the ALTER statement to change the assignment of the catalog and the directory. BP0 is the default buffer pool for sorting, but you can change that by assigning the work file table spaces to another buffer pool. BP0 has a default size of 20000 and a minimum size of 2000. As with any other buffer pool, you can change the size using the ALTER BUFFERPOOL command.

Buffer pool thresholds

The information under this heading, up to "Determining size and number of buffer pools" on page 671, is General-use Programming Interface and Associated Guidance Information, as defined in "Notices" on page 1301.

DB2's use of a buffer pool is governed by several preset values called *thresholds*. Each threshold is a level of use which, when exceeded, causes DB2 to take some action. Certain thresholds might indicate a buffer pool shortage problem, while other thresholds merely report normal buffer management by DB2. The level of use is usually expressed as a percentage of the total size of the buffer pool. For example, the "immediate write threshold" of a buffer pool (described in more detail later) is set at 97.5%. When the percentage of unavailable pages in a buffer pool exceeds that value, DB2 writes pages to disk when updates are completed.

Thresholds for very small buffer pools: This section describes fixed and variable thresholds that are in effect for buffer pools that are sized for the best performance; that is, for buffer pools of 1000 buffers or more. For very small buffer pools, some of the thresholds are lower to prevent "buffer pool full" conditions, but those thresholds are not described.

Fixed thresholds

Some thresholds, like the immediate write threshold, you cannot change. Monitoring buffer pool usage includes noting how often those thresholds are reached. If they are reached too often, the remedy is to increase the size of the buffer pool, which you can do with the ALTER BUFFERPOOL command. Increasing the size, though, can affect other buffer pools, depending on the total amount of real storage available for your buffers.

The fixed thresholds are more critical for performance than the variable thresholds. Generally, you want to set buffer pool sizes large enough to avoid reaching any of these thresholds, except occasionally.

Each of the fixed thresholds is expressed as a percentage of the buffer pool that might be occupied by unavailable pages.

From the highest value to the lowest value, the fixed thresholds are:

- **Immediate write threshold (IWTH): 97.5%**

  This threshold is checked whenever a page is to be updated. If the threshold has been exceeded, the updated page is written to disk as soon as the update completes. The write is synchronous with the SQL request; that is, the request waits until the write is completed. The two operations do not occur concurrently.

  Reaching this threshold has a significant effect on processor usage and I/O resource consumption. For example, updating three rows per page in 10 sequential pages ordinarily requires one or two write operations. However, when IWTH has been exceeded, the updates require 30 synchronous writes.

  Sometimes DB2 uses synchronous writes even when the IWTH has not been exceeded. For example, when more than two checkpoints pass without a page being written, DB2 uses synchronous writes. Situations such as these do not indicate a buffer shortage.

- **Data management threshold (DMTH): 95%**

  This threshold is checked before a page is read or updated. If the threshold is not exceeded, DB2 accesses the page in the buffer pool once for each **page**, no matter how many rows are retrieved or updated in that page. If the threshold is exceeded, DB2 accesses the page in the buffer pool once for each **row** that is retrieved or updated in that page.

  Recommendation: Avoid reaching the DMTH because it has a significant effect on processor usage.

  The DMTH is maintained for each individual buffer pool. When the DMTH is reached in one buffer pool, DB2 does **not** release pages from other buffer pools.

- **Sequential prefetch threshold (SPTH): 90%**

  This threshold is checked at two different times:

  – Before scheduling a prefetch operation. If the threshold has been exceeded, the prefetch is not scheduled.

  – During buffer allocation for an already-scheduled prefetch operation. If the threshold has been exceeded, the prefetch is canceled.

  When the sequential prefetch threshold is reached, sequential prefetch is inhibited until more buffers become available. Operations that use sequential prefetch, such as those using large and frequent scans, are adversely affected.

Thresholds you can change

You can change some thresholds directly by using the ALTER BUFFERPOOL command. Changing a threshold in one buffer pool has no effect on any other buffer pool.

From highest to lowest default value, the variable thresholds are:

- **Sequential steal threshold (VPSEQT)**

  This threshold is a percentage of the buffer pool that might be occupied by sequentially accessed pages. These pages can be in any state: updated, in-use, or available. Hence, any page might or might not count toward exceeding any other buffer pool threshold.

  The default value for this threshold is 80%. You can change that to any value from 0% to 100% by using the VPSEQT option of the ALTER BUFFERPOOL command.

  This threshold is checked before stealing a buffer for a sequentially accessed page instead of accessing the page in the buffer pool. If the threshold has been exceeded, DB2 tries to steal a buffer holding a sequentially accessed page rather than one holding a randomly accessed page.

  Setting the threshold to 0% would prevent any sequential pages from taking up space in the buffer pool. In this case, prefetch is disabled, and any sequentially accessed pages are discarded as soon as they are released.

  Setting the threshold to 100% allows sequential pages to monopolize the entire buffer pool.

- **Virtual buffer pool parallel sequential threshold (VPPSEQT)**

  This threshold is a portion of the buffer pool that might be used to support parallel operations. It is measured as a percentage of the sequential steal threshold (VPSEQT). Setting VPPSEQT to zero disables parallel operation.

  The default value for this threshold is 50% of the sequential steal threshold (VPSEQT). You can change that to any value from 0% to 100% by using the VPPSEQT option on the ALTER BUFFERPOOL command.

- **Virtual buffer pool assisting parallel sequential threshold (VPXPSEQT)**

  This threshold is a portion of the buffer pool that might be used to assist with parallel operations initiated from another DB2 in the data sharing group. It is measured as a percentage of VPPSEQT. Setting VPXPSEQT to zero disallows this DB2 from assisting with Sysplex query parallelism at run time for queries that use this buffer pool. For more information about Sysplex query parallelism, see Chapter 6 of *DB2 Data Sharing: Planning and Administration*.

  The default value for this threshold is 0% of the parallel sequential threshold (VPPSEQT). You can change that to any value from 0% to 100% by using the VPXPSEQT option on the ALTER BUFFERPOOL command.

- **Deferred write threshold (DWQT)**

  This threshold is a percentage of the buffer pool that might be occupied by unavailable pages, including both updated pages and in-use pages.

  The default value for this threshold is 50%. You can change that to any value from 0% to 90% by using the DWQT option on the ALTER BUFFERPOOL command.

  DB2 checks this threshold when an update to a page is completed. If the percentage of unavailable pages in the buffer pool exceeds the threshold, write operations are scheduled for enough data sets (at up to 128 pages per data set) to decrease the number of unavailable buffers to 10% below the threshold. For example, if the threshold is 50%, the number of unavailable buffers is reduced to 40%.

  When the deferred write threshold is reached, the data sets with the oldest updated pages are written asynchronously. DB2 continues writing pages until the ratio goes below the threshold.

- **Vertical deferred write threshold (VDWQT)**

This threshold is similar to the deferred write threshold, but it applies to the number of updated pages for a single page set in the buffer pool. If the percentage or number of updated pages for the data set exceeds the threshold, writes are scheduled for that data set, up to 128 pages.

You can specify this threshold in one of two ways:

– As a *percentage* of the buffer pool that might be occupied by updated pages from a single page set.

  The default value for this threshold is 10%. You can change the percentage to any value from 0% to 90%.

– As the total *number of buffers* in the buffer pool that might be occupied by updated pages from a single page set.

  You can specify the number of buffers from 0 to 9999. If you want to use the number of buffers as your threshold, you must set the percentage threshold to 0.

Changing the threshold: Change the percent or number of buffers by using the VDWQT keyword on the ALTER BUFFERPOOL command.

Because any buffers that count toward VDWQT also count toward DWQT, setting the VDWQT percentage higher than DWQT has no effect: DWQT is reached first, write operations are scheduled, and VDWQT is never reached. Therefore, the ALTER BUFFERPOOL command does not allow you to set the VDWQT percentage to a value greater than DWQT. You can specify a number of buffers for VDWQT than is higher than DWQT, but again, with no effect.

This threshold is overridden by certain DB2 utilities, which use a constant limit of 64 pages rather than a percentage of the buffer pool size. LOAD, REORG, and RECOVER use a constant limit of 128 pages.

Setting VDWQT to 0: If you set VDQWT to zero, DB2 implicitly uses the minimum value of 1% of the buffer pool (a specific number of pages) to avoid synchronous writes to disk. The number of pages is determined by the buffer pool page size, as shown in Table 105:

Table 105. Number of change pages based on buffer pool size

Buffer pool page size	Number of changed pages
4 KB	40
8 KB	24
16 KB	16
32 KB	12

Guidelines for setting buffer pool thresholds

How you set buffer pools depends on your workload and the type and size of data being cached. But always think about the entire system when making buffer pool tuning decisions.

For help in tuning your buffer pools, try the Buffer Pool Analyzer for z/OS.

Pages are frequently re-referenced and updated: Suppose that you have a workload such as a branch table in a bank that contains a few hundred rows and is updated by every transaction. For such a workload, you want a high value for the deferred write and vertical deferred write threshold (90%). The result is that I/O is deferred until DB2 checkpoint and you have a lower I/O rate to disk.

However, if the set of pages updated exceeds the size of the buffer pool, setting both DWQT and VDWQT to 90% might cause the sequential prefetch threshold

(and possibly the data management threshold and the immediate write threshold) to be reached frequently. You might need to set DWQT and VDWQT lower in that case.

Pages are rarely referenced: Suppose that you have a customer table in a bank that has millions of rows that are accessed randomly or are updated sequentially in batch. In this case, lowering the DWQT or VDWQT thresholds (perhaps down to 0) can avoid a surge of write I/Os caused by DB2 checkpoint. Lowering those thresholds causes the write I/Os to be distributed more evenly over time. Secondly, this can improve performance for the storage controller cache by avoiding the problem of flooding the device at DB2 checkpoint.

Query-only buffer pools: For a buffer pool used exclusively for query processing, setting VPSEQT to 100% is reasonable. If parallel query processing is a large part of the workload, set VPPSEQT and, if applicable, VPXPSEQT, to a very high value.

Mixed workloads: For a buffer pool used for both query and transaction processing, the value you set for VPSEQT should depend on the respective priority of the two types of processing. The higher you set VPSEQT, the better queries tend to perform, at the expense of transactions. If you are not sure what value to set for VPSEQT, use the default setting.

Buffer pools containing LOBs: Put LOB data in buffer pools that are not shared with other data. For both LOG YES and LOG NO LOBs, use a deferred write threshold (DWQT) of 0. LOBs specified with LOG NO have their changed pages written at commit time (*force-at-commit* processing). If you set DWQT to 0, those writes happen continuously in the background rather than in a large surge at commit.

LOBs defined with LOG YES can use deferred write, but by setting DWQT to 0, you can avoid massive writes at DB2 checkpoints.

Determining size and number of buffer pools

The size and the number of buffer pools that you use in you DB2 subsystem can significantly affect the performance of that subsystem. This section helps to explain how to choose the best number of buffer pools for your subsystem, and the size of those buffer pools.

Buffer pool sizes

Initially, you set the sizes (in number of pages) of your buffer pools on installation panels DSNTIP1 and DSNTIP2. Because you can modify the sizes of buffer pools using the ALTER BUFFERPOOL command, choosing an exact size initially is not important.

The buffer-pool hit ratio

You can help some of your applications and queries by making the buffer pools large enough to increase the *buffer hit ratio*. Buffer hit ratio is a measure of how often a page access (a getpage) is satisfied without requiring an I/O operation.

Accounting reports, which are application related, show the hit ratio for specific applications. An accounting trace report shows the ratio for single threads. The OMEGAMON buffer pool statistics report shows the hit ratio for the subsystem as a whole. For example, the buffer-pool hit ratio is shown in field **A** in Figure 68 on page 677. The buffer hit ratio uses the following formula to determine how many getpage operations did not require an I/O operation:

```
Hit ratio = (getpages - pages_read_from_disk) / getpages
```

where pages_read_from_disk is the sum of the following fields:

- Number of synchronous reads (field **B** in Figure 68 on page 677)
- Number of pages read via sequential prefetch (field **C**)
- Number of pages read via list prefetch (field **D**)
- Number of pages read via dynamic prefetch (field **E**)

Example: If you have 1000 getpages and 100 pages were read from disk, the equation would be as follows:

```
Hit ratio = (1000-100)/1000
```

The hit ratio in this case is 0.9.

Highest hit ratio: The highest possible value for the hit ratio is 1.0, which is achieved when every page requested is always in the buffer pool. Reading index non-leaf pages tend to have a very high hit ratio since they are frequently re-referenced and thus tend to stay in the buffer pool.

Lowest hit ratio: The lowest hit ratio occurs when the requested page is not in the buffer pool; in this case, the hit ratio is 0 or less. A negative hit ratio means that prefetch has brought pages into the buffer pool that are not subsequently referenced. The pages are not referenced because either the query stops before it reaches the end of the table space or DB2 must take the pages away to make room for newer ones before the query can access them.

A low hit ratio is not always bad: While it might seem desirable to make the buffer hit ratio as close to 1.0 as possible, do not automatically assume a low buffer-pool hit ratio is bad. The hit ratio is a relative value, based on the type of application. For example, an application that browses huge amounts of data using table space scans might very well have a buffer-pool hit ratio of 0. What you want to watch for is those cases where the hit ratio drops significantly for the same application. In those cases, it might be helpful to investigate further.

Hit ratios for additional processes: The hit ratio measurement becomes less meaningful if the buffer pool is being used by additional processes, such as work files or utilities. Some utilities and SQL statements use a special type of getpage request that reserve an empty buffer without requiring that the page be read from disk.

A getpage is issued for each empty work file page without read I/O during sort input processing. The hit ratio can be calculated if the work files are isolated in their own buffer pools. If they are, then the number of getpages used for the hit ratio formula is divided in half as follows:

```
Hit ratio = ((getpages / 2) - pages_read_from_disk) / (getpages / 2)
```

Buffer pool size guidelines

DB2 handles large buffer pools very efficiently. Searching in large buffer pools does not use any more of the processor's resources than searching in smaller pools.

If insufficient real storage exists to back the buffer pool storage, the resulting paging activity may cause performance degradation. If you see significant paging activity, increase the amount of real storage or decrease the size of the buffer pools.

Important: Sufficient real and auxiliary storage must exist to support the combined size of all the buffer pools that are defined. Insufficient storage might cause the system to enter into a wait state and to require an IPL.

Problems with paging: Paging occurs when the virtual storage requirements for a buffer pool exceeds the real storage capacity for the z/OS image. In this case, the least recently used data pages in the buffer pool are migrated to auxiliary storage. Subsequent access to these pages results in a page fault and the page must be brought into real storage from auxiliary storage. Paging of buffer pool storage can negatively affect DB2 performance. The statistics for PAGE-INS REQUIRED FOR WRITE and PAGE-INS REQUIRED FOR READ shown in Figure 68 on page 677 are useful in determining if the buffer pool size setting is too large for available real storage.

Allocating buffer pool storage: DB2 limits the total amount of storage that is allocated for virtual buffer pools to approximately twice the amount of real storage. However, to avoid paging, it is strongly recommended that you set the total buffer pool size to less than the real storage that is available to DB2.

Recommendation: The total buffer pool storage should not exceed the available real storage.

DB2 allocates the minimum buffer pool storage as shown in Table 106.

Table 106. Buffer pool storage allocation

Buffer pool page size	Minimum number of pages allocated
4 KB	2000
8 KB	1000
16 KB	500
32 KB	250

If the amount of virtual storage that is allocated to buffer pools is more than twice the amount of real storage, you cannot increase the buffer pool size.

Advantages of large buffer pools

In general, larger buffer pool sizes can:

- Result in a higher buffer-pool hit ratio, which can reduce the number of I/O operations. Fewer I/O operations can reduce I/O contention, which can provide better response time and reduce the processor resource needed for I/O operations.
- Give an opportunity to achieve higher transaction rates with the same response time. For any given response time, the transaction rate depends greatly on buffer pool size.
- Prevent I/O contention for the most frequently used disks, particularly the catalog tables and frequently referenced user tables and indexes. In addition, a large buffer pool is beneficial when a DB2 sort is used during a query, because I/O contention on the disks containing the work file table spaces is reduced.

Choosing one or many buffer pools

Whether you choose one or many 4-KB buffer pools, you must have at least 250 buffers in BP32K. Some SQL operations, such as joins, can create a result row that does not fit into a 4-KB page. You must also have 8-KB and 16-KB buffer pools.

Buffer Pool Analyzer: You can use the Buffer Pool Analyzer for z/OS to recommend buffer pool allocation changes and to do “what if” analysis of your buffer pools. Specifically, the “what if” analysis can help you compare the performance of a single buffer pool and multiple buffer pools. Additionally, the Buffer Pool Analyzer can recommend how to split buffer pools and where to place table space and index space objects.

Reasons to choose a single buffer pool: If your system has any or all of the following conditions, it is probably best to choose a single 4 KB buffer pool:

- It is already storage constrained.
- You have no one with the application knowledge necessary to do more specialized tuning.
- It is a test system.

Reasons to choose more than one buffer pool: You can benefit from the following advantages if you use more than one buffer pool:

- You can isolate data in separate buffer pools to favor certain applications, data, and indexes. This benefit is twofold:
  - You can favor certain data and indexes by assigning more buffers. For example, you might improve the performance of large buffer pools by putting indexes into separate pools from data.
  - You can customize buffer pool tuning parameters to match the characteristics of the data. For example, you might want to put tables and indexes that are updated frequently into a buffer pool with different characteristics from those that are frequently accessed but infrequently updated.
- You can put work files into a separate buffer pool. This can provide better performance for sort-intensive queries. Applications that use created temporary tables use work files for those tables. Keeping work files separate allows you to monitor temporary table activity more easily.
- This process of segregating different activities and data into separate buffer pools has the advantage of providing good and relatively inexpensive performance diagnosis data from statistics and accounting traces.

Choosing a page-stealing algorithm

When DB2 must take away a page in the buffer pool to make room for a newer page, this action is called “stealing” the page from the buffer pool. DB2 usually uses a least-recently-used (LRU) algorithm for managing pages in storage. That is, it takes away older pages so that more recently used pages can remain in the buffer pool.

However, using the ALTER BUFFERPOOL command, you can also choose to have DB2 use a first-in, first-out (FIFO) algorithm. With this simple algorithm, DB2 does not keep track of how often a page is referenced—the pages that are oldest are moved out, no matter how frequently they are referenced. This simple approach to page stealing results in a small decrease in the cost of doing a getpage operation, and it can reduce internal DB2 latch contention in environments that require very high concurrency.

Recommendations:

- In most cases, keep the default, LRU.
- Use FIFO for buffer pools that have no I/O; that is, the table space or index remains in the buffer pool. Because all the pages are there, the additional cost of a more complicated page management algorithm is not required. FIFO is also beneficial if buffer hit ratio is low, for example, less than 1%.
- Keep objects that can benefit from the FIFO algorithm in different buffer pools from those that benefit from the LRU algorithm. See options for PGSTEAL in ALTER BUFFERPOOL command in *DB2 Command Reference*.

Long-term page fix option for buffer pools

You can use the PGFIX keyword with the ALTER BUFFERPOOL command to fix a buffer pool in real storage for an extended period of time. The PGFIX keyword has the following options:

PGFIX(YES) The buffer pool is fixed in real storage for the long term. Page buffers are fixed when they are first used and remain fixed.

PGFIX(NO) The buffer pool is not fixed in real storage for the long term. Page buffers are fixed and unfixed in real storage, allowing for paging to disk. PGFIX(NO) is the default option.

Recommendation: Use PGFIX(YES) for buffer pools with a high I/O rate, that is, a high number of pages read or written. For buffer pools with zero I/O, such as some read-only data or some indexes with a nearly 100% hit ratio, PGFIX(YES) is not recommended. In these cases, PGFIX(YES) does not provide a performance advantage.

To prevent PGFIX(YES) buffer pools from exceeding the real storage capacity, DB2 uses an 80% threshold when allocating PGFIX(YES) buffer pools. If the threshold is exceeded, DB2 overrides the PGFIX(YES) option with PGFIX(NO).

Monitoring and tuning buffer pools using online commands

The information under this heading, up to “Using OMEGAMON to monitor buffer pool statistics” on page 677, is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page 1301.

The DISPLAY BUFFERPOOL and ALTER BUFFERPOOL commands allow you to monitor and tune buffer pools on line, while DB2 is running, without the overhead of running traces.

You can use the ALTER BUFFERPOOL command to change the following attributes:

- Size
- Thresholds
- Page stealing algorithm

You can use the DISPLAY BUFFERPOOL command to display the current status of one or more active or inactive buffer pools. For example, the following command produces a detailed report of the status of BP0, as shown in Figure 67 on page 676.

```
DISPLAY BUFFERPOOL(BP0) DETAIL
```

```
+DISPLAY BPOOL(BP0) DETAIL
DSNB401I + BUFFERPOOL NAME BP0, BUFFERPOOL ID 0, USE COUNT 47
DSNB402I + BUFFERPOOL SIZE  =  2000 BUFFERS
             ALLOCATED       =     2000   TO BE DELETED   =        0
             IN-USE/UPDATED  =        0
DSNB406I + PAGE STEALING METHOD = LRU
DSNB404I + THRESHOLDS -
             VP SEQUENTIAL         = 80
             DEFERRED WRITE        = 85   VERTICAL DEFERRED WRT = 80,  0
             PARALLEL SEQUENTIAL   = 50   ASSISTING PARALLEL SEQ = 0
DSNB409I + INCREMENTAL STATISTICS SINCE 14:57:55 JAN 22, yyyy
DSNB411I + RANDOM GETPAGE    =   491222 SYNC READ I/O (R) = A  18193
           SEQ.   GETPAGE    =  1378500 SYNC READ I/O (S) = B      0
           DMTH HIT          =        0 PAGE-INS REQUIRED =   460400
DSNB412I + SEQUENTIAL PREFETCH
             REQUESTS D      =    41800   PREFETCH I/O  C =    14473
             PAGES READ E    =   444030
DSNB413I + LIST PREFETCH -
             REQUESTS        =     9046   PREFETCH I/O    =     2263
             PAGES READ      =     3046
DSNB414I + DYNAMIC PREFETCH -
             REQUESTS        =     6680   PREFETCH I/O    =      142
             PAGES READ      =     1333
DSNB415I + PREFETCH DISABLED -
             NO BUFFER       =        0   NO READ ENGINE  =        0
DSNB420I + SYS PAGE UPDATES  = F 220425 SYS PAGES WRITTEN =  G 35169
           ASYNC WRITE I/O   =     5084 SYNC WRITE I/O    =        3
           PAGE-INS REQUIRED =       45
DSNB421I + DWT HIT H        =        2 VERTICAL DWT HIT  =  I     0
           NO WRITE ENGINE   =        0
DSNB440I + PARALLEL ACTIVITY -
           PARALLEL REQUEST  =        0 DEGRADED PARALLEL   =      0
DSNB441I + LPL ACTIVITY -
           PAGES ADDED       =        0
DSN9022I + DSNB1CMD '+DISPLAY BPOOL' NORMAL COMPLETION
```

Figure 67. Sample output from the DISPLAY BUFFERPOOL command.

In Figure 67, find the following fields:

- SYNC READ I/O (R) (**A**) shows the number of random synchronous read I/O operations. SYNC READ I/O (S) (**B**) shows the number of sequential synchronous read I/O operations. Sequential synchronous read I/Os occur when prefetch is disabled.

  To determine the total number of synchronous read I/Os, add SYNC READ I/O (S) and SYNC READ I/O (R).
- In message DSNB412I, REQUESTS (**C**) shows the number of times that sequential prefetch was triggered, and PREFETCH I/O (**D**) shows the number of times that sequential prefetch occurred. PAGES READ (**E**) shows the number of pages read using sequential prefetch.
- SYS PAGE UPDATES (**F**) corresponds to the number of buffer updates.
- SYS PAGES WRITTEN (**G**) is the number of pages written to disk.
- DWT HIT (**H**) is the number of times the deferred write threshold (DWQT) was reached. This number is workload dependent.
- VERTICAL DWT HIT (**I**) is the number of times the vertical deferred write threshold (VDWQT) was reached. This value is per data set, and it is related to the number of asynchronous writes.

Because the number of synchronous read I/Os (**A**) and the number of SYS PAGE UPDATES (**F**) are relatively high, you would want to tune the buffer pools by changing the buffer pool specifications. For example, you could increase the buffer

pool size to reduce the amount of unnecessary I/O, which would make buffer operations more efficient. To do that, enter the following command:

```
-ALTER BUFFERPOOL(BP0) VPSIZE(6000)
```

To obtain buffer pool information on a specific data set, you can use the LSTATS option of the DISPLAY BUFFERPOOL command. For example, you can use the LSTATS option to:

- Provide page count statistics for a certain index. With this information, you could determine whether a query used the index in question, and perhaps drop the index if it was not used.
- Monitor the response times on a particular data set. If you determine that I/O contention is occurring, you could redistribute the data sets across your available disks.

This same information is available with IFCID 0199 (statistics class 8).

For more information on the ALTER BUFFERPOOL or DISPLAY BUFFERPOOL commands, see Chapter 2 of *DB2 Command Reference*.

Using OMEGAMON to monitor buffer pool statistics

You can find information about the database buffer pools in the statistics report produced by OMEGAMON, as Figure 68 shows.

```
TOT4K  READ OPERATIONS       QUANTITY  TOT4K  WRITE OPERATIONS      QUANTITY
---------------------------  --------  ---------------------------  --------
BPOOL HIT RATIO (%) A           73.12  BUFFER UPDATES                 220.4K
                                       PAGES WRITTEN                35169.00
GETPAGE REQUEST               1869.7K  BUFF.UPDATES/PAGES WRITTEN H    6.27
GETPAGE REQUEST-SEQUENTIAL    1378.5K
GETPAGE REQUEST-RANDOM         491.2K  SYNCHRONOUS WRITES I            3.00
                                       ASYNCHRONOUS WRITES           5084.00
SYNCHRONOUS READS B          54187.00
SYNCHRON. READS-SEQUENTIAL   35994.00  PAGES WRITTEN PER WRITE I/O J   5.78
SYNCHRON. READS-RANDOM       18193.00
                                       HORIZ.DEF.WRITE THRESHOLD        2.00
GETPAGE PER SYN.READ-RANDOM     27.00  VERTI.DEF.WRITE THRESHOLD        0.00
                                       DM THRESHOLD K                   0.00
SEQUENTIAL PREFETCH REQUEST  41800.00  WRITE ENGINE NOT AVAILABLE L     0.00
SEQUENTIAL PREFETCH READS    14473.00  PAGE-INS REQUIRED FOR WRITE M   45.00
PAGES READ VIA SEQ.PREFETCH C 444.0K
S.PRF.PAGES READ/S.PRF.READ     30.68

LIST PREFETCH REQUESTS        9046.00
LIST PREFETCH READS           2263.00
PAGES READ VIA LST PREFETCH D 3046.00
L.PRF.PAGES READ/L.PRF.READ      1.35

DYNAMIC PREFETCH REQUESTED    6680.00
DYNAMIC PREFETCH READS         142.00
PAGES READ VIA DYN.PREFETCH E 1333.00
D.PRF.PAGES READ/D.PRF.READ      9.39
PREF.DISABLED-NO BUFFER F        0.00
PREF.DISABLED-NO READ ENG G      0.00

PAGE-INS REQUIRED FOR READ     460.4K
```

Figure 68. OMEGAMON database buffer pool statistics (modified)

The formula for the buffer-pool hit ratio (fields A through E) is explained in "The buffer-pool hit ratio" on page 671

Increase the buffer pool size or reduce the workload if:

- Sequential prefetch is inhibited. PREF.DISABLED-NO BUFFER (**F**) shows how many times sequential prefetch is disabled because the sequential prefetch threshold (90% of the pages in the buffer pool are unavailable) has been reached.
- You detect poor update efficiency. You can determine update efficiency by checking the values in both of the following fields:
  - BUFF.UPDATES/PAGES WRITTEN (**H**)
  - PAGES WRITTEN PER WRITE I/O (**J**)

  In evaluating the values you see in these fields, remember that no values are absolutely acceptable or absolutely unacceptable. Each installation's workload is a special case. To assess the update efficiency of your system, monitor for overall trends rather than for absolute high values for these ratios.

  The following factors impact buffer updates per pages written and pages written per write I/O:
  - Sequential nature of updates
  - Number of rows per page
  - Row update frequency

  For example, a batch program that processes a table in skip sequential mode with a high row update frequency in a dedicated environment can achieve very good update efficiency. In contrast, update efficiency tends to be lower for transaction processing applications, because transaction processing tends to be random.

  The following factors affect the ratio of pages written per write I/O:
  - *Checkpoint frequency.* The CHECKPOINT FREQ field on installation panel DSNTIPN specifies either the number of consecutive log records written between DB2 system checkpoints or the number of minutes between DB2 system checkpoints. You can use a large value to specify CHECKPOINT FREQ in number of log records; the default value is 500 000. You can use a small value to specify CHECKPOINT FREQ in minutes. If you specify CHECKPOINT FREQ in minutes, the recommended setting is 2 to 5 minutes. At checkpoint time, I/Os are scheduled to write all updated pages on the deferred write queue to disk. If system checkpoints occur too frequently, the deferred write queue does not grow large enough to achieve a high ratio of pages written per write I/O.
  - *Frequency of active log switch.* DB2 takes a system checkpoint each time the active log is switched. If the active log data sets are too small, checkpoints occur often, which prevents the deferred write queue from growing large enough to achieve a high ratio of pages written per write I/O. For recommendations on active log data set size, see "Log capacity" on page 697.
  - *Buffer pool size.* The deferred write thresholds (VDWQT and DWQT) are a function of buffer pool size. If the buffer pool size is decreased, these thresholds are reached more frequently, causing I/Os to be scheduled more often to write some of the pages on the deferred write queue to disk. This prevents the deferred write queue from growing large enough to achieve a high ratio of pages written per write I/O.
  - *Number of data sets, and the spread of updated pages across them.* The maximum number of pages written per write I/O is 32, subject to a limiting scope of 180 pages (roughly one cylinder).

    Example: If your application updates page 2 and page 179 in a series of pages, the two changed pages could potentially be written with one write I/O. But if your application updates page 2 and page 185 within a series of pages, writing the two changed pages would require two write I/Os because of the 180-page limit. Updated pages are placed in a deferred write queue

based on the data set. For batch processing it is possible to achieve a high ratio of pages written per write I/O, but for transaction processing the ratio is typically lower.

For LOAD, REORG, and RECOVER, the maximum number of pages written per write I/O is 64, and there is typically no limiting scope. However, in some cases, such as loading a partitioned table space with nonpartitioned indexes, a 180-page limit scope exists.

- SYNCHRONOUS WRITES (**I**) is a high value. This field counts the number of immediate writes. However, immediate writes are not the only type of synchronous write. Therefore, providing a monitoring value for the number of immediate writes can be difficult.

  Ignore SYNCHRONOUS WRITES when DM THRESHOLD is zero.
- DM THRESHOLD (**K**) is reached. This field shows how many times a page was immediately released because the data management threshold was reached. The quantity listed for this field should be zero.

Also note the following fields:

- WRITE ENGINE NOT AVAILABLE (**L**)

  This field records the number of times that asynchronous writes were deferred because DB2 reached its maximum number of concurrent writes. You cannot change this maximum value. This field has a nonzero value occasionally.
- PREF.DISABLED-NO READ ENG (**G**)

  This field records the number of times that a sequential prefetch was not performed because the maximum number of concurrent sequential prefetches was reached. Instead, normal reads were done. You cannot change this maximum value.
- PAGE-INS REQUIRED FOR WRITE (**M**)

  This field records the number of page-ins that are required for a read or write I/O. When the buffer pools are first allocated, the count might be high. After the first allocations are complete, the count should be close to zero.

Tuning EDM storage

EDM storage is composed of these three components, each of which is in a separate storage area:

- The EDM pool, which contains:
  - Skeleton cursor tables (SKCTs)
  - Cursor tables (CTs), or copies of the SKCTs
  - Skeleton package tables (SKPTs)
  - Package tables (PTs), or copies of the SKPTs
  - An authorization cache block for each plan, excluding plans that are bound with CACHESIZE(0)
- The EDM DBD cache, which contains database descriptors (DBDs)
- The EDM statement cache, which contains the skeletons of dynamic SQL if your installation has YES for the CACHE DYNAMIC SQL field of installation panel DSNTIP4

In releases of DB2 prior to Version 8, these components were in a single storage pool. With Version 8, they are in three separate storage pools, and the option to have the dynamic SQL statement cache in a data space is removed.

During the installation process, DSNTINST CLIST calculates the size of the EDM pool and the EDM DBD cache. You can check the calculated sizes on installation panel DSNTIPC. For more information on estimating and specifying the sizes, see *DB2 Installation Guide*.

For data sharing, you might need to increase the EDM DBD cache storage estimate. For more information, see Chapter 2 of *DB2 Data Sharing: Planning and Administration*.

Because of an internal process that changes the size of plans initially bound in one release and then are rebound in a later release, you should carefully monitor the size of the EDM pool, the EDM DBD cache, and the EDM statement cache and increase their sizes, if necessary.

EDM storage space handling

When pages are needed from the EDM pool, the EDM DBD cache, or the EDM statement cache, any pages that are available are allocated first. If the available pages do not provide enough space to satisfy the request, pages are "stolen" from an inactive SKCT, SKPT, DBD, or dynamic SQL skeleton. If enough space is still not available, an SQL error code is sent to the application program.

Table 107 shows how you should design EDM storage:

Table 107. Designing the EDM storage pools

Design...	To contain...
EDM pool	• The CTs and PTs in use • The SKCTs for the most frequently used applications • The SKPTs for the most frequently used applications • The cache blocks for your plans that have caches
EDM DBD cache	• The DBDs in use • The DBDs referred to by the SKCTs and SKPTs for the most frequently used applications
EDM statement cache	• The skeletons of the most frequently used dynamic SQL statements, if your system has enabled the dynamic statement cache

By designing the EDM storage pools this way, you can avoid allocation I/Os, which can represent a significant part of the total number of I/Os for a transaction. You can also reduce the processing time necessary to check whether users attempting to execute a plan are authorized to do so.

EDM storage pools that are too small cause:

- Increased I/O activity in DSNDB01.SCT02, DSNDB01.SPT01, and DSNDB01.DBD01
- Increased response times, due to loading the SKCTs, SKPTs, and DBDs
- Fewer threads used concurrently, due to a lack of storage

Implications for database design

When you design your databases, be aware that a very large number of objects in your database means a larger DBD for that database. And when you drop objects, storage is not automatically reclaimed in that DBD, which can mean that DB2 must

take more locks for the DBD. To reclaim storage in the DBD, use the MODIFY utility, as described in Part 2 of *DB2 Utility Guide and Reference*.

Monitor and manage DBDs to prevent them from becoming too large. Very large DBDs can reduce concurrency and degrade the performance of SQL operations that create or alter objects because of increased I/O and logging. DBDs that are created or altered in DB2 Version 6 or later do not need contiguous storage, but can use pieces of approximately 32 KB. Older DBDs require contiguous storage.

Monitoring and EDM storage

The DB2 statistics record provides information on the EDM pool, the EDM DBD cache, and the EDM statement cache. Figure 69 shows how OMEGAMON presents this information in the statistics report.

```
EDM POOL                      QUANTITY
---------------------------   --------
PAGES IN EDM POOL  A           8192.00
  HELD BY DBDS                   55.00
  HELD BY CTS                     0.00
  HELD BY SKCTS                   6.00
  HELD BY SKPTS                  35.00
  HELD BY PTS                     0.00
  FREE PAGES       B           8151.00
% PAGES IN USE                    0.50
% NON STEAL. PAGES IN USE         0.00

FAILS DUE TO POOL FULL            0.00

DBD REQUESTS                    629.00
DBD NOT IN EDM POOL               0.00
DBD HIT RATIO (%)  C            100.00
CT REQUESTS                       0.00
CT NOT IN EDM POOL                0.00
CT HIT RATIO (%)   D            100.00
CT HIT RATIO (%)                   N/C
PT REQUESTS                     629.00
PT NOT IN EDM POOL                0.00
PT HIT RATIO (%)   E            100.00

PAGES USED IN STMT POOL         184.00
FAILS DUE TO STMT POOL FULL       0.00
PAGES IN STATEMENT POOL         262.1K
FREE PGS IN STMT FREE CHAIN     262.0K


DYNAMIC SQL STMT              QUANTITY
---------------------------   --------
PREPARE REQUESTS   F           8897.00
  FULL PREPARES    G              0.00
  SHORT PREPARES               9083.00
GLOBAL CACHE HIT RATIO (%)  H  100.00

IMPLICIT PREPARES                 0.00
PREPARES AVOIDED                  0.00
CACHE LIMIT EXCEEDED              0.00
PREP STMT PURGED                  0.00
LOCAL CACHE HIT RATIO (%)          N/C
```

Figure 69. EDM storage utilization in the OMEGAMON statistics report

The important values to monitor are:

Efficiency of the EDM pool: You can measure the efficiency of the EDM pool by using the following ratios:

DBD HIT RATIO (%) **C**
CT HIT RATIO (%) **D**
PT HIT RATIO (%) **E**

These ratios for the EDM pool depend upon your location's work load. In most DB2 subsystems, a value of 80% or more is acceptable. This value means that at least 80% of the requests were satisfied without I/O.

The number of free pages is shown in FREE PAGES (**B**) in Figure 69. If this value is more than 20% of PAGES IN EDM STORAGE (**A**) during peak periods, the EDM pool size is probably too large. In this case, you can reduce its size without affecting the efficiency ratios significantly.

EDM statement cache hit ratio: If you have caching turned on for dynamic SQL, the EDM storage statistics have information that can help you determine how successful your applications are at finding statements in the cache. See mapping macro DSNDQISE for descriptions of these fields.

PREPARE REQUESTS (**F**) in Figure 69 records the number of requests to search the cache. FULL PREPARES (**G**) records the number of times that a statement was inserted into the cache, which can be interpreted as the number of times a statement was not found in the cache. To determine how often the dynamic statement was used from the cache, check the value in GLOBAL CACHE HIT RATIO (**H**). The value is calculated with the following formula:

```
(PREPARE REQUESTS - FULL PREPARES) / PREPARE REQUESTS = hit ratio
```

EDM pool space utilization and performance: For smaller EDM pools, space utilization or fragmentation is normally more critical than for larger EDM pools. For larger EDM pools, performance is normally more critical. DB2 emphasizes performance and uses less optimum EDM storage allocation when the EDM pool size exceeds 40 MB. For systems with large EDM pools that are greater than 40 MB to continue to use optimum EDM storage allocation at the cost of performance, you can set the keyword EDMBFIT in the DSNTIJUZ job to YES. The EDMBFIT keyword adjusts the search algorithm on systems with EDM pools that are larger than 40 MB. The default NO tells DB2 to use a first-fit algorithm while YES tells DB2 to use a better-fit algorithm.

Recommendation: Set EDMBFIT to NO in most cases. It is especially important to set EDMBFIT to NO when a high class 24 latch (EDM LRU latch) contention exists. For example, be sure to set EDMBFIT to NO when class 24 latch contention exceeds 500 contentions per second.

Recommendation: Set EDMBFIT to YES when EDMPOOL full conditions occur for an EDM pool size that exceeds 40 MB.

Tips for managing EDM storage

Here are guidelines for helping keep EDM storage, especially the EDM pool, under control.

Use packages

By using multiple packages you can increase the effectiveness of EDM pool storage management by having smaller objects in the pool.

Use RELEASE(COMMIT) when appropriate

Using the bind option RELEASE(COMMIT) for infrequently used packages and plans can cause objects to be removed from the EDM pool sooner.

Be aware of large DBDs

DBDs in new function mode can be larger than DBDs in previous releases. Therefore, the DBDs might contain additional expansion areas. DBDs that were created before Version 8 are expanded when they are read. When a DBD is expanded, DB2 requires storage for two copies while the DBD is loaded.

Understand the impact of using DEGREE(ANY)

A plan or package that is bound with DEGREE(ANY) can require 50 to 70% more storage for the CTs and PTs in the EDM pool than one bound with DEGREE(1). If you change a plan or package to DEGREE(ANY), check the change in the column AVGSIZE in SYSPLAN or SYSPACKAGE to determine the increase required.

Increasing RID pool size

The RID pool is used for all record identifier (RID) processing. It is used for enforcing unique keys while updating multiple rows and for sorting RIDs during the following operations:

List prefetch, including single index list prefetch

Access via multiple indexes

Hybrid joins

To favor the selection and efficient completion of those access paths, increase the maximum RID pool size. However, if there is not enough RID pool storage, it is possible that the statement might revert to a table space scan.

A RID pool size of 0 disables those access paths. If you specify a RID pool size of 0, plans or packages that were previously bound with a non-zero RID pool size might experience significant performance degradation. Rebind any plans or packages that include SQL statements that use RID processing.

The default RID pool size is 8 MB. You can override this value on installation panel DSNTIPC.

To determine if a transaction used the RID pool, see the RID Pool Processing section of the OMEGAMON accounting trace record.

The RID pool, which all concurrent work shares, is limited to a maximum of 10 000 MB. The RID pool is created at system initialization, but no space is allocated until RID storage is needed. It is then allocated in 32-KB blocks as needed, until the maximum size that you specified on installation panel DSNTIPC is reached.

The general formula for computing RID pool size is:

```
Number of concurrent RID processing activities ×
  average number of RIDs × 2 × 5 bytes per RID
```

Example: Three concurrent RID processing activities, with an average of 4000 RIDs each, would require 120 KB of storage, because:

```
3 × 4000 × 2 × 5 = 120KB
```

Whether your SQL statements that use RID processing complete efficiently or not depends on other concurrent work using the RID pool.

Controlling sort pool size and sort processing

Sort is invoked when a cursor is opened for a SELECT statement that requires sorting. The maximum size of the sort work area allocated for each concurrent sort user depends on the value you specified for the SORT POOL SIZE field on installation panel DSNTIPC. The default value is 2 MB.

Estimating the maximum size of the sort pool

You can change the maximum size of the sort pool by using the installation panels in UPDATE mode. A rough formula for determining the maximum size is as follows:

```
32000 × (16 + sort key length + sort data length)
```

For sort key length and sort data length, use values that represent the maximum values for the queries you run. To determine these values, refer to fields QW0096KL (key length) and QW0096DL (data length) in IFCID 0096, as mapped by macro DSNDQW01. You can also determine these values from an SQL activity trace.

If a column is in the ORDER BY clause that is not in the select clause, that column should be included in the sort data length and the sort key length as shown in the following example:

```
SELECT C1, C2, C3
 FROM tablex
 ORDER BY C1, C4;
```

If C1, C2, C3, and C4 are each 10 bytes in length, you could estimate the sort pool size as follows:

```
32000 × (16 + 20 + (10 + 10 + 10 + 10)) = 2342000 bytes
```

Where the values are determined in the following way:

- 32000 = maximum number of sort nodes
- 16 = size (in bytes) of each node
- 20 = sort key length (ORDER BY C1, C4)
- 10+10+10+10 = sort data length (each column is 10 bytes in length)

How sort work files are allocated

The sort begins with the input phase when ordered sets of rows are written to work files. At the end of the input phase, when all the rows have been sorted and inserted into the work files, the work files are merged together, if necessary, into one work file containing the sorted data. The merge phase is skipped if there is only one work file at the end of the input phase. In some cases, intermediate merging might be needed if the maximum number of sort work files has been allocated.

The work files that are used in sort are logical work files, which reside in work file table spaces in your work file database (which is DSNDB07 in a non data-sharing environment). DB2 uses the buffer pool when writing to the logical work file. Only the buffer pool size limits the number of work files that can be used for sorting.

A sort can complete in the buffer pool without I/Os. This is the ideal situation, but it might be unlikely, especially if the amount of data being sorted is large. The sort row size is actually made up of the columns being sorted (the sort key length) and the columns the user selects (the sort data length). Having a very large buffer pool for sort activity can help avoid disk I/Os.

When your application needs to sort data, the work files are allocated on a least recently used basis for a particular sort. For example, if five logical work files (LWFs) are to be used in the sort, and the installation has three work file table spaces (WFTSs) allocated, then:

- LWF 1 would be on WFTS 1.
- LWF 2 would be on WFTS 2.
- LWF 3 would be on WFTS 3.
- LWF 4 would be on WFTS 1.
- LWF 5 would be on WFTS 2.

To support large sorts, DB2 can allocate a single logical work file to several physical work file table spaces.

Improving the performance of sort processing

The following factors affect the performance of DB2 sort processing:

- The larger the sort pool, the more efficient the sort is.
- Minimize I/O contention on the I/O paths to the physical work files. Also, make sure that physical work files are allocated on different I/O paths and packs to minimize I/O contention. Using disk devices with Parallel Access Volumes (PAV) support is another way to significantly minimize I/O contention. (For more information, see "Parallel Access Volumes (PAV)" on page 710.)
- Allocate additional physical work files in excess of the defaults, and put those work files in their own buffer pool.

  Segregating work file activity enables you to better monitor and tune sort performance. It also allows DB2 to handle sorts more efficiently because these buffers are available only for sort without interference from other DB2 work.

  Applications using created temporary tables use work file space until a COMMIT or ROLLBACK occurs. (If a cursor is defined WITH HOLD, then the data is held past the COMMIT.) If sorts are happening concurrently with the temporary table's existence, then you probably need more space to handle the additional use of the work files. Applications that require star join, materialized views, materialized nested table expressions, non-correlated subqueries or triggers also use work files.

  For information about defining additional work file table spaces, refer to "Creating additional work file table spaces" on page 657.
- Applications should only sort those columns that need to be sorted, as these key fields appear twice in the sort row size. The smaller the sort row size, the more rows can fit it.
- Because varying-length columns are padded to their maximum length, do not select VARCHAR columns unless they are required.
- A buffer pool sequential steal threshold (VPSEQT) of 100% avoids wasting buffers, unless a sparse index is used to access the work files. The default value of 80% allows 20% of the buffers to go unused. A value of 99% prevents space map pages, which are randomly accessed, from being overwritten by massive prefetch. If sparse index access is used, the default of 80% is recommended.

Other factors that influence sort performance include the following:

- The better sorted the data is, the more efficient the sort is.
- If the buffer pool deferred write threshold (DWQT) or data set deferred write threshold (VDWQT) are reached, writes are scheduled. For a large sort using many logical work files, this is difficult to avoid, even if a very large buffer pool is specified.

- If I/Os occur in the sorting process, in the merge phase DB2 uses sequential prefetch to bring pages into the buffer pool with a prefetch quantity of eight pages. However, if the buffer pool is constrained, then DB2 uses a prefetch quantity of four pages or less, or disables prefetch entirely because of the unavailability of enough pages.

For any SQL statement that initiates sort activity, the OMEGAMON SQL activity reports provide information on the efficiency of the sort that is involved.

Chapter 28. Improving resource utilization

When system resources are shared among transactions, end user queries, Web requests, distributed application requests, and batch programs, it is important to control how those resources are used. You need to separate data and set priorities carefully. You might choose to emphasize resource use, performance, concurrency, or data security.

Choose the controls that best match your goals. You may, for example, want to minimize resource usage, maximize throughput or response time, ensure a certain level of service to some users, or avoid conflicts between users. Your goal might be to favor a certain class of users or to achieve the best overall system performance.

Many of the things you currently do for a single DB2 to improve response time or reduce processor consumption also hold true in the data sharing environment. Thus, most of the information in this chapter holds true for data sharing as well.

This chapter covers the following topics:

- "Managing the opening and closing of data sets"
- "Planning the placement of DB2 data sets" on page 692
- "DB2 logging" on page 694
- "Improving disk utilization: space and device utilization" on page 701
- "Improving real storage utilization" on page 706
- "Performance and storage" on page 708
- "z/OS performance options for DB2" on page 711
- "Controlling resource usage" on page 714
- "Resource limit facility (governor)" on page 716

For more information about tuning in a data sharing environment, see Chapter 6 of *DB2 Data Sharing: Planning and Administration*. For more information about reducing resource usage in distributed applications, see "Tuning distributed applications" on page 1004. For more information about setting thread limits, see Chapter 29, "Managing DB2 threads," on page 729

Managing the opening and closing of data sets

Having the needed data sets open and available for use is important for the performance of transactions. However, the number of open data sets affects the amount of available storage, and number of open data sets in read-write state affects restart time. This section describes how DB2 manages this open and close activity and gives some recommendations about how you can influence this processing.

This section includes the following topics:

- "Determining the maximum number of open data sets"
- "Understanding the CLOSE YES and CLOSE NO options" on page 690
- "Switching to read-only for infrequently updated and infrequently accessed page sets" on page 691

Determining the maximum number of open data sets

DB2 defers closing and deallocating the table spaces or indexes until the number of open data sets reaches 99% of the value that you specified for DSMAX.

© Copyright IBM Corp. 1982, 2008

When DSMAX is reached, DB2 closes 300 data sets or 3% of the value of DSMAX, whichever number of data sets is fewer. Thus, DSMAX controls not only the limit of open data sets, but in some cases also controls the number of data sets that are closed when that limit is reached.

How DB2 determines DSMAX

Initially, DB2 calculates DSMAX as follows:

- Let *concdb* be the number of concurrent databases specified on installation panel DSNTIPE.
- Let *tables* be the number of tables per database specified on installation panel DSNTIPD.
- Let *indexes* be the number of indexes per table. The installation CLIST sets this variable to 2.
- Let *tblspaces* be the number of table spaces per database specified on installation panel DSNTIPD.

DB2 calculates the number of open data sets with the following formula:

$$concdb \times \{(tables \times indexes) + tblspaces\}$$

Modifying DSMAX

The formula used by DB2 does not take partitioned or LOB table spaces into account. Those table spaces can have many data sets. If you have many partitioned table spaces or LOB table spaces, you might need to increase DSMAX. Don't forget to consider the data sets for nonpartitioned indexes defined on partitioned table spaces. If those indexes are defined with a small PIECESIZE, there could be many data sets. You can modify DSMAX by updating field DSMAX - MAXIMUM OPEN DATA SETS on installation panel DSNTIPC.

Calculating the size of DSMAX: DSMAX should be larger than the maximum number of data sets that are open and in use at one time. For the most accurate count of open data sets, refer to the OPEN/CLOSE ACTIVITY section of the OMEGAMON statistics report. Make sure the statistics trace was run at a peak period, so that you can obtain the most accurate maximum figure.

The best indicator of when to increase DSMAX is when the open and close activity of data sets is high, 1 per second as a general guideline. Refer to the OPEN/CLOSE value under the SER.TASK SWITCH section of the OMEGAMON accounting report. Consider increasing DSMAX when this value shows more than 1 event per second.

To calculate the total number of data sets (rather than the number that are open during peak periods), you can do the following:

1. To find the number of simple and segmented table spaces, use the following query. The calculation assumes that you have one data set for each simple, segmented, and LOB table space.

   These catalog queries are included in DSNTESP in SDSNSAMP. You can use them as input to SPUFI.

   General-use Programming Interface

   Query 1

```
SELECT CLOSERULE, COUNT(*)
  FROM SYSIBM.SYSTABLESPACE
  WHERE PARTITIONS = 0
  GROUP BY CLOSERULE;
```

End of General-use Programming Interface

2. To find the number of data sets for the partitioned table spaces, use the following query, which returns the number of partitioned table spaces and the total number of partitions. Partitioned table spaces can require up to 4096 data sets for the data, and a corresponding number of data sets for each partitioned index.

General-use Programming Interface

Query 2

```
SELECT CLOSERULE, COUNT(*), SUM(PARTITIONS)
  FROM SYSIBM.SYSTABLESPACE
  WHERE PARTITIONS > 0
  GROUP BY CLOSERULE;
```

End of General-use Programming Interface

3. To find the number of data sets required for each nonpartitioned index, use the following query. The calculation assumes that you have only one data set for each nonpartitioned index. If you use pieces, adjust accordingly.

General-use Programming Interface

Query 3

```
SELECT CLOSERULE, COUNT(*)
  FROM   SYSIBM.SYSINDEXES T1, SYSIBM.SYSINDEXPART T2
  WHERE  T1.NAME = T2.IXNAME
  AND    T1.CREATOR = T2.IXCREATOR
  AND    T2.PARTITION = 0
  GROUP BY CLOSERULE;
```

End of General-use Programming Interface

4. To find the number of data sets for the partitioned indexes, use the following query, which returns the number of index partitions. You have one data set for each index partition.

General-use Programming Interface

Query 4

```
SELECT CLOSERULE, COUNT(*)
  FROM   SYSIBM.SYSINDEXES T1, SYSIBM.SYSINDEXPART T2
  WHERE  T1.NAME = T2.IXNAME
  AND    T1.CREATOR = T2.IXCREATOR
  AND    T2.PARTITION > 0
  GROUP BY CLOSERULE;
```

End of General-use Programming Interface

5. To find the total number of data sets, add the numbers that result from the four queries. (For Query 2, use the sum of the partitions that was obtained.)

Recommendations

As with many recommendations in DB2, you must weigh the cost of performance versus availability when choosing a value for DSMAX. Consider the following factors:

- For best performance, you should leave enough margin in your specification of DSMAX so that frequently used data sets can remain open after they are no

longer referenced. If data sets are opened and closed frequently, such as every few seconds, you can improve performance by increasing DSMAX.

- The number of open data sets on your subsystem that are in read/write state affects checkpoint costs and log volumes. To control how long data sets stay open in a read/write state, specify values for the RO SWITCH CHKPTS and RO SWITCH TIME fields of installation panel DSNTIPN. See "Switching to read-only for infrequently updated and infrequently accessed page sets" on page 691 for more information.
- Consider segmented table spaces to reduce the number of data sets.

  To reduce open and close activity, you can try reducing the number of data sets by combining tables into segmented table spaces. This approach is most useful for development or end-user systems where there are a lot of smaller tables that can be combined into single table spaces.

Understanding the CLOSE YES and CLOSE NO options

General-use Programming Interface

This section describes how DB2 manages data set closing and how the CLOSE value for a table space or index affects the process of closing an object's data sets. The term **page set** refers to a table space or index.

The process of closing

DB2 dynamically manages page sets using two levels of page set closure—logical close and physical close.

Logical close: This occurs when the application has been deallocated from that page set. This is at either commit or deallocation time, depending on the RELEASE(COMMIT/DEALLOCATE) option of the BIND command, and is driven by the use count. When a page set is logically closed, the page set use count is decremented. When the page set use count is zero, the page set is considered not in use; this makes it a candidate for physical close.

Physical close: This happens when DB2 closes and deallocates the data sets for the page set.

When the data sets are closed

As described in "Determining the maximum number of open data sets" on page 687, the number of open data sets determines when it is necessary to close data sets. When DB2 closes data sets, all data sets for a particular table space, index, or partition are closed.

The value you specify for CLOSE determines the *order* in which page sets that are not in use are closed. When the open data set count becomes greater than 99% of DSMAX, DB2 first closes page sets defined with CLOSE YES. The least recently used page sets are closed first.

If the number of open data sets cannot be limited by closing page sets or partitions defined with CLOSE YES, DB2 then must close page sets or partitions defined with CLOSE NO. The least recently used CLOSE NO data sets are closed first.

Delaying the physical closure of page sets or partitions until necessary is called *deferred close*. Deferred closing of a page set or partition that is no longer being used means that another application or user can access the table space and employ

the accompanying indexes without DB2 reopening the data sets. Thus, deferred closing of page sets or partitions can improve your applications' performance by avoiding I/O processing.

Recommendation: For a table space whose data is continually referenced, in most cases it does not matter whether it is defined with CLOSE YES or CLOSE NO; the data sets remain open. This is also true, but less so, for a table space whose data is not referenced for short periods of time; because DB2 uses deferred close to manage data sets, the data sets are likely to be open when they are used again.

You could find CLOSE NO appropriate for page sets that contain data you do not frequently use but is so performance-critical that you cannot afford the delay of opening the data sets.

If the number of open data sets is a concern, choose CLOSE YES for page sets with many partitions or data sets.

End of General-use Programming Interface

Switching to read-only for infrequently updated and infrequently accessed page sets

For both CLOSE YES and CLOSE NO page sets, DB2 automatically converts infrequently updated page sets or partitions from read-write to read-only state according to the values you specify in the RO SWITCH CHKPTS and RO SWITCH TIME fields of installation panel DSNTIPL.

With data sharing, DB2 uses the CLOSE YES or CLOSE NO attribute of the table
space or index space to determine whether to physically close infrequently
accessed page sets that have had global buffer pool dependencies. Infrequently
accessed CLOSE YES page sets are physically closed. Infrequently accessed CLOSE
NO page sets remain open.

RO SWITCH CHKPTS is the number of consecutive DB2 checkpoints since a page set or partition was last updated; the default is 5. RO SWITCH TIME is the amount of elapsed time since a page set or partition was last updated; the default is 10 minutes. If either condition is met, the page set or partition is converted from read-write to read-only state.

Updating SYSLGRNX: For both CLOSE YES and CLOSE NO page sets, SYSLGRNX entries are updated when the page set is converted from read-write state to read-only state. When this conversion occurs for table spaces, the SYSLGRNX entry is closed and any updated pages are externalized to disk. For indexes defined as COPY NO, there is no SYSLGRNX entry, but the updated pages are externalized to disk.

Performance benefits of read-only switching: An infrequently used page set's conversion from read-write to read-only state results in the following performance benefits:

- Improved data recovery performance because SYSLGRNX entries are more precise, closer to the last update transaction commit point. As a result, the RECOVER utility has fewer log records to process.
- Minimized logging activities. Log records for page set open, checkpoint, and close operations are only written for updated page sets or partitions. Log records are not written for read-only page sets or partitions.

Recommendations for RO SWITCH TIME and RO SWITCH CHKPTS: In most cases, the default values are adequate. However, if you find that the amount of R/O switching is causing a performance problem for the updates to SYSLGRNX, consider increasing the value of RO SWITCH TIME, perhaps to 30 minutes.

Planning the placement of DB2 data sets

To improve performance, plan the placement of DB2 data sets carefully. Concentrate mainly on data sets for system files (especially the active logs), for the DB2 catalog and directory, and for user data and indexes. The objective is to balance I/O activity between different volumes, control units, and channels, which minimizes the I/O elapsed time and I/O queuing.

This section includes the following topics:

- "Estimating concurrent I/O requests"
- "Crucial DB2 data sets"
- "Changing catalog and directory size and location" on page 693
- "Monitoring I/O activity of data sets" on page 693
- "Work file data sets" on page 693

Estimating concurrent I/O requests

DB2 has a multi-tasking structure in which each user's request runs under a different task control block (TCB). In addition, the DB2 system itself has its own TCBs and SRBs for logging and database writes. When your DB2 system is loaded with data, you can estimate the maximum number of concurrent I/O requests as:

MAX USERS + 600 sequential prefetches + 600 asynchronous writes

This figure is important when you calculate the number of data paths for your DB2 subsystem.

Crucial DB2 data sets

First, decide which data sets are crucial to DB2's functioning. To gather this information, use the I/O reports from the DB2 performance trace. If these reports are not available, consider these the most important data sets:

For transactions:
- DSNDB01.SCT02 and its index
- DSNDB01.SPT01 and its index
- DSNDB01.DBD01
- DSNDB06.SYSPLAN table space and indexes on SYSPLANAUTH table
- DSNDB06.SYSPKAGE
- Active and archive logs
- Most frequently used user table spaces and indexes

For queries:
- DSNDB01.DBD01
- DSNDB06.SYSPLAN table space and indexes on SYSPLANAUTH
- DSNDB06.SYSPKAGE
- DSNDB06.SYSDBASE table space and its indexes
- DSNDB06.SYSVIEWS table space and the index on SYSVTREE
- Work file table spaces
- QMF system table data sets
- Most frequently used user table spaces and indexes

These lists do not include other data sets that are less crucial to DB2's performance, such as those that contain program libraries, control blocks, and formats. Those types of data sets have their own design recommendations.

Changing catalog and directory size and location

Consider changing the size or location of your DB2 catalog or directory if necessary for your site. See Appendix F, "Using tools to monitor performance," on page 1185 for guidelines on when to do this.

To change the size or location of DB2 catalog or directory data sets, you must run the RECOVER utility on the appropriate database, or the REORG utility on the appropriate table space. A hierarchy of recovery dependencies determines the order in which you should try to recover data sets. This order is discussed in the description of the RECOVER utility in Part 2 of *DB2 Utility Guide and Reference*.

Monitoring I/O activity of data sets

The best way to monitor your I/O activity against database data sets is through IFCID 0199 (statistics class 8). This IFCID can give you information such as the average write I/O delay or the maximum delay for a particular data set over the last statistics reporting interval. The same information is also reported in the DISPLAY BUFFERPOOL command with the LSTATS option. Furthermore, OMEGAMON reports IFCID 0199 in batch reports and in online monitoring.

More detailed information is available, with more overhead, with the I/O trace (performance class 4). If you want information about I/O activity to the log and BSDS data sets, use performance class 5.

You can also use RMF to monitor data set activity. SMF record type 42-6 provides activity information and response information for a data set over a time interval. These time intervals include the components of I/O time, such as IOS queue time. Using RMF incurs about the same overhead as statistics class 8.

For information on how to tune your environment to improve I/O performance, see "Reducing the time needed to perform I/O operations" on page 654 and "Synchronous I/O suspension time" on page 643.

Work file data sets

Work file data sets are used for sorting, for materializing views and nested table expressions, for temporary tables, and for other activities. DB2 does not distinguish or place priorities on these uses of the work file data sets. Excessive activity from one type of use can interfere and detract from the performance of others. It is important to monitor how work files use devices, both in terms of space use and I/O response times.

More about temporary work file result tables: Part 2 of *DB2 Installation Guide* contains information about how to estimate the disk storage required for temporary work file result tables. The storage is similar to that required for regular tables. When a temporary work file result table is populated using an INSERT statement, it uses work file space.

No other process can use the same work file space as that temporary work file table until the table goes away. The space is reclaimed when the application process commits or rolls back, or when it is deallocated, depending which RELEASE option was used when the plan or package was bound.

To best monitor these result tables, keep work files in a separate buffer pool. Use IFCID 0311 in performance trace class 8 to distinguish these tables from other uses of the work file.

DB2 logging

DB2 logs changes made to data, and other significant events, as they occur. You can find background information on the DB2 log in Chapter 18, “Managing the log and the bootstrap data set,” on page 423. When you focus on logging performance issues, remember that the characteristics of your workload have a direct effect on log write performance. Long-running tasks that commit infrequently have a lot more data to write at commit than a typical transaction. These tasks can cause subsystem impact because of the excess storage consumption, locking contention, and resources that are consumed for a rollback.

Don’t forget to consider the cost of reading the log as well. The cost of reading the log directly affects how long a restart or a recovery occurs because DB2 must read the log data before applying the log records back to the table space.

This section includes the following topics:
- “Logging performance issues and recommendations”
- “Log capacity” on page 697
- “Controlling the amount of log data” on page 699

Logging performance issues and recommendations

This section provides background information on logging operations. By understanding the day-to-day activity on the log, you can more effectively pinpoint when problems occur and better understand how to tune for best performance.

Log writes

Log writes are divided into two categories: synchronous and asynchronous.

Asynchronous writes: Asynchronous writes are the most common. These asynchronous writes occur when data is updated. Before- and after-image records are usually moved to the log output buffer, and control is returned to the application. However, if no log buffer is available, the application must wait for one to become available.

Synchronous writes: Synchronous writes usually occur at commit time when an application has updated data. This write is called 'forcing' the log because the application must wait for DB2 to force the log buffers to disk before control is returned to the application. If the log data set is not busy, all log buffers are written to disk. If the log data set is busy, the requests are queued until it is freed.

Writing to two logs: Dual logging is shown in Figure 70 on page 695.

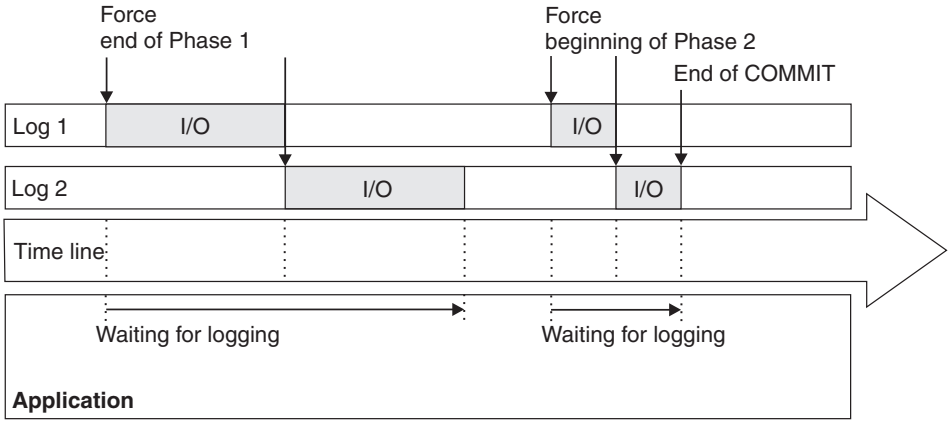


Figure 70. Dual logging during two-phase commit

If there are two logs (recommended for availability), the write to the first log, in general, must complete before the write to the second log begins. The first time a log control interval is written to disk, the write I/Os to the log data sets are performed in parallel. However, if the same 4 KB log control interval is again written to disk, then the write I/Os to the log data sets must be done serially to prevent any possibility of losing log data in case of I/O errors on both copies simultaneously.

Two-phase commit log writes: Because they use two-phase commit, applications that use the CICS, IMS, and RRS attachment facilities force writes to the log twice, as shown in Figure 70. The first write forces all the log records of changes to be written (if they have not been written previously because of the write threshold being reached). The second write writes a log record that takes the unit of recovery into an in-commit state.

Recommendations: To improve log write performance:

- **Choose a large size for OUTPUT BUFFER size:** The OUTPUT BUFFER field of installation panel DSNTIPL lets you specify the size of the output buffer used for writing active log data sets. The maximum size of this buffer (OUTBUFF) is 400 000 KB. Choose as large a size as your system can tolerate to decrease the number of forced I/O operations that occur because there are no more buffers. A large size can also reduce the number of wait conditions. A non-zero value for **A** in Figure 71 on page 696 is an indicator that your output buffer is too small. Ensure that the size you choose is backed up by real storage. A non-zero value for **B** in Figure 71 on page 696 is an indicator that your output buffer is too large for the amount of available real storage.

```
LOG ACTIVITY                  QUANTITY  /SECOND  /THREAD  /COMMIT
---------------------------   --------  -------  -------  -------
READS SATISFIED-OUTPUT BUFF    211.00     0.12      N/C     0.00
READS SATISFIED-OUTP.BUF(%)    100.00
READS SATISFIED-ACTIVE LOG       0.00     0.00      N/C     0.00
READS SATISFIED-ACTV.LOG(%)      0.00
READS SATISFIED-ARCHIVE LOG      0.00     0.00      N/C     0.00
READS SATISFIED-ARCH.LOG(%)      0.00

TAPE VOLUME CONTENTION WAIT      0.00     0.00      N/C     0.00
READ DELAYED-UNAVAIL.RESOUR      0.00     0.00      N/C     0.00
ARCHIVE LOG READ ALLOCATION      0.00     0.00      N/C     0.00
ARCHIVE LOG WRITE ALLOCAT.       0.00     0.00      N/C     0.00
CONTR.INTERV.OFFLOADED-ARCH      0.00     0.00      N/C     0.00
LOOK-AHEAD MOUNT ATTEMPTED       0.00     0.00      N/C     0.00
LOOK-AHEAD MOUNT SUCCESSFUL      0.00     0.00      N/C     0.00

UNAVAILABLE OUTPUT LOG BUFF  A   0.00     0.00      N/C     0.00
OUTPUT LOG BUFFER PAGED IN   B   8.00     0.00      N/C     0.10

LOG RECORDS CREATED  C         234.00     0.13      N/C     2.85
LOG CI CREATED  D                9.00     0.01      N/C     0.11
LOG WRITE I/O REQ (COPY1&2)     96.00     0.05      N/C     1.17
LOG CI WRITTEN (COPY1&2)        96.00     0.05      N/C     1.17
LOG RATE FOR 1 LOG (MB/Sec)       N/A     0.00      N/A      N/A
LOG WRITE SUSPENDED             39.00     0.02      N/C     0.48
```

Figure 71. Log statistics in the OMEGAMON statistics report

- **Choose fast devices for log data sets:** The devices assigned to the active log data sets must be fast ones. Because of its very high sequential performance, ESS is particularly recommended in environments in which the write activity is high to avoid logging bottlenecks.
- **Avoid device contention:** Place the copy of the bootstrap data set and, if using dual active logging, the copy of the active log data sets, on volumes that are accessible on a path different than that of their primary counterparts.
- **Preformat new active log data sets:** Whenever you allocate new active log data sets, preformat them using the DSNJLOGF utility described in Part 3 of *DB2 Utility Guide and Reference*. This action avoids the overhead of preformatting the log, which normally occurs at unpredictable times.

Log reads

During a rollback, restart, and database recovery, the performance impact of log reads is evident. DB2 must read from the log and apply changes to the data on disk. Every process that requests a log read has an input buffer dedicated to that process. DB2 searches for log records in the following order:

1. Output buffer
2. Active log data set
3. Archive log data set

If the log records are in the output buffer, DB2 reads the records directly from that buffer. If the log records are in the active or archive log, DB2 moves those log records into the input buffer used by the reading process (such as a recovery job or a rollback).

It is always fastest for DB2 to read the log records from the active log rather than the archive log. Access to archived information can be delayed for a considerable length of time if a unit is unavailable or if a volume mount is required (for example, a tape mount).

Recommendations: To improve log read performance:

- **Archive to disk:** If the archive log data set resides on disk, it can be shared by many log readers. In contrast, an archive on tape cannot be shared among log readers. Although it is always best to avoid reading archives altogether, if a process must read the archive, that process is serialized with anyone else who must read the archive tape volume. For example, every rollback that accesses the archive log must wait for any previous rollback work that accesses the same archive tape volume to complete.
- **Avoid device contention on the log data sets:** Place your active log data sets on different volumes and I/O paths to avoid I/O contention in periods of high concurrent log read activity.

  When there are multiple concurrent readers of the active log, DB2 can ease contention by assigning some readers to a second copy of the log. Therefore, for performance and error recovery, use dual logging and place the active log data sets on a number of different volumes and I/O paths. Whenever possible, put data sets within a copy or within different copies on different volumes and I/O paths. Ensure that no data sets for the first copy of the log are on the same volume as data sets for the second copy of the log.
- **Stripe active log data sets:** The active logs can be striped using DFSMS. *Striping* is a technique to improve the performance of data sets that are processed sequentially. Striping is achieved by splitting the data set into segments or stripes and spreading those stripes across multiple volumes. Striping can improve the maximum throughput log capacity and is most effective when there are many log records between commits. Striping is useful if you have a high I/O rate for the logs. Striping is needed more with ESCON® channels than with the faster FICON™ channels.

Log capacity

The capacity that you specify for the active log affects DB2 performance significantly. If you specify a capacity that is too small, DB2 might need to access data in the archive log during rollback, restart, and recovery. Accessing an archive takes a considerable amount of time.

The following subsystem parameters affect the capacity of the active log. In each case, increasing the value you specify for the parameter increases the capacity of the active log. See Part 2 of *DB2 Installation Guide* for more information on updating the active log parameters. The parameters are:

- The NUMBER OF LOGS field on installation panel DSNTIPL controls the number of active log data sets you create.
- The ARCHIVE LOG FREQ field on installation panel DSNTIPL is where you provide an estimate of how often active log data sets are copied to the archive log.
- The UPDATE RATE field on installation panel DSNTIPL is where you provide an estimate of how many database changes (inserts, update, and deletes) you expect per hour.

  The DB2 installation CLIST uses UPDATE RATE and ARCHIVE LOG FREQ to calculate the data set size of each active log data set.
- The CHECKPOINT FREQ field on installation panel DSNTIPN specifies the number of log records that DB2 writes between checkpoints or the number of minutes between checkpoints.

Part 2 of *DB2 Installation Guide* goes into more detail on the relationships among these parameters and their effects on operations and performance.

Total capacity and the number of logs

Determining the configuration of your active log data sets is challenging. That is, you need to have sufficient capacity in the active log to avoid reading the archives, and you need to consider how that total capacity should be divided. Having too many or too few active log data sets has ramifications. This information is summarized in Table 108.

Table 108. The effects of installation options on log data sets. You can modify the size of the data sets in installation job DSNTIJIN

Value for ARCHIVE LOG FREQ	Value for NUMBER OF LOGS	Result
Low	High	Many small data sets. Can cause operational problems when archiving to tape. Checkpoints occur too frequently.
High	Low	Few large data sets. Can result in a shortage of active log data sets.

Choosing a checkpoint frequency

At least one checkpoint is taken each time DB2 switches to a new active log data set. If the data sets are too small, checkpoints occur too frequently. As a result, database writes are not efficient. As a guideline, provide enough active log space for at least 10 checkpoint intervals.

A checkpoint interval is defined by the number you specify for checkpoint frequency (the CHECKPOINT FREQ subsystem parameter). You can specify the interval in terms of the number of log records that are written between checkpoints or the number of minutes between checkpoints. Avoid taking more than one checkpoint per minute by raising the CHECKPOINT FREQ value so that the checkpoint interval becomes at least one minute during peak periods. You can change CHECKPOINT FREQ dynamically with the SET LOG or SET SYSPARM command.

Calculating average log record size

One way to determine how much log volume you need is to consider the average size of a log record. As a general estimate, you can start with 200 bytes as the average size. To increase the accuracy of this estimate, get the real size of the log records that are written. To calculate the average size of log records that are written, you need values from the statistics report shown in Figure 71 on page 696: the LOG RECORDS CREATED counter (**C**) and the number of control intervals created in the active log counter (**D**). Use the following formula:

```
D × 4 KB / C = avg size of log record
```

Increasing the number of active log data sets

You can use the change log inventory utility (DSNJU003) to add more active log data sets to the BSDS. If the BSDS conversion utility (DSNJCNVB) has been run, you can specify up to a maximum of 93 data sets per active log copy instead of 31.

For more information about the change log inventory utility, see *DB2 Utility Guide and Reference*. For more information about the BSDS conversion utility, see Part 2 of *DB2 Installation Guide*.

Tips on setting the size of active log data sets

You can modify installation job DSNTIJIN to change the size of your active log data set. Some things to consider:

- When you calculate the size of the active log data set, identify the longest unit of work in your application programs. For example, if a batch application program commits only once every 20 minutes, the active log data set should be twice as large as the update information produced during this period by all of the application programs that are running.

  Allow time for possible operator interventions, I/O errors, and tape drive shortages if off-loading to tape. DB2 supports up to 20 tape volumes for a single archive log data set. If your archive log data sets are under the control of DFSMShsm, also consider the Hierarchical Storage Manager recall time, if the data set has been migrated by Hierarchical Storage Manager.

  For more information on determining and setting the size of your active log data sets, refer to *DB2 Installation Guide*.
- When archiving to disk, set the primary space quantity and block size for the archive log data set so that you can offload the active log data set without forcing the use of secondary extents in the archive log data set. This action avoids space abends when writing the archive log data set.
- Make the number of records for the active log be divisible by the blocking factor of the archive log (disk or tape).

  DB2 always writes complete blocks when it creates the archive log copy of the active log data set. If you make the archive log blocking factor evenly divisible into the number of active log records, DB2 does not have to pad the archive log data set with nulls to fill the block. This action can prevent REPRO errors if you should ever have to REPRO the archive log back into the active log data set, such as during disaster recovery.

  To determine the blocking factor of the archive log, divide the value specified on the BLOCK SIZE field of installation panel DSNTIPA by 4096 (that is, BLOCK SIZE / 4096). Then modify the DSNTIJIN installation job so that the number of records in the DEFINE CLUSTER field for the active log data set is a multiple of the blocking factor.
- If you offload to tape, consider adjusting the size of each of your active log data sets to contain the same amount of space as can be stored on a nearly full tape volume. This minimizes tape handling and volume mounts and maximizes the use of the tape resource.

  If you change the size of your active log data set to fit on one tape volume, remember that the bootstrap data set is copied to the tape volume along with the copy of the active log data set. Therefore, decrease the size of your active log data set to offset the space that is required on the archive tape for the bootstrap data set.

Controlling the amount of log data

Certain processes cause a large amount of information to be logged, requiring a large amount of log space.

Utilities

The utility operations REORG and LOAD LOG(YES) cause all reorganized or loaded data to be logged. For example, if a table space contains 200 million rows of data, this data, along with control information, is logged when this table space is the object of a REORG utility job. If you use REORG with the DELETE option to eliminate old data in a table and run CHECK DATA to delete rows that are no longer valid in dependent tables, you can use LOG(NO) to control log volume.

Recommendation: When populating a table with many records or reorganizing table spaces or indexes, specify LOG(NO) and take an inline copy or take a full image copy immediately after the LOAD or REORG.

Specify LOG(YES) when adding less than 1% of the total table space. This creates additional logging, but eliminates the need for a full image copy.

SQL

The amount of logging performed for applications depends on how much data is changed. Certain SQL statements are quite powerful, making it easy to modify a large amount of data with a single statement. These statements include:

- INSERT with a fullselect
- Mass deletes and mass updates (except for deleting all rows for a table in a segmented table space)
- Data definition statements log an entire database descriptor for which the change was made. For very large DBDs, this can be a significant amount of logging.
- Modification to a row that contains a LOB column defined as LOG YES.

For nonsegmented table spaces, each of these statements results in the logging of all database data that change. For example, if a table contains 200 million rows of data, that data and control information are logged if all of the rows are deleted in a table using the SQL DELETE statement. No intermediate commit points are taken during this operation.

For segmented table spaces, a mass delete results in the logging of the data of the deleted records when any of the following conditions are true:

- The table is the parent table of a referential constraint.
- The table is defined as DATA CAPTURE(CHANGES), which causes additional information to be logged for certain SQL operations.
- A delete trigger is defined on the table.

Recommendations:

- For **mass delete** operations, consider using segmented table spaces. If segmented table spaces are not an option, create one table per table space and use LOAD REPLACE with no rows in the input data set to empty the entire table space.
- For **inserting a large amount of data**, instead of using an SQL INSERT statement, use the LOAD utility with LOG(NO) and take an inline copy.
- For **updates**, consider your workload when defining a table's columns. The amount of data that is logged for update depends on whether the row contains all fixed-length columns or not. For fixed-length non-compressed rows, changes are logged only from the beginning of the first updated column to the end of the last updated column.

  For varying-length rows, data is logged from the first changed byte to the end of the last updated column. (A varying-length row contains one or more varying-length columns.)

  To determine your workload type, read-intensive or update-intensive, check the log data rate. Use the formula in "Calculating average log record size" on page 698 to determine the average log size and divide that by 60 to get the average number of log bytes written per second.
  - If you log less than 5 MB per second, the workload is read-intensive.
  - If you log more than 5 MB per second, the workload is update-intensive.

  Table 109 on page 701 summarizes the recommendations for the type of row and type of workload you run.

Table 109. Recommendations for database design to reduce log quantities

	Read-intensive workload	Update-intensive workload
Fixed-length non-compressed rows		Keep frequently updated columns close to each other.
Varying-length rows	Keep varying-length columns at the end of the row to improve read performance.	Keep all frequently updated columns near the end of the row. However, if only fixed-length columns will be updated, keep those columns close to each other at the beginning of the row.

- If you have many data definition statements (CREATE, ALTER, DROP) for a single database, issue them within a single unit of work to avoid logging the changed DBD for each data definition statement. However, be aware that the DBD is locked until the COMMIT is issued.
- Use LOG NO for any LOBs that require frequent updating and for which the tradeoff of nonrecoverability of LOB data from the log is acceptable. (You can still use the RECOVER utility on LOB table spaces to recover control information that ensures physical consistency of the LOB table space.)

  Because LOB table spaces defined as LOG NO are nonrecoverable from the DB2 log, make a recovery plan for that data. For example, if you run batch updates, be sure to take an image copy after the updates are complete.

Improving disk utilization: space and device utilization

To use disk space more efficiently, you can:

- Change your allocation of data sets to keep data sets within primary allocations. To understand how DB2 extends data sets, see "Allocating and extending data sets."
- Manage them with the Hierarchical Storage Management functional component (DFSMShsm) of DFSMS, as described in "Managing DB2 data sets with DFSMShsm" on page 35.
- Compress your data, as described in "Compressing your data" on page 702.
- Choose a page size that gives you good disk use and I/O performance characteristics, as described in "Choosing a page size" on page 45.
- Evaluate the need for and characteristics of your indexes, as described in "Evaluating your indexes" on page 705.

To manage the use of disk, you can use RMF to monitor how your devices are used. Watch for usage rates that are higher than 30% to 35%, and for disk devices with high activity rates. Log devices can have more than 50% utilization without performance problems.

Allocating and extending data sets

Primary and secondary allocation sizes are the main factors that affect the amount of disk space that DB2 uses.

In general, the primary allocation must be large enough to handle the storage needs that you anticipate. The secondary allocation must be large enough for your applications to continue operating until the data set is reorganized.

If the secondary allocation space is too small, the data set might have to be extended more times to satisfy those activities that need a large space.

IFCID 0258 allows you to monitor data set extension activities by providing information, such as the primary allocation quantity, maximum data set size, high allocated space before and after extension activity, number of extents before and after the extend, maximum volumes of a VSAM data set, and number of volumes before and after the extend. Access IFCID 0258 in Statistics Class 3 (SC03) through an IFI READA request.

See Chapter 3, "Creating storage groups and managing DB2 data sets," on page 27 for more information about extending data sets.

Compressing your data

You can compress data in a table space or partition by specifying COMPRESS YES on CREATE TABLESPACE or ALTER TABLESPACE and then running the LOAD or REORG utility. When you compress data, bit strings that occur frequently are replaced by shorter strings. Information about the mapping of bit strings to their replacements is stored in a *compression dictionary*. Computer processing is required to compress data before it is stored and to decompress the data when it is retrieved from storage. In many cases, using the COMPRESS clause can significantly reduce the amount of disk space needed to store data, but the compression ratio you achieve depends on the characteristics of your data.

With compressed data, you might see some of the following performance benefits, depending on the SQL work load and the amount of compression:
- Higher buffer pool hit ratios
- Fewer I/Os
- Fewer getpage operations

As described under "Determining the effectiveness of compression" on page 704, you can use the DSN1COMP utility to determine how well your data will compress. Data in a LOB table space or a table space that is defined in a TEMP database (a table space for declared temporary tables) cannot be compressed.

Deciding whether to compress

Consider these points before compressing data:

- Data row size

  DB2 compresses the data of one record at a time. (The prefix of the record is not compressed.) As row lengths become shorter, compression yields diminishing returns because 8 bytes of overhead are required to store each record in a data page. On the other hand, when row lengths are very long, compression of the data portion of the row may yield little or no reduction in data set size because DB2 rows cannot span data pages. In the case of very long rows, using a larger page size can enhance the benefits of compression, especially if the data is accessed primarily in a sequential mode.

  If compressing the record produces a result that is no shorter than the original, DB2 does not compress the record.
- Table space size

  Compression can work very well for large table spaces. With small table spaces, the size of the compression dictionary (64 KB) can offset the space savings that compression provides.
- Processing costs

Compressing data can result in a higher processor cost, depending on the SQL work load. However, if you use IBM's synchronous data compression hardware, processor time is significantly less than if you use just the DB2-provided software simulation or an edit or field procedure to compress the data.

Decompressing a row of data costs significantly less than compressing that same row. This rule applies regardless of whether the compression uses the synchronous data compression hardware or the software simulation that is built into DB2.

The data access path that DB2 uses affects the processor cost for data compression. In general, the relative overhead of compression is higher for table space scans and is less costlier for index access.

- I/O costs

  When rows are accessed sequentially, fewer I/Os might be required to access data that is stored in a compressed table space. However, there is a tradeoff between reduced I/O resource consumption and the extra processor cost for decoding the data.

  – If random I/O is necessary to access the data, the number of I/Os will not decrease significantly, unless the associated buffer pool is larger than the table and the other applications require little concurrent buffer pool usage.
  – Some types of data compress better than others. Data that contains hexadecimal characters or strings that occur with high frequency compresses quite well, while data that contains random byte frequencies might not compress at all. For example, textual and decimal data tends to compress well because certain byte strings occur frequently.

- Data patterns

  The frequency of patterns in the data determines the compression savings. Data with many repeated strings (such as state and city names or numbers with sequences of zeros) results in good compression savings.

- Table space design

  Each table space or partition that contains compressed data has a compression dictionary, which is built by using the LOAD utility with the REPLACE or RESUME NO options or the REORG TABLESPACE utility without the KEEPDICTIONARY option for either utility. A dictionary is also built for LOAD RESUME YES without KEEPDICTIONARY if the table space has no rows. The dictionary contains a fixed number of entries, usually 4096, and resides with the data. The dictionary content is based on the data at the time it was built, and does not change unless the dictionary is rebuilt or recovered, or compression is disabled with ALTER TABLESPACE.

  If you use LOAD to build the compression dictionary, the first *n* rows loaded in the table space determine the contents of the dictionary. The value of *n* is determined by how much your data can be compressed.

  If you have a table space with more than one table and the data used to build the dictionary comes from only one or a few of those tables, the data compression might not be optimal for the remaining tables. Therefore, put a table you want to compress into a table space by itself, or into a table space that only contains tables with similar kinds of data.

  REORG uses a sampling technique to build the dictionary. This technique uses the first *n* rows from the table space and then continues to sample rows for the remainder of the UNLOAD phase. In most cases, this sampling technique produces a better dictionary than does LOAD, and using REORG might produce better results for table spaces that contain tables with dissimilar kinds of data.

For more information about using LOAD or REORG to create a compression dictionary, see Part 2 of *DB2 Utility Guide and Reference*.

- Existing exit routines

  An exit routine is executed before compressing or after decompressing, so you can use DB2 data compression with your existing exit routines. However, do not use DB2 data compression in conjunction with DSN8HUFF. (DSN8HUFF is a sample edit routine that compresses data using the Huffman algorithm, which is provided with DB2). This adds little additional compression at the cost of significant extra CPU processing.

- Logging effects

  If a data row is compressed, all data that is logged because of SQL changes to that data is compressed. Thus, you can expect less logging for insertions and deletions; the amount of logging for updates varies. Applications that are sensitive to log-related resources can experience some benefit with compressed data.

  External routines that read the DB2 log cannot interpret compressed data without access to the compression dictionary that was in effect when the data was compressed. However, using IFCID 306, you can cause DB2 to write log records of compressed data in decompressed format. You can retrieve those decompressed records by using the IFI function READS.

- Distributed data

  DB2 decompresses data before transmitting it to VTAM.

Tuning recommendation

In some cases, using compressed data results in an increase in the number of getpages, lock requests, and synchronous read I/Os. Sometimes, updated compressed rows cannot fit in the home page, and they must be stored in the overflow page. This can cause additional getpage and lock requests. If a page contains compressed fixed-length rows with no free space, an updated row probably has to be stored in the overflow page.

To avoid the potential problem of more getpage and lock requests, add more free space within the page. Start with 10% additional free space and adjust further, as needed. If, for example, 10% free space was used without compression, then start with 20% free space with compression for most cases. This recommendation is especially important for data that is heavily updated.

Determining the effectiveness of compression

Before compressing data, you can use the DSN1COMP stand-alone utility to estimate how well it will compress. After data is compressed, use compression reports and catalog statistics to determine how effectively it was compressed.

- DSN1COMP

  Use stand-alone utility DSN1COMP to find out how much space it will save and how much processing it will require to compress your data. Run DSN1COMP on a data set that contains a table space, a table space partition, or an image copy. DSN1COMP generates a report of compression statistics but does not compress the data. For instructions on using DSN1COMP, see Part 3 of *DB2 Utility Guide and Reference*.

- Compression reports

  Examine the compression reports after you use REORG or LOAD to build the compression dictionary and compress the data. Both utilities issue a report message (DSNU234I or DSNU244I). This report message gives information about

how well the data is compressed and how much space is saved. (REORG with the KEEPDICTIONARY option does not produce the report.)

- Catalog statistics

  In addition to the compression reports, these columns in the catalog tables contain information about data compression:

  – PAGESAVE column of the SYSIBM.SYSTABLEPART tells you the percentage of pages that are saved by compressing the data.

  – PCTROWCOMP columns of SYSIBM.SYSTABLES and SYSIBM.SYSTABSTATS tells you the percentage of the rows that were compressed in the table or partition the last time RUNSTATS was run. Use the RUNSTATS utility to update these catalog columns.

Evaluating your indexes

In many cases, you might be able to eliminate indexes that are no longer necessary or change the characteristics of an index to reduce disk usage. In addition to disk space savings, dropping unneeded indexes improves performance because of savings in index maintenance.

- "Eliminating unnecessary partitioning indexes"
- "Dropping indexes created to avoid sorts"
- "Using nonpadded indexes" on page 706

Eliminating unnecessary partitioning indexes

Before DB2 Version 8, when you created a partitioned table space, you needed to define a partitioning index on the table to specify the partitioning keys and the limit key values. This is called index-controlled partitioning. The partitioning index required as many data sets as the partitioned table space. Dropping partitioning indexes that are used solely to partition the data and not to access it and converting to table-controlled partitioning eliminates the disk space that is required for the partitioning index data sets. Likewise, creating new table spaces with table-controlled partitioning, in which the partitioning key and limit keys are specified in the CREATE TABLESPACE statement and not the CREATE INDEX statement, eliminates the need for any partitioning index data sets.

For more information about table-controlled partitioning, see "Implementing table-controlled partitioning" on page 51.

Dropping indexes created to avoid sorts

Drop any indexes that are defined only to avoid a sort for queries with an ORDER BY if DB2 can perform a backward scan of another index to avoid the sort. Prior to DB2 Version 8, you might have had an ascending index and created a descending version of the index for the sole purpose of avoiding a sort. For example, consider the following query:

```
SELECT C1, C2, C3 FROM T
   WHERE C1 > 1
   ORDER BY C1 DESC;
```

Having an ascending index on C1 would not have prevented a sort to order the data. To avoid the sort, you needed a descending index on C1. Beginning in Version 8, DB2 can scan an index either forwards or backwards, which can eliminate the need to have indexes with the same columns but with different ascending and descending characteristics.

For DB2 to be able to scan an index backwards, the index must be defined on the same columns as the ORDER BY and the ordering must be exactly opposite of what is requested in the ORDER BY. For example, if an index is defined as C1 DESC, C2 ASC, then DB2 can use:

- A forward scan of the index for ORDER BY C1 DESC, C2 ASC
- A backward scan of the index for ORDER BY C1 ASC, C2 DESC

However, DB2 would need to sort for either of these two ORDER BY clauses:

- ORDER BY C1 ASC, C2 ASC
- ORDER BY C1 DESC, C2 DESC

Using nonpadded indexes

When you define an index as NOT PADDED, the varying-length columns in the index are not padded to their maximum length. If the index contains at least one varying-length column, the length information is stored with the key. Thus, you can save disk space by using nonpadded indexes instead of padded indexes. The amount of savings depends on the number of varying-length columns in the index and the actual length of the columns in those indexes versus their maximum lengths.

For example, assume that you have an index key that is defined on a VARCHAR(128) column and the actual length of the key is 8 bytes. An index that is defined as NOT PADDED would require approximately 9 times less storage than an index that is defined as PADDED, as shown by the following calculation:

```
(128 + 4) / (8 + 2 + 4) = 9
```

Improving real storage utilization

This section provides specific information for both real and virtual storage tuning. With DB2, the amount of real storage often needs to be close to the amount of virtual storage. For a general overview of some factors relating to virtual storage planning, see Part 2 of *DB2 Installation Guide*.

Use the following techniques to reduce your use of real storage:

Minimize storage needed for locks: You can save real storage by using the LOCKSIZE TABLESPACE option on the CREATE TABLESPACE statements for large tables, which affects concurrency. This option is most practical when concurrent read activity without a write intent, or a single write process, is used.

You can use LOCKSIZE PAGE or LOCKSIZE ROW more efficiently when you commit your data more frequently or when you use cursor stability with CURRENTDATA NO. For more information on specifying LOCKSIZE TABLESPACE, see “Monitoring of DB2 locking” on page 869.

Reduce the number of open data sets: You can reduce the number of open data sets by:

- Including multiple tables in segmented table spaces
- Using fewer indexes
- Reducing the value you use for DSMAX

Improve the performance for sorting: The highest performance sort is the sort that is avoided. However, because some sorting cannot be avoided, make sorting as efficient as possible. For example, assign the buffer pool for your work file table spaces in database DSNDB07, which are used in sorting, to a buffer pool other than BP0, such as to BP07. For more information about how to:

- Make sorting more efficient, see "Improving the performance of sort processing" on page 685.
- Reduce the need to sort, see "Overview of index access" on page 948.
- Determine the amount of sort activity, see "Determining sort activity" on page 973.

Release unused thread storage: If you experience virtual storage constraints due to thread storage, consider having DB2 periodically free unused thread storage. To release unused storage threads, specify YES for the CONTRACT THREAD STG field on installation panel DSNTIPE. However, you should use this option only when your DB2 subsystem has many long-running threads and your virtual storage is constrained. To determine the level of thread storage on your subsystem, see IFCID 0225 or QW0225AL in statistics trace class 6.

Recommendation:

Provide for pooled threads: As described in "Using threads in INACTIVE MODE for DRDA-only connections" on page 736, distributed threads that are allowed to be pooled use less storage than inactive database access threads. On a per connection basis, pooled threads use even less storage than inactive database access threads. Inactive database access threads require around 70 KB of storage in the *ssnm*DBM1 address space per thread, and each thread must be paired with a connection. In contrast, pooled threads are not directly linked with connections. While each pooled thread requires 200 KB in the *ssnm*DBM1 address space, each thread does not require a corresponding connection, and each connection uses only about 8 KB total in the DDF address space and the extended common storage area.

Ensure ECSA size is adequate: The extended common service area (ECSA) is a system area that DB2 shares with other programs. Shortage of ECSA at the system level leads to use of the common service area.

DB2 places some load modules and data into the common service area. These modules require primary addressability to any address space, including the application's address space. Some control blocks are obtained from common storage and require global addressability. For more information, see *DB2 Installation Guide*.

Ensure EDM pool space is being used efficiently: Monitor your use of EDM pool storage using DB2 statistics and see "Tips for managing EDM storage" on page 682.

Use less buffer pool storage: Using fewer and smaller buffer pools reduces the amount of real storage space DB2 requires. Buffer pool size can also affect the number of I/O operations performed; the smaller the buffer pool, the more I/O operations needed. Also, some SQL operations, such as joins, can create a result row that will not fit on a 4-KB page. For information about this, see "Making buffer pools large enough for the workload" on page 654.

Control maximum number of LE tokens: When a function is executed and needs to access storage used by Language Environment, it obtains an LE token from the pool. Language Environment provides a common run-time environment for programming languages. A token is taken each time one of the following functions is executed:

- Log functions (LOG, LN, LOG10)

- Trigonometry functions (ACOS, ASIN, ATAN, ATANH, ATAN2, COS, COSH, SIN, SINH, TAN, and TANH)
- EXP
- POWER
- RAND
- ADD_MONTHS
- LAST_DAY
- NEXT_DAY
- ROUND_TIMESTAMP
- TRUNC_TIMESTAMP
- LOWER
- TRANSLATE
- UPPER

On completion of the call to LE, the token is returned to the pool. The MAXIMUM LE TOKENS (LEMAX) field on installation panel DSNTIP7 controls the maximum number of LE tokens that are active at any time. The LEMAX default value is 20 with a range of 0 to 50. If the value is zero, no tokens are available. If a large number of functions are executing at the same time, all the token may be used. Thus, if a statement needs a token and none is available, the statement is queued. If the statistics trace QLENTRDY is very large, indicating a delay for an application because an LE token is not immediately available, the LEMAX may be too small. If the statistics trace QLETIMEW for cumulative time spent is very large, the LEMAX may be too small. Increase the number of tokens for the MAXIMUM LE TOKENS field on installation panel DSNTIP7.

Performance and storage

To meet the diverse needs of application data, a range of storage options is available, each with different access speeds, capacities, and costs per megabyte. This broad selection of storage alternatives supports requirements for enhanced performance and expanded online storage options, providing more options in terms of performance and price.

The levels in the DB2 storage hierarchy include real storage, storage controller cache, disk, and auxiliary storage. Increasing the I/O rate and decreasing the frequency of disk access to move data between real storage and storage devices is key to good performance.

Real storage

Real storage refers to the processor storage where program instructions reside while they are executing. It also refers to where data is held, for example, data in DB2 buffer pools that has not been paged out to auxiliary storage, the EDM pools, and the sort pool. To be used, data must either reside or be brought into processor storage or processor special registers. The maximum amount of real storage that one DB2 subsystem can use is the real storage of the processor, although other limitations may be encountered first.

DB2's large capacity for buffers in real storage and its write avoidance and sequential access techniques allow applications to avoid a substantial amount of read and write I/O, combining single accesses into sequential access, so that the disk devices are used more effectively.

Storage devices

A wide range of storage devices exist. This section presents information about newer storage servers, such as the IBM Enterprise Storage Server (ESS), and older storage devices types, such as RVA and 3390.

Planning considerations for storage device types

Some storage device types are more optimal for certain types of applications. When choosing storage devices types, consider the following hardware characteristics that affect performance:

- The size of the cache
- The number of channels and type of channels that are attached and online to a group of logical volumes
- The size of non-volatile storage (NVS), if deferred write performance is a problem
- Disk arrays
- Advanced features such as Parallel Access Volumes (PAV), Multiple Allegiance, and Flashcopy

Storage servers

An I/O subsystem typically consists of many storage disks, which are housed in storage servers, for example, the IBM Enterprise Storage Server (ESS). Storage servers provide increased functionality and performance over that of Just a Bunch of Disks (JBOD) technology.

Cache is one of the additional functions. Cache acts as a secondary buffer as data is moved between real storage and disk. Storing the same data in processor storage and the cache is not useful. To be useful, the cache must be significantly larger than the buffers in real storage, store different data, or provide another performance advantage. ESS and many other new storage servers use large caches and always prestage the data in the cache. You do not need to actively manage the cache in the newer storage servers as you must do with older storage device types.

With ESS and other new storage servers, disk performance does not generally affect sequential I/O performance. The measure of disk speed in terms of RPM (revolutions per minute) is relevant only if the cache hit ratio is low and the I/O rate is very high. If the I/O rate per disk is proportional to the disk size, small disks perform better than large disks. Large disks are very efficient for storing infrequently accessed data. As with cache, spreading the data across more disks is always better.

Storage servers and channel subsystems: The channels can be more of a bottleneck than any other component of the I/O subsystem with ESS and other newer storage servers. The degree of I/O parallelism that can be sustained efficiently is largely a function of the number of channels. In general, more channels mean better performance.

However, not all channels are alike. ESCON channels, which used to be the predominant channel type, have a maximum instantaneous data transfer rate of approximately 17 MB per second. FICON channels currently have a speed of 200 MB per second. FICON is the z/OS equivalent of Open Systems Fibre Channel Protocol (FCP). The FICON speed is bidirectional, theoretically allowing 200 MB per second to be sustained in both directions. Channel adaptors in the host processor and the storage server limit the actual speed. The FICON channels in the zSeries 900 servers are faster than those in the prior processors.

Storage servers and advanced features: ESS offers many advanced features to further boost performance. Other storage servers may offer similar function.

Parallel Access Volumes (PAV): The Parallel Access Volumes (PAV) feature allows multiple concurrent I/Os on a given device when the I/O requests originate from the same system. PAVs make storing multiple partitions on the same volume with almost no loss of performance possible. In older disk subsystems, if more than one partition is placed on the same volume (intentionally or otherwise), attempts to read the partitions result in contention, which shows up as I/O subsystem queue (IOSQ) time. Without PAVs, poor placement of a single data set can almost double the elapsed time of a parallel query.

Multiple Allegiance: The Multiple Allegiance feature allows multiple active concurrent I/Os on a given device when the I/O requests originate from different systems. PAVs and multiple allegiance dramatically improve I/O performance for parallel work on the same volume by nearly eliminating IOSQ or PEND time and drastically lowering elapsed time for transactions and queries.

Flashcopy: The Flashcopy feature provides for fast copying of full volumes. After an initialization period is complete, the logical copy is considered complete but the physical movement of the data is deferred.

Peer-to-Peer Remote Copy (PPRC): The PPRC and PPRC XD (Extended Distance) provides a faster method for recovering DB2 subsystems at a remote site in the event of a disaster at the local site. For more information about using PPRC and PPRC XD, see "Backing up with RVA storage control or Enterprise Storage Server" on page 491.

Older storage device types

Unlike cache on the newer storage servers, older devices have much smaller caches. The small caches require some user management. You can use DFSMS to provide dynamic management of the cache storage.

The amount of storage controller cache: The amount of cache to use for DB2 depends primarily on the relative importance of price and performance. It is not often effective to have large memory resources for both DB2 buffers and storage controller cache. If you decide to concentrate on the storage controller cache for performance gains, then use the maximum available cache size. If the cache is substantially larger than the DB2 buffer pools, DB2 can make effective use of the cache to reduce I/O times for random I/O. For sequential I/O, the improvement the cache provides is generally small.

Sort work files: Sort work files can have a large number of concurrent processes that can overload a storage controller with a small cache and thereby degrade system performance. For example, one large sort could use 100 sequential files, needing 60 MB of storage. Unless the cache sizes are large, you might need to specify BYPASS on installation panel DSNTIPE or use DFSMS controls to prevent the use of the cache during sort processing. Separate units for sort work can give better performance.

z/OS performance options for DB2

You can set z/OS performance options for DB2 using z/OS workload management (WLM). With workload management, you define performance goals and assign a business importance to each goal. You define the goals for work in business terms, and the system decides how much resource, such as CPU and storage, should be given to the work to meet its goal.

WLM controls the dispatching priority based on the goals you supply. WLM raises or lowers the priority as needed to meet the specified goal. Thus, you do not need to fine-tune the exact priorities of every piece of work in the system and can focus instead on business objectives.

The three kinds of goals are:

- *Response time.* How quickly you want the work to be processed.
- *Execution velocity.* How fast the work should be run when ready, without being delayed for processor, storage, I/O access, and queue delay.
- *Discretionary.* A category for low priority work for which there are no performance goals.

Response times are appropriate goals for "end user" applications, such as QMF users running under the TSO address space goals, or users of CICS using the CICS work load goals. You can also set response time goals for distributed users, as described in "Using z/OS workload management to set performance objectives" on page 739.

For DB2 address spaces, velocity goals are more appropriate. A small amount of the work done in DB2 is counted toward this velocity goal. Most of the work done in DB2 applies to the end user goal.

This section describes ways to set performance options for DB2 address spaces:

- "Determining z/OS workload management velocity goals"
- "How DB2 assigns I/O priorities" on page 712

For information about WLM and defining goals through the service definition, see *z/OS MVS Planning: Workload Management*.

Determining z/OS workload management velocity goals

This section tells you how to set velocity goals when you are managing CICS, IMS, or DDF according to WLM response-time goals and you are set up to use WLM-established stored procedures address spaces. Use following service classes for velocity:

- The default SYSSTC service class for:

  VTAM and TCP/IP address spaces

  IRLM address space (IRLMPROC)

  Important: The VTAM, TCP/IP, and IRLM address spaces must always have a higher dispatching priority than all of the DBMS address spaces, their attached address spaces, and their subordinate address spaces. Do not allow WLM to reduce the priority of VTAM, TCP/IP, or IRLM to or below that of the other DBMS address spaces.
- A high velocity goal for a service class whose name you define, such as PRODREGN, for the following:

DB2 (all address spaces, except for the DB2-established stored procedures address space)

Important: The DB2-started address spaces, including *ssnm*DBM1,
*ssnm*MSTR, and *ssnm*DIST, must have the same service class.

CICS (all region types)

IMS (all region types except BMPs)

The velocity goals for CICS and IMS regions are only important during startup or restart. After transactions begin running, WLM ignores the CICS or IMS velocity goals and assigns priorities based on the goals of the transactions that are running in the regions. A high velocity goal is good for ensuring that startups and restarts are performed as quickly as possible.

Similarly, when you set response time goals for DDF threads or for stored procedures in a WLM-established address space, the only work controlled by the DDF or stored procedure velocity goals are the DB2 service tasks (work performed for DB2 that cannot be attributed to a single user). The user work runs under separate goals for the enclave, as described in "Using z/OS workload management to set performance objectives" on page 739.

For the DB2-established stored procedures address space, use a velocity goal that reflects the requirements of the stored procedures in comparison to other application work. Depending on what type of distributed work you do, this might be equal to or lower than the goal for PRODREGN.

IMS BMPs can be treated along with other batch jobs or given a velocity goal, depending on what business and functional requirements you have at your site.

Consider the following other workload management considerations:

- IRLM must be eligible for the SYSSTC service class. To make IRLM eligible for SYSSTC, do not classify IRLM to one of your own service classes.
- If you need to change a goal, change the velocity between 5 and 10%. Velocity goals do not translate directly to priority. Higher velocity tends to have higher priority, but this is not always the case.
- WLM can assign I/O priority (based on I/O delays) separately from processor priority.

  See "How DB2 assigns I/O priorities" for information about how read and write I/O priorities are determined.
- z/OS workload management dynamically manages storage isolation to meet the goals you set.

How DB2 assigns I/O priorities

DB2 informs z/OS about which address space's priority is to be associated with a particular I/O request. Then WLM handles the management of the request from there. Table 110 describes to which enclave or address space DB2 associates I/O read requests.

Table 110. How read I/O priority is determined

Request type	Synchronous reads	Prefetch reads
Local	Application's address space	Application's address space

Table 110. How read I/O priority is determined (continued)

Request type	Synchronous reads	Prefetch reads
DDF or Sysplex query parallelism (assistant only)	Enclave priority	Enclave priority

Table 111 describes to which enclave or address space DB2 associates I/O write requests.

Table 111. How write I/O priority is determined

Request type	Synchronous writes	Deferred writes
Local	Application's address space	ssnmDBM1 address space
DDF	DDF address space	ssnmDBM1 address space

IBM System z9 Integrated Information Processor usage monitoring

The System z9 Integrated Information Processor (IBM zIIP) is a specialty engine that runs eligible database workloads. The IBM zIIP is designed to help free-up general computing capacity and lower software costs for select DB2 workloads such as business intelligence (BI), enterprise resource planning (ERP), and customer relationship management (CRM) on the mainframe. The IBM zIIP is a high-speed engine that better enables you to centralize your data on the mainframe, breaking down traditional walls between transactional data stores on the mainframe and the BI, ERP, and CRM applications that run on distributed computers.

With the zIIP capability, the System z9 mainframe helps minimize the need to maintain duplicate copies of data and provides better security between applications and data. New accounting fields and improvements to RMF also permit you to project the amount of CPU time eligible to run on an IBM zIIP without installing any IBM zIIPs.

Portions of DDF server thread processing, utility processing, and star join parallel child processing can be directed to an IBM zIIP. The amount of general-purpose processor savings will vary based on the amount of workload executed by the zIIP, among other factors.

Processes that can use IBM zIIPs:

- DDF server threads that process SQL requests from applications that access DB2 by TCP/IP.
- Parallel child processes. A portion of each child process executes under a dependent enclave SRB if it processes on behalf of an application that DB2 from an allied address space, or under an independent enclave SRB if the processing is performed on behalf of a remote application that accesses DB2 by TCP/IP. The enclave priority is inherited from the invoking allied address space for a dependent enclave or from the main DDF server thread enclave classification for an independent enclave.
- Utility index build and maintenance processes for the LOAD, REORG, and REBUILD INDEX utilities. A portion of the index build/maintenance runs under a dependent enclave SRB.

Processes that cannot use IBM zIIPs:

- External stored procedures
- Triggers or functions that join the enclave SRB using a TCB
- DDF server threads that use SNA to connect to DB2

Using trace to monitor zIIP usage: The DB2 accounting trace records provide information related to application programs including processor resources consumed by the application. Accumulated zIIP CPU time is not accumulated in the existing class 1, class 2, and class 7 accounting fields.

Using RMF to monitor zIIP usage: The Resource Measurement Facility (RMF) provides information on zIIP usage to help you identify when to consider purchasing a zIIP or adding more zIIPs. Also, SMF Type 72 records contain information on zIIP usage and fields in SMF Type 30 records let you know how much time is spent on zIIPs, as well as how much time was spent executing zIIP eligible work on standard processors. For more details about the new RMF support for zIIPs, refer to the z/OS MVS Initialization and Tuning Reference.

Considerations for data sharing: In a data sharing environment, where DB2 Version 8 is part of the data sharing group, DRDA work running in the Version 7 member cannot run on the zIIP. Work running on the Version 8 member will run on the zIIP. IBM zIIP capacity is not included in the sysplex routing information returned to DB2 Connect Server. Accordingly, members without zIIPs might be favored over members with zIIPs.

Controlling resource usage

DB2 includes a resource limit facility (governor), which helps control the use of DB2 resources. Other facilities, such as z/OS workload management (WLM), and the QMF governor, complement the DB2 governor. Because DB2 is integrated with the operating system and transaction subsystems, control definitions are used in most cases. This integration simplifies the task of controlling resources for the user.

Each of the objectives presented in Table 112 is matched with a control facility that you can use to achieve the objective.

Table 112. Controlling the use of resources

Objective	How to accomplish it	Where it is described
Prioritizing resources	z/OS workload management	“Prioritizing resources” on page 715, “z/OS performance options for DB2” on page 711 and “Using z/OS workload management to set performance objectives” on page 739
Limiting resources for each job	Time limit on job or step (through z/OS system settings or JCL)	“Limiting resources for each job” on page 715
Limiting resources for TSO sessions	Time limit for TSO logon	“Limiting resources for TSO sessions” on page 715
Limiting resources for IMS and CICS	IMS and CICS controls	“Limiting resources for IMS and CICS” on page 716
Limiting resources for a stored procedure	ASUTIME column of SYSIBM.SYSROUTINES catalog table.	“Limiting resources for a stored procedure” on page 716

Table 112. Controlling the use of resources (continued)

Objective	How to accomplish it	Where it is described
Limiting dynamic statement execution time	QMF governor and DB2 resource limit facility	"Resource limit facility (governor)" on page 716
Reducing locking contention	DB2 locking parameters, DISPLAY DB LOCKS, lock trace data, database design	Chapter 31, "Improving concurrency," on page 809
Evaluating long-term resource usage	Accounting trace data, OMEGAMON reports	"OMEGAMON" on page 1196
Predicting resource consumption	DB2 EXPLAIN statement, Visual Explain, DB2 Estimator, predictive governing capability	Chapter 34, "Using EXPLAIN to improve SQL performance," on page 927 and "Predictive governing" on page 724
Controlling use of parallelism	DB2 resource limit facility, SET CURRENT DEGREE statement	"Disabling query parallelism" on page 1001

Prioritizing resources

z/OS workload management (WLM) controls the execution of DB2 work based on the priorities that you set. See *z/OS MVS Initialization and Tuning Guide* for more information about setting priorities on work.

In CICS environments without the Open Transaction Environment (OTE) function, DB2 work and application work is performed in different tasks. DB2 work is managed at the subtask level. With CICS OTE, DB2 work and application work can be performed in the same task. You can manage the DB2 subtasks through various settings in the CICS resource definition online (RDO). Without OTE, some overhead is incurred for each task switch. Therefore, depending on the SQL activity, CICS OTE can improve performance significantly because of the reduction of needing to switch tasks.

In other environments such as batch and TSO, which typically have a single task requesting DB2 services, the task-level processor dispatching priority is irrelevant. Access to processor and I/O resources for synchronous portions of the request is governed solely by WLM.

Limiting resources for each job

Because most of the resource usage occurs within the standard job structure, you can control processor usage by changing the TIME parameter for the job or step. The time limit applies even when DB2 is sorting the result rows. If the time limit is exceeded, the job step abends, and any uncommitted work is rolled back. If you want to control the total amount of resources used, rather than the amount used by a single query, then use this control.

Refer to the *z/OS MVS JCL User's Guide* for more information on setting resource limits.

Limiting resources for TSO sessions

Time limits can apply to either TSO sessions or to batch jobs. Your z/OS system programmer can provide a time parameter on the logon procedure or on a job statement in the logon preprompt exit. This time limit is for the session, rather

than for an individual query or a single program. If you want to control the amount of resources used for an entire TSO session, rather than the amount used by a single query, then use this control.

You can find more information about setting the resource limit for a TSO session in these manuals:
- *z/OS TSO/E Programming Guide*
- *z/OS TSO/E Customization*

Limiting resources for IMS and CICS

You can use various IMS commands (including PROCLIM, or processing limit) to limit the resources used by a transaction, a class, a program, a database, or a subsystem. For more information, see *IMS Command Reference*.

For a detailed description of performance factors in a CICS system, see *CICS Transaction Server for z/OS Performance Guide*.

Limiting resources for a stored procedure

DB2 stored procedures are especially designed for high volume online transactions. To establish limits for stored procedures, you can:
- Set a processor limit for each stored procedure, by updating the ASUTIME column of the SYSIBM.SYSROUTINES catalog table. This limit allows DB2 to cancel procedures that loop.
- Set a limit for the maximum number of times that a procedure can terminate abnormally, by specifying a value in the MAX ABEND COUNT field on installation panel DSNTIPX. This limit is a system limit that applies to all stored procedures and prevents a problem procedure from overwhelming the system with abend dump processing.
- Set a limit for the maximum number of times that a specific procedure can terminate abnormally, by specifying the STOP AFTER FAILURES option on the ALTER or CREATE PROCEDURE statement. This limit allows you to override the system limit specified in MAX ABEND COUNT and specify different limits for different procedures.

For information about controlling the amount of storage used by stored procedures address spaces, see “Controlling address space storage” on page 1019.

Resource limit facility (governor)

The DB2 resource limit facility (governor) lets you perform the following activities:
- Set warning and error thresholds for dynamic SELECT, INSERT, UPDATE, or DELETE statements. The resource limit facility can inform users (via your application programs) that a processing limit might be exceeded for a particular dynamic SQL statement. This is called *predictive governing*.
- Stop dynamic SQL statements (SELECT, INSERT, UPDATE, or DELETE) that exceed the processor limit that you have specified. This is sometimes called *reactive governing*. The resource limit facility does not control static SQL statements whether or not they are executed locally or remotely.
- Restrict bind and rebind activities to avoid performance impacts on production data.
- Restrict particular parallelism modes for dynamic queries.

Note: No limits apply to primary or secondary authorization IDs with installation SYSADM or installation SYSOPR authority.

Data sharing: See *DB2 Data Sharing: Planning and Administration* for information about special considerations for using the resource limit facility in a data sharing group.

This section includes the following topics:

- "Using resource limit tables (RLSTs)"
- "Governing dynamic queries" on page 722
- "Restricting bind operations" on page 726
- "Restricting parallelism modes" on page 727

Using resource limit tables (RLSTs)

Use one or more *resource limit specification tables* (RLSTs) to give governing information to DB2.

If you are a system administrator, you must determine how your location intends to use the governor and create several local procedures. Establish a procedure for creating and maintaining your RLSTs, and for establishing limits for any newly written applications. Develop procedures for console operators, such as switching RLSTs every day at a certain time.

Creating an RLST

Resource limit specification tables can reside in any database; however, because a database has some special attributes while the resource limit facility is active, it is best to put RLSTs in their own database.

When you install DB2, installation job DSNTIJSG creates a database, table space, table, and descending index for the resource limit specification. You can tailor those statements. For more information about job DSNTIJSG, see Part 2 of *DB2 Installation Guide*.

To create a new resource limit specification table, use the following statements, also included in installation job DSNTIJSG. You must have sufficient authority to define objects in the DSNRLST database and to specify *authid*, which is the authorization ID specified on field RESOURCE AUTHID of installation panel DSNTIPP.

Creating the table: Use the following statement:

```
CREATE TABLE authid.DSNRLSTxx
       (AUTHID   VARCHAR(128)     NOT NULL WITH DEFAULT,
        PLANNAME CHAR(8)       NOT NULL WITH DEFAULT,
        ASUTIME  INTEGER,
        -------3-column format --------
        LUNAME   CHAR(8)       NOT NULL WITH DEFAULT,
        -------4-column format --------
        RLFFUNC  CHAR(1)       NOT NULL WITH DEFAULT,
        RLFBIND  CHAR(1)       NOT NULL WITH DEFAULT,
        RLFCOLLN VARCHAR(128)     NOT NULL WITH DEFAULT,
        RLFPKG   VARCHAR(128)  NOT NULL WITH DEFAULT),
        -------8-column format --------
        RLFASUERR  INTEGER,
        RLFASUWARN INTEGER,
        RLF_CATEGORY_B CHAR(1) NOT NULL WITH DEFAULT)
        -------11-column format --------
         IN DSNRLST.DSNRLSxx;
```

The name of the table is *authid*.DSNRLST*xx*, where *xx* is any two-character alphanumeric value, and *authid* is specified when DB2 is installed. Because the two characters *xx* must be entered as part of the START command, they must be alphanumeric—no special or DBCS characters.

All future column names defined by IBM will appear as RLF*xxxxx*. To avoid future naming conflicts, begin your own column names with characters other than RLF.

Creating the index: To create an index for the 11-column format, use the following SQL:

```
CREATE UNIQUE INDEX authid.DSNARLxx
       ON authid.DSNRLSTxx
         (RLFFUNC, AUTHID DESC, PLANNAME DESC,
          RLFCOLLN DESC, RLFPKG DESC, LUNAME DESC)
         CLUSTER CLOSE NO;
```

The *xx* in the index name (DSNARL*xx*) must match the *xx* in the table name (DSNRLST*xx*) and it must be a descending index.

Populating the RLST: Use the SQL statements INSERT, UPDATE, and DELETE to populate the resource limit specification table. The limit that exists when a job makes its first dynamic SELECT, INSERT, UPDATE, or DELETE statement applies throughout the life of the job. If you update the resource limit specification table while a job is executing, that job's limit does not change; instead, the updates are effective for all new jobs and for those that have not issued their first dynamic SELECT, INSERT, UPDATE, or DELETE statement.

To insert, update, or delete from the resource limit specification table, you need only the usual table privileges on the RLST. No higher authority is required.

Starting and stopping the RLST: Activate any particular RLST by using the DB2 command START RLIMIT ID=*xx* where *xx* is the two-character identifier that you specified on the name DSNRLSTxx. This command gives you the flexibility to use a different RLST at different times; however, only one RLST can be active at a time.

Example: You can use different RLSTs for the day shift and the evening shift, as shown in Table 113 and Table 114.

Table 113. Example of RLST for the day shift

AUTHID	PLANNAME	ASUTIME	LUNAME
BADUSER		0	LUDBD1
ROBYN		100000	LUDBD1
	PLANA	300000	LUDBD1
		50000	LUDBD1

Table 114. Example of RLST for the night shift. During the night shift AUTHID, ROBYN, and all PLANA users from LUDBD1 run without limit.

AUTHID	PLANNAME	ASUTIME	LUNAME
BADUSER		0	LUDBD1
ROBYN		NULL	LUDBD1
	PLANA	NULL	LUDBD1
		50000	LUDBD1

At installation time, you can specify a default RLST to be used each time that DB2 is restarted. For more information on resource limit facility subsystem parameters, see Part 2 of *DB2 Installation Guide*.

If the governor is active and you restart it without stopping it, any jobs that are active continue to use their original limits, and all new jobs use the limits in the new table.

If you stop the governor while a job is executing, the job runs with *no limit*, but its processing time continues to accumulate. If you later restart the governor, the new limit takes effect for an active job only when the job passes one of several internal checkpoints. A typical dynamic statement, which builds a result table and fetches from it, passes those checkpoints at intervals that can range *from moments to hours*. As a result, your change to the governor might not stop an active job within the time you expect.

Use the DB2 command CANCEL THREAD to stop an active job that does not pick up the new limit when you restart the governor.

Restricted activity on the RLST: While the governor is active, you cannot execute the following SQL statements on the RLST, or the table space and database in which the RLST is contained:
- DROP DATABASE
- DROP INDEX
- DROP TABLE
- DROP TABLESPACE
- RENAME TABLE

You cannot stop a database or table space that contains an active RLST; nor can you start the database or table space with ACCESS(UT).

Descriptions of the RLST columns

Here is a complete description for all RLST columns. In no case will all columns in a particular row be populated; the columns you must populate depend on what function is performed by that row (determined by the value in RLFFUNC) and how narrowly you want to qualify values. For example, you can qualify broadly by leaving the AUTHID column blank, which means that the row applies to all authorization IDs. Or, you can qualify very narrowly by specifying a different row for each authorization ID for which the function applies.

Search order: DB2 tries to find the most exact match when it determines which row to use for a particular function. The search order depends on which function is being requested (type of governing, bind operations, or parallelism restrictions). The search order is described under each of those functions.

AUTHID
The resource specification limits apply to the primary authorization ID that you specify in this column. A blank means that the limit specifications in this row apply to all authorization IDs for the location that is specified in
LUNAME. No limits apply to primary or secondary authorization IDs with
installation SYSADM or installation SYSOPR authority.

PLANNAME
The resource specification limits apply to this plan. If you are specifying a function that applies to plans (RLFFUNC=blank or '6'), a blank means that the limit specifications in this row apply to all plans for the location that is specified in LUNAME. Qualify by plan name only if the dynamic

statement is issued from a DBRM bound in a plan, not a package; otherwise, DB2 does not find this row. If the RLFFUNC column contains a function for packages ('1,' '2,' or '7'), this column must be blank; if it is not blank, the row is ignored.

ASUTIME
The number of processor service units allowed for any single dynamic SELECT, INSERT, UPDATE, or DELETE statement. Use this column for reactive governing.

Other possible values and their meanings are:

null No limit

0 (zero) or a negative value
No dynamic SELECT, INSERT, UPDATE, or DELETE statements are permitted.

The governor samples the processing time in service units. Service units are independent of processor changes. The processing time for a particular SQL statement varies according to the processor on which it is executed, but the service units required remains roughly constant. The service units consumed are not exact between different processors because the calculations for service units are dependent on measurement averages performed before new processors are announced. A relative metric is used so that the RLST values do not need to be modified when processors are changed. However, in some cases, DB2 workloads can differ from the measurement averages. In these cases, RLST value changes may be necessary. For information about how to calculate service units, see "Calculating service units" on page 726.

LUNAME
The LU name of the location where the request originated. A blank value in this column represents the local location, *not* all locations. The value PUBLIC represents all of the DBMS locations in the network; these locations do not need to be DB2 subsystems. PUBLIC is the only value for TCP/IP connections.

RLFFUNC
Specifies how the row is used. The values that have an effect are:

blank The row reactively governs dynamic SELECT, INSERT, UPDATE, or DELETE statements by plan name.

'1' The row reactively governs bind operations.

'2' The row reactively governs dynamic SELECT, INSERT, UPDATE, or DELETE statements by package or collection name.

'3' The row disables query I/O parallelism.

'4' The row disables query CP parallelism.

'5' The row disables Sysplex query parallelism.

'6' The row predictively governs dynamic SELECT, INSERT, UPDATE, or DELETE statements by plan name.

'7' The row predictively governs dynamic SELECT, INSERT, UPDATE, or DELETE statements by package or collection name.

All other values are ignored.

RLFBIND
Shows whether bind operations are allowed. An 'N' implies that bind operations are not allowed. Any other value means that bind operations are allowed. This column is used only if RLFFUNC is set to '1'.

RLFCOLLN
Specifies a package collection. A blank value in this column means that the row applies to all package collections from the location that is specified in LUNAME. Qualify by collection name only if the dynamic statement is issued from a package; otherwise DB2 does not find this row. If RLFFUNC=blank, '1,' or '6', then RLFCOLLN must be blank.

RLFPKG
Specifies a package name. A blank value in this column means that the row applies to all packages from the location that is specified in LUNAME. Qualify by package name only if the dynamic statement is issued from a package; otherwise DB2 does not find this row. If RLFFUNC=blank, '1', or '6', then RLFPKG must be blank.

RLFASUERR
Used for predictive governing (RLFFUNC= '6' or '7'), and only for statements that are in cost category A. The error threshold number of system resource manager processor service units allowed for a single dynamic SELECT, INSERT, UPDATE, or DELETE statement. If the predicted processor cost (in service units) is greater than the error threshold, an SQLCODE -495 is returned to the application.

Other possible values and their effects are:

null No error threshold

0 (zero) or a negative value
All dynamic SELECT, INSERT, UPDATE, or DELETE statements receive SQLCODE -495.

RLFASUWARN
Used for predictive governing (RELFFUNC= '6' or '7'), and only for statements that are in cost category A. The warning threshold number of processor service units that are allowed for a single dynamic SELECT, INSERT, UPDATE, or DELETE statement. If the predicted processor cost (in service units) is greater than the warning threshold, an SQLCODE +495 is returned to the application.

Other possible values and their effects are:

null No warning threshold

0 (zero) or a negative value
All dynamic SELECT, INSERT, UPDATE, or DELETE statements receive SQLCODE +495.

Important: Make sure the value for RLFASUWARN is less than that for RLFASUERR. If the warning value is higher, the warning is never reported. The error takes precedence over the warning.

RLF_CATEGORY_B
Used for predictive governing (RLFFUNC='6' or '7'). Tells the governor the default action to take when the cost estimate for a given statement falls into cost category B, which means that the predicted cost is indeterminate and probably too low. You can tell if a statement is in cost category B by running EXPLAIN and checking the COST_CATEGORY column of the DSN_STATEMNT_TABLE.

The acceptable values are:

blank By default, prepare and execute the SQL statement.

Y Prepare and execute the SQL statement.

N Do not prepare or execute the SQL statement. Return SQLCODE -495 to the application.

W Complete the prepare, return SQLCODE +495, and allow the application logic to decide whether to execute the SQL statement or not.

Governing dynamic queries

You can use two modes of governing to control the amount of system resources that a dynamic SELECT, INSERT, UPDATE, or DELETE uses. These modes are called *reactive* and *predictive*. You can use either mode or both together.

Specifying reactive governing: Specify either of the following values in the RLFFUNC column of the RLST:

blank Govern by plan name

'2' Govern by package name

Any statement that exceeds a limit you set in the RLST terminates with a -905 SQLCODE and a corresponding '57014' SQLSTATE. You can establish a single limit for all users, different limits for individual users, or both. Limits do not apply to primary or secondary authorization IDs with installation SYSADM or installation SYSOPR authority. For queries entering DB2 from a remote site, the local site limits are used.

Specifying predictive governing: Specify either of the following values in the RLFFUNC column of the RLST:

'6' Govern by plan name

'7' Govern by package name

See "Qualifying rows in the RLST" for more information about how to qualify rows in the RLST. See "Predictive governing" on page 724 for more information about using predictive governing.

This section includes the following topics:

- "Qualifying rows in the RLST"
- "Predictive governing" on page 724
- "Combining reactive and predictive governing" on page 725
- "Governing statements from a remote site" on page 725
- "Calculating service units" on page 726

Qualifying rows in the RLST

DB2 tries to find the closest match when determining limits for a particular dynamic statement. In summary, the search order of matching is as follows:

1. Exact match
2. Authorization ID
3. Plan name, or collection name and package name
4. LU name

5. No row match

Governing by plan or package name: Governing by plan name and package name are mutually exclusive.

- Plan name

  The RLF governs the DBRMs in the MEMBER list specified on the BIND PLAN command. The RLFFUNC, RLFCOLLN, and RLFPKG columns must contain blanks. Table 115 is an example of this:

Table 115. Qualifying rows by plan name

RLFFUNC	AUTHID	PLANNAME	LUNAME	ASUTIME
(blank)	JOE	PLANA	(blank)	(null)
(blank)	(blank)	WSPLAN	SAN_JOSE	15000
(blank)	(blank)	(blank)	PUBLIC	10000

  The first row in Table 115 shows that when Joe runs PLANA at the local location, there are no limits for any dynamic statements in that plan.

  The second row shows that if anyone runs WSPLAN from SAN_JOSE, the dynamic statements in that plan are restricted to 15 000 SUs each.

  The third row is entered as a cap for any unknown authorization IDs or plan names from any location in the network, including the local location. (An alternative would be to let the default values on installation panel DSNTIPR and DSNTIPO serve as caps.)

- Collection and package name

  The resource limit facility governs the packages used during the execution of the SQL application program. PLANNAME must contain blank, and RLFFUNC must contain '2'.Table 116 is an example of this:

Table 116. Qualifying rows by collection or package name

	RLFFUNC	AUTHID	RLFCOLLN	RLFPKG	LUNAME	ASUTIME
#	2	JOE	DSNESPCS	(blank)	(blank)	40000
#	2	(blank)	(blank)	DSNESM68	PUBLIC	15000

  The first row in Table 116 shows that when Joe runs any package in collection
DSNESPCS from the local location, dynamic statements are restricted to 40 000 SUs.

  The second row indicates that if anyone from any location (including the local
location) runs SPUFI packageDSNESM68, dynamic statements are limited to 15 000 SUs.

Governing by LU name: Specify an originating system's LU name in the LUNAME column, or, specify PUBLIC for all remote LUs. An LUNAME with a value other than PUBLIC takes precedence over PUBLIC. If you leave LUNAME blank, DB2 assumes that you mean the local location only and none of your incoming distributed requests will qualify. PUBLIC is the only value for TCP/IP connections.

Setting a default for when no row matches: If no row in the RLST matches the currently executing statement, then DB2 uses the default set on the RLST ACCESS ERROR field of installation panel DSNTIPO (for queries that originate locally) or DSNTIPR (for queries that originate remotely). This default applies to reactive governing only.

For predictive governing, if no row matches, then no predictive governing occurs.

Predictive governing

DB2's predictive governing capability has an advantage over the reactive governor in that it avoids wasting processing resources by giving you the ability to prevent a query from running when it appears that it exceeds processing limits. With the reactive governor, those resources are already used before the query is stopped.

See Figure 72 for an overview of how predictive governing works.

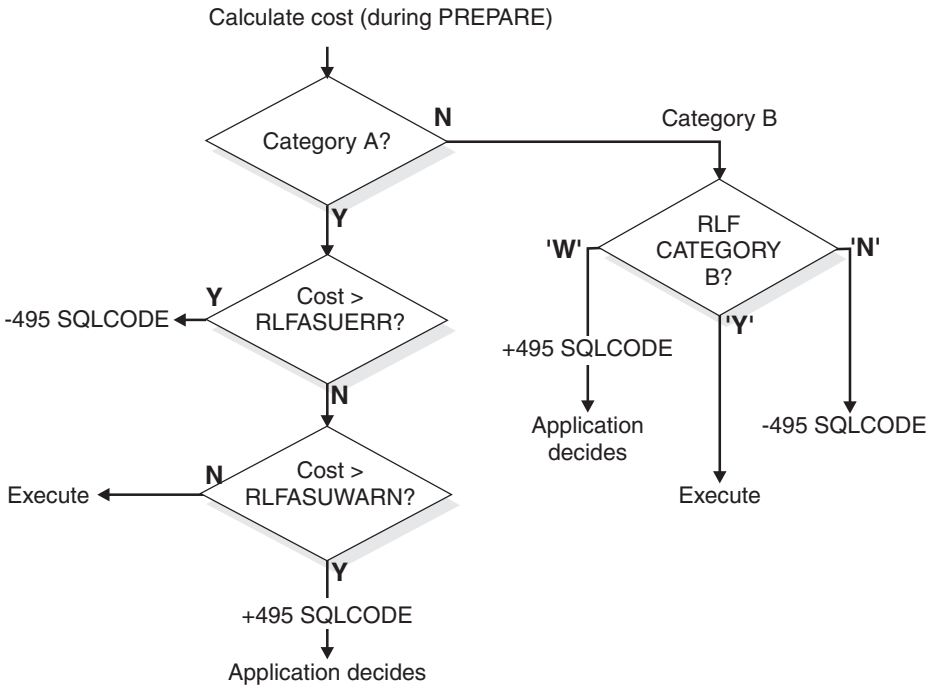


Figure 72. Processing for predictive governing

At prepare time for a dynamic SELECT, INSERT UPDATE, or DELETE statement, DB2 searches the active RLST to determine if the processor cost estimate exceeds the error or warning threshold that you set in RLFASUWARN and RLFASUERR columns for that statement. DB2 compares the cost estimate for a statement to the thresholds you set, and the following actions occur:

- If the cost estimate is in cost category A and the error threshold is exceeded, DB2 returns a -495 SQLCODE to the application, and the statement is not prepared or run.
- If the estimate is in cost category A and the warning threshold is exceeded, a +495 SQLCODE is returned at prepare time. The prepare is completed, and the application or user decides whether to run the statement.
- If the estimate is in cost category B, DB2 takes the action you specify in the RLF_CATEGORY_B column; that is, it either prepares and executes the statement, does not prepare or execute the statement, or returns a warning SQLCODE, which lets the application decide what to do.
- If the estimate is in cost category B and the warning threshold is exceeded, a +495 SQLCODE is returned at prepare time. The prepare is completed, and the application or user decides whether to run the statement.

Example: Table 117 on page 725 is an RLST with two rows that use predictive governing.

Table 117. Predictive governing example

RLFFUNC	AUTHID	RLFCOLLN	RLFPKG	RLFASUWARN	RLFASUERR	RLF_CATEGORY_B
7	(blank)	COLL1	C1PKG1	900	1500	Y
7	(blank)	COLL2	C2PKG1	900	1500	W

The rows in the RLST for this example cause DB2 to act as follows for all dynamic INSERT, UPDATE, DELETE, and SELECT statements in the packages listed in this table (C1PKG1 and C2PKG1):

- Statements in cost category A that are predicted to be less than 900 SUs will execute.
- Statements in cost category A that are predicted to be between 900 and 1500 SUs receive a +495 SQLCODE.
- Statements in cost category A that are predicted to be greater than 1500 SUs receive SQLCODE -495, and the statement is not executed.

Cost category B: The two rows differ only in how statements in cost category B are treated. For C1PKG1, the statement will execute. For C2PKG2, the statements receive a +495 SQLCODE and the user or application must decide whether to execute the statement.

Combining reactive and predictive governing

A dynamic statement can be governed both before and after the statement is executed. For example, if the processing cost estimate is in cost category B and you decide to run the statement, you can use the RLST to terminate the statement after a certain amount of processor time, the same as it does today. To use both modes of governing, you need two rows in the RLST as shown in Table 118.

Table 118. Combining reactive and predictive governing

RLFFUNC	AUTHID	PLANNAME	ASUTIME	RLFASUWARN	RLFASUERR	RLF_CATEGORY_B
6	USER1	PLANA	0	800	1000	W
(blank)	USER1	PLANA	1100	0	0	(blank)

The rows in the RLST for this example cause DB2 to act as follows for a dynamic SQL statement that runs under PLANA:

Predictive mode:

- If the statement is in COST_CATEGORY A and the cost estimate is greater than 1000 SUs, USER1 receives SQLCODE -495 and the statement is not executed.
- If the statement is in COST_CATEGORY A and the cost estimate is greater than 800 SUs but less than 1000 SUs, USER1 receives SQLCODE +495.
- If the statement is in COST_CATEGORY B, USER1 receives SQLCODE +495.

Reactive mode: In either of the following cases, a statement is limited to 1100 SUs:

- The cost estimate for a statement in COST_CATEGORY A is less than 800 SUs
- The cost estimate for a COST_CATEGORY A is greater than 800 and less than 1000 or is in COST_CATEGORY B and the user chooses to execute the statement

Governing statements from a remote site

For distributed processing, keep in mind the following guidelines:

- For dynamic statements coming from requesters using DRDA protocols, you must govern by package name (RLFFUNC=2 or RLFFUNC=7), which means that PLANNAME must be blank. Specify the originating system's LU name in the LUNAME column, or specify PUBLIC for all remote LUs. If you leave LUNAME blank, DB2 assumes that you mean the local location only and none of your incoming distributed requests will qualify.
- For dynamic statements coming from requesters using DB2 private protocol, you must govern by plan name (RLFFUNC=(blank) or '6'), which means that RLFCOLLN and RLFPKG must be blank. Specify the originating system's LU name in the LU column, or specify PUBLIC for all remote LUs. Again, a value other than PUBLIC takes precedence over PUBLIC. PLANNAME can be blank or the name of the plan created at the requester's location. RLFPKG and RLFCOLLN must be blank.
- For dynamic statements coming from requesters using TCP/IP, you cannot specify the LU name. You must use PUBLIC.
- If no row is present in the RLST to govern access from a remote location, the limit is the default set on the RLST ACCESS ERROR field of installation panel DSNTIPR.

Calculating service units

The easiest way to get SU times for the ASUTIME, RLFASUWARN, or RLFASUERR columns is to use the value in the PROCSU column of DSN_STATEMNT_TABLE as your starting point. You can also get the value from the IFCID 0022 record.

However, if you don't have statement table information or if there are adhoc queries for which you have no information, you can use the following formula to calculate SU time:

```
SU time = processor time × service units per second value
```

The value for service units per second depends on the processor model. You can find this value for your processor model in *z/OS MVS Initialization and Tuning Guide*, where SRM is discussed.

For example, if processor A is rated at 900 service units per second and you do not want any single dynamic SQL statement to use more than 10 seconds of processor time, you could set ASUTIME as follows:

```
ASUTIME time = 10 seconds × 900 service units/second = 9000 service units
```

Later, you could upgrade to processor B, which is rated at 1000 service units per second. If the value you set for ASUTIME remains the same (9000 service units), your dynamic SQL is only allowed 9 seconds for processing time but an equivalent number of processor service units:

```
ASUTIME = 9 seconds × 1000 service units/second = 9000 service units
```

As this example illustrates, after you establish an ASUTIME (or RLFASUWARN or RLFASUERR) for your current processor, there is no need to modify it when you change processors.

Restricting bind operations

To restrict bind operations, use RLF function 1 (RLFFUNC='1') and qualify rows by authorization ID and LU name. The same precedence search order applies:

1. AUTHID and LUNAME match
2. AUTHID matches

3. LUNAME matches

   A value of PUBLIC for LUNAME applies to all authorization IDs at all locations, while a blank LUNAME governs bind operations for IDs at the local location only.

4. If no entry matches, or if your RLST cannot be read, then the resource limit facility does not disable bind operations.

Example

Table 119 is an example of an RLST that disables bind operations for all but three authorization IDs. Notice that BINDER from the local site is able to bind but that BINDER from San Francisco is not able to bind. Everyone else from all locations, including the local one, is disabled from doing binds.

Table 119. Restricting bind operations

RLFFUNC	AUTHID	LUNAME	RLFBIND
1	BINDGUY	PUBLIC	
1	NIGHTBND	PUBLIC	
1	(blank)	PUBLIC	N
1	BINDER	SANFRAN	N
1	BINDER	(blank)	

Restricting parallelism modes

The RLST lets you disable parallelism for dynamic queries. You might want to do this if, for example, one mode of parallelism for a particular query performs better or one mode for a particular query does not perform well. For governing query parallelism, remember the following guidelines:

- The resource limit facility only governs query parallelism for dynamic queries when the value of the CURRENT DEGREE special register is 'ANY'.
- To disable all query parallelism for a dynamic query, you need rows in your RLST to cover all possible modes of parallelism for your system. You need one row with RLFFUNC='3' and one row with RLFFUNC='4', and, if Sysplex query parallelism is possible in your system, then you must add an additional row containing RLFFUNC='5'. If parallelism is disabled for a query, the query runs sequentially.
- Qualifying by plan or by package are not separate functions as they are for predictive and reactive governing. If you want to qualify by plan name, the query must be executed from a plan or DB2 will not find the row. If you want to qualify by package name, the query must be executed from a package or DB2 will not find the row.

  If you want to qualify rows by authorization ID only, you can leave all three columns blank: RLFCOLLN, RLFPKG, and RLFPLAN.
- If no entry can be found in your RLST that applies to parallelism, or if your RLST cannot be read, then the resource limit facility does not disable query parallelism.

Table 120. Example RLST to govern query parallelism

RLFFUNC	AUTHID	LUNAME	RLFCOLLN	RLFPKG
3	(blank)	PUBLIC	blank	IOHOG
4	(blank)	PUBLIC	blank	CPUHOG
5	(blank)	PUBLIC	blank	CPUHOG

If the RLST in Table 120 on page 727 is active, it causes the following effects:

- Disables I/O parallelism for all dynamic queries in IOHOG.
- Disables CP parallelism and Sysplex query parallelism for all dynamic queries in CPUHOG.

Chapter 29. Managing DB2 threads

Threads are an important DB2 resource. A thread is a DB2 structure that describes an application's connection and traces its progress. When you install DB2, you choose a maximum number of active allied and database access threads that can be allocated concurrently. Choosing a good number for maximum threads is important to keep applications from queuing and to provide good response time.

When writing an application, you should know when threads are created and terminated and when they can be reused, because thread allocation can be a significant part of the cost in a short transaction.

This chapter provides a general introduction on how DB2 uses threads. It includes the following sections:

- A discussion of how to choose the maximum number of concurrent threads, in "Setting thread limits"
- A description of the steps in creating and terminating an allied thread, in "Allied thread allocation" on page 730
- An explanation of the differences between allied threads and database access threads (DBATs) and a description of how DBATs are created, including how they become active or pooled and how to set performance goals for individual DDF threads, under "Database access threads" on page 735
- Design options for reducing thread allocations and improving performance generally, under "CICS design options" on page 742, "IMS design options" on page 742, "TSO design options" on page 743, and "DB2 QMF design options" on page 744

Setting thread limits

You set the limit of the number of allied and database access threads that can be allocated concurrently using the MAX USERS and MAX REMOTE ACTIVE fields on installation panel DSNTIPE. The combined maximum allowed for MAX USERS and MAX REMOTE ACTIVE is 2000.

Set these values to provide good response time without wasting resources, such as virtual and real storage. The value you specify depends upon your machine size, your work load, and other factors. When specifying values for these fields, consider the following:

- Fewer threads than needed under utilize the processor and cause queuing for threads.
- More threads than needed do not improve the response time. They require more real storage for the additional threads and might cause more paging and, hence, performance degradation.

If virtual storage or real storage is the limiting factor, set MAX USERS and MAX REMOTE ACTIVE according to the available storage. For more information on storage, refer to Part 2 of *DB2 Installation Guide*.

Thread limits for TSO and call attachment: For the TSO and call attachment facilities, you limit the number of threads indirectly by choosing values for the MAX TSO CONNECT and MAX BATCH CONNECT fields of installation panel

© Copyright IBM Corp. 1982, 2008

DSNTIPE. These values limit the number of connections to DB2. The number of threads and connections allowed affects the amount of work that DB2 can process.

Allied thread allocation

This section describes at a high level the steps in allocating an allied thread, and some of the factors related to the performance of those steps. This section does not explain how a database access thread is allocated. For more information on database access threads, see "Database access threads" on page 735.

Step 1: Thread creation

During thread creation with ACQUIRE(ALLOCATE), the resources needed to execute the application are acquired. During thread creation with ACQUIRE(USE), only the thread is created.

The following list shows the main steps in thread creation.

1. Check the maximum number of threads.

   DB2 checks whether the maximum number of active threads, specified as MAX USERS for local threads or MAX REMOTE ACTIVE for remote threads on the Storage Sizes panel (DSNTIPE) when DB2 was installed, has been exceeded. If it has been exceeded, the request waits. The wait for threads is not traced, but the number of requests queued is provided in the performance trace record with IFCID 0073.
2. Check the plan authorization.

   The authorization ID for an application plan is checked in the SYSPLANAUTH catalog table (IFCID 0015). If this check fails, the table SYSUSERAUTH is checked for the SYSADM special privilege.
3. For an application plan, load the control structures associated with the plan.

   The control block for an application plan is divided into sections. The header and directory contain control information; SQL sections contain SQL statements from the application. A copy of the plan's control structure is made for each thread executing the plan. Only the header and directory are loaded when the thread is created.
4. Load the descriptors necessary to process the plan.

   Some of the control structures describe the DB2 table spaces, tables, and indexes used by the application. If ACQUIRE(ALLOCATE) is used, all the descriptors referred to in the plan are loaded now. If the plan is bound with ACQUIRE(USE), they are loaded when SQL statements are executed.

The most relevant factors from a system performance point of view are:

Thread reuse: Thread creation is a significant cost for small and medium transactions. When execution of a transaction is terminated, the thread can sometimes be reused by another transaction using the same plan. For more information on thread reuse, see "Providing for thread reuse" on page 733.

ACQUIRE option of BIND: ACQUIRE(ALLOCATE) causes all the resources referred to in the application to be allocated when the thread is created. ACQUIRE(USE) allocates the resources only when an SQL statement is about to be executed. In general, ACQUIRE(USE) is recommended. However, if most of the SQL is used in every execution of the transaction, ACQUIRE(ALLOCATE) is recommended.

EDM pool size: The size of the EDM pool influences the number of I/Os needed to load the control structures necessary to process the plan or package. To avoid a large number of allocation I/Os, the EDM pool must be large enough to contain the structures that are needed. See "Tuning EDM storage" on page 679 for more information.

Step 2: Resource allocation

Some of the structures necessary to process the statement are stored in 4 KB pages. If they are not already present, those are read into database buffer pool BP0 and copied from there into the EDM pool. If the plan was bound with ACQUIRE(USE), it acquires resources when the statement is about to execute.

1. Load the control structures necessary to process the SQL section.

   If it is not already in the EDM pool, DB2 loads the control structure's section corresponding to this SQL statement.
2. Load structures necessary to process statement.

   Load any structures referred to by this SQL statement that are not already in the EDM pool.
3. Allocate and open data sets.

   When the control structure is loaded, DB2 locks the resources used.

The most important performance factors for resource allocation are the same as the factors for thread creation.

Step 3: SQL statement execution

If the statement resides in a package, the directory and header of the package's control structure is loaded at the time of the first execution of a statement in the package. The control structure for the package is allocated at statement execution time. This is contrasted with the control structures for plans bound with ACQUIRE(ALLOCATE), which are allocated at thread creation time. The header of the plan's control structures is allocated at thread creation time regardless of ACQUIRE(ALLOCATE) or ACQUIRE(USE).

When the package is allocated, DB2 checks authorization using the package authorization cache or the SYSPACKAUTH catalog table. DB2 checks to see that the plan owner has execute authority on the package. On the first execution, the information is not in the cache; therefore, the catalog is used. Thereafter, the cache is used. For more information about package authorization caching, see "Caching authorization IDs for best performance" on page 152.

For dynamic bind, authorization checking also occurs at statement execution time.

A summary record, produced at the end of the statement (IFCID 0058), contains information about each scan that is performed. Included in the record is the following information:

- The number of updated rows
- The number of processed rows
- The number of deleted rows
- The number of examined rows
- The number of pages that are requested through a getpage operation
- The number of rows that are evaluated during the first stage (stage 1) of processing

- The number of rows that are evaluated during the second stage (stage 2) of processing
- The number of getpage requests that are issued to enforce referential constraints
- The number of rows that are deleted or set null to enforce referential constraints
- The number of inserted rows

From a system performance perspective, the most important factor in the performance of SQL statement execution is the size of the database buffer pool. If the buffer pool is large enough, some index and data pages can remain there and can be accessed again without an additional I/O operation. For more information on buffer pools, see Chapter 27, "Tuning DB2 buffer, EDM, RID, and sort pools," on page 665.

Step 4: Commit and thread termination

Commit processing can occur many times while a thread is active. For example, an application program running under the control structure of the thread could issue an explicit COMMIT or SYNCPOINT several times during its execution. When the application program or the thread terminates, an implicit COMMIT or SYNCPOINT is issued.

When a COMMIT or SYNCPOINT is issued from an IMS application running with DB2, the two-phase commit process begins if DB2 resources have been changed since the last commit point. In a CICS or RRSAF application, the two-phase commit process begins only if DB2 resources have changed and a non-DB2 resource has changed within the same commit scope. For more information on the commit process for IMS and CICS applications, see "Multiple system consistency" on page 455.

The significant events that show up in a performance trace of a commit and thread termination operation are as follows:

1. Commit phase 1

   In commit phase 1 (IFCID 0084), DB2 writes an end of phase 1 record to the log (IFCIDs 0032 and 0033). There are two I/Os, one to each active log data set (IFCIDs 0038 and 0039).

2. Commit phase 2

   In commit phase 2 (IFCID 0070), DB2 writes a beginning of phase 2 record to the log. Again, the trace shows two I/Os. Page and row locks (except those protecting the current position of cursors declared with the WITH HOLD option), held to a commit point, are released. An unlock (IFCID 0021) with a requested token of zeros frees any lock for the specified duration. A summary lock record (IFCID 0020) is produced, which gives the maximum number of page locks held and the number of lock escalations. DB2 writes an end of phase 2 record to the log.

   If RELEASE(COMMIT) is used, the following events also occur:

   - Table space locks are released.
   - All the storage used by the thread is freed, including storage for control blocks, CTs and PTs, and working areas.
   - The use counts of the DBDs are decreased by one. If space is needed in the EDM DBD cache, a DBD can be freed when its use count reaches zero.
   - Those table spaces and index spaces with no claimers are made candidates for deferred close. See "Understanding the CLOSE YES and CLOSE NO options" on page 690 for more information on deferred close.

3. Thread termination

   When the thread is terminated, the accounting record is written. It does not report transaction activity that takes place before the thread is created.

   If RELEASE(DEALLOCATE) is used to release table space locks, the DBD use count is decreased, and the thread storage is released.

Variations on thread management

Minor differences exist in the transaction flow in different environments and for SQL statements originating dynamically.

TSO and call attachment facility differences

The TSO attachment facility and call attachment facility (CAF) can be used to request that SQL statements from CICS, IMS, or RRSAF be executed in TSO foreground and batch. The processes differ from CICS or IMS transactions in that:

- There is no sign-on. The user is identified when the TSO address space is connected.
- Commit requires only a single phase and only one I/O operation to each log. Single phase commit records are IFCID 0088 and 0089.
- Threads cannot be reused because the thread is allocated to the user address space.

Thread management for Resource Recovery Services attachment facility (RRSAF)

With RRSAF, you have sign-on capabilities, the ability to reuse threads, and the ability to coordinate commit processing across different resource managers. For more information, see Part 6 of *DB2 Application Programming and SQL Guide*.

Differences for SQL under DB2 QMF

DB2 QMF uses CAF to create a thread when a request for work, such as a SELECT statement, is issued. A thread is maintained until the end of the session only if the requester and the server reside in different DB2 subsystems. If the requester and the server are both in the local DB2 subsystem, the thread is not maintained.

For more information about DB2 QMF connections, see *DB2 Query Management Facility: Installing and Managing DB2 QMF for TSO/CICS*.

Providing for thread reuse

In general, you want transactions to reuse threads when transaction volume is high and the cost of creating threads is significant, but thread reuse is also useful for a lower volume of priority transactions. For a transaction of five to ten SQL statements (10 I/O operations), the cost of thread creation can be 10% of the processor cost. But the steps needed to reuse threads can incur costs of their own.

Later in this chapter, the following sections cover thread reuse for specific situations:

- “Reusing threads for remote connections” on page 739 provides information on the conditions for thread reuse for database access threads.
- “CICS design options” on page 742 tells how to write CICS transactions to reuse threads.
- “IMS design options” on page 742 tells how to write IMS transactions to reuse threads.

Bind options for thread reuse

In DB2, you can prepare allied threads for reuse by binding the plan with the ACQUIRE(USE) and RELEASE(DEALLOCATE) options; otherwise, the allocation cost is not eliminated but only slightly reduced. Be aware of the following effects:

- ACQUIRE(ALLOCATE) acquires all resources needed by the plan, including locks, when the thread is created; ACQUIRE(USE) acquires resources only when they are needed to execute a particular SQL statement. If most of the SQL statements in the plan are executed whenever the plan is executed, ACQUIRE(ALLOCATE) costs less. If only a few of the SQL statements are likely to be executed, ACQUIRE(USE) costs less and improves concurrency. But with thread reuse, if most of your SQL statements eventually get issued, ACQUIRE(USE) might not be as much of an improvement.
- RELEASE(DEALLOCATE) does not free cursor tables (SKCTs) at a commit point; hence, the cursor table could grow as large as the plan. If you are using created temporary tables, the logical work file space is not released until the thread is deallocated. Thus, many uses of the same created temporary table do not cause reallocation of the logical work files, but be careful about holding onto this resource for long periods of time if you do not plan to use it.

Using reports to tell when threads were reused

The NORMAL TERM., ABNORMAL TERM., and IN DOUBT sections of the OMEGAMON accounting report, shown in Figure 73, can help you identify, by plan, when threads were reused. In the figure:

- NEW USER (**A**) tells how many threads were not terminated at the end of the previous transaction or query, and hence reused.
- DEALLOCATION (**B**) tells how many threads were terminated at the end of the query or transaction.
- APPL. PROGR. END (**C**) groups all the other reasons for accounting. Since the agent did not abend, these are considered normal terminations.

This technique is accurate in IMS but not in CICS, where threads are reused frequently by the same user. For CICS, also consider looking at the number of commits and aborts per thread. For CICS:

- NEW USER (**A**) is thread reuse with a different authorization ID or transaction code.
- RESIGN-ON (**D**) is thread reuse with the same authorization ID.

```
NORMAL TERM.      TOTAL  ABNORMAL TERM.      TOTAL  IN DOUBT            TOTAL
----------------  -----  ------------------  -----  -----------------  ------

NEW USER  A           0  APPL.PROGR. ABEND       0  APPL.PGM. ABEND          0
DEALLOCATION  B       0  END OF MEMORY           0  END OF MEMORY           0
APPL.PROGR. END  C    0  RESOL.IN DOUBT          0  END OF TASK             0
RESIGNON   D          0  CANCEL FORCE            0  CANCEL FORCE            0
DBAT INACTIVE         0
IN2 INACTIVE        193
RRS COMMIT            0
END U. THRESH         0
BLOCK STORAGE         0
STALENESS             0
```

Figure 73. OMEGAMON accounting report - information about thread termination

Database access threads

This section describes the following topics:

- "Using allied threads and database access threads"
- "Setting thread limits for database access threads"
- "Using threads in INACTIVE MODE for DRDA-only connections" on page 736
- "Using threads with private-protocol connections" on page 738
- "Reusing threads for remote connections" on page 739
- "Using z/OS workload management to set performance objectives" on page 739

For information on performance considerations for distributed processing, see "Tuning distributed applications" on page 1004.

Using allied threads and database access threads

Database access threads are created to access data at a DB2 server on behalf of a requester using either DRDA or DB2 private protocol. A database access thread is created when an SQL request is received from the requester. Allied threads perform work at a requesting DB2.

Database access threads differ from allied threads in the following ways:

- Database access threads have two modes of processing: ACTIVE MODE and INACTIVE MODE. These modes are controlled by setting the DDF THREADS field on the installation panel DSNTIPR.
  - ACTIVE MODE: When the value of DDF THREADS is ACTIVE, a database access thread is always active from initial creation to termination.
  - INACTIVE MODE: When the value of DDF THREADS is INACTIVE, a database access thread can be active or pooled. When a database access thread in INACTIVE MODE is active, it is processing requests from client connections within units of work. When a database access thread is pooled, it is waiting for the next request from a client to start a new unit of work.
- Database access threads run in enclave SRB mode.
- Only when in INACTIVE MODE, a database access thread is terminated after it has processed 200 units of work or after the database access thread has been idle in the pool for the amount of time specified in the POOL THREAD TIMEOUT field on the installation panel DSNTIP5. This termination does not prevent another database access thread from being created to meet processing demand, as long as the value of the MAX REMOTE ACTIVE field on panel DSNTIPE has not been reached.

Recommendation: Use INACTIVE MODE threads instead of ACTIVE MODE threads if you can.

Setting thread limits for database access threads

When you install DB2, you choose a maximum number of active threads that can be allocated concurrently; the MAX USERS field on panel DSNTIPE represents the maximum number of allied threads, and the MAX REMOTE ACTIVE field on panel DSNTIPE represents the maximum number of database access threads. Together, the values you specify for these fields cannot exceed 1999.

In the MAX REMOTE CONNECTED field of panel DSNTIPE, you can specify up to 150 000 as the maximum number concurrent remote connections that can concurrently exist within DB2. This upper limit is only obtained if you specify the

recommended value INACTIVE for the DDF THREADS field of installation panel DSNTIPR. Figure 74 illustrates the relationship among the number of active threads in the system and the total number of connections.

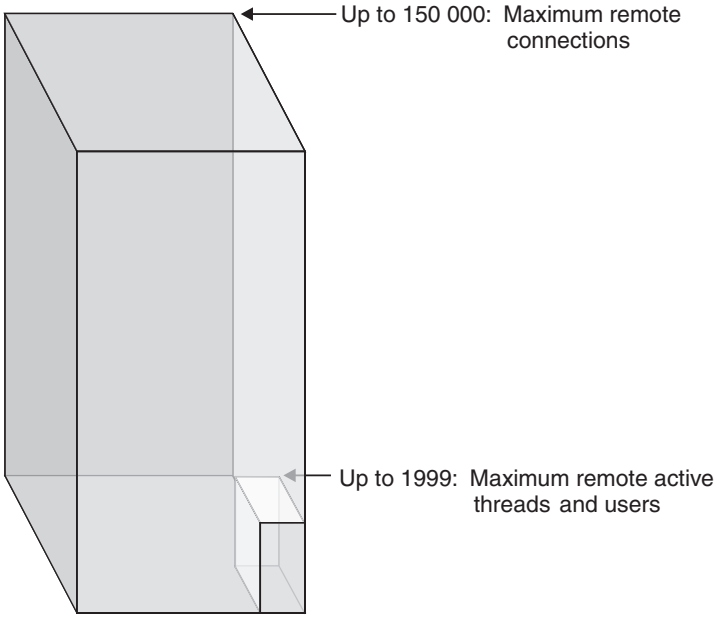


Figure 74. Relationship between active threads and maximum number of connections.

Using threads in INACTIVE MODE for DRDA-only connections

When the value of DDF THREADS is INACTIVE, DRDA connections with a client cause a pooling behavior in database access threads. A different database access behavior is used when private-protocol connections are involved. For information about private-protocol connections and database access threads, see "Using threads with private-protocol connections" on page 738.

A database access thread that is not currently processing a unit of work is called a pooled thread, and it is disconnected. DB2 always tries to pool database access threads, but in some cases cannot do so. The conditions listed in Table 121 determine if a thread can be pooled.

Table 121. Requirements for pooled threads

If there is...	Thread can be pooled?
A DRDA hop to another location	Yes
A package that is bound with RELEASE(COMMIT)	Yes
A package that is bound with RELEASE(DEALLOCATE)	Yes
A held cursor, a held LOB locator, or a package bound with KEEPDYNAMIC(YES)	No
A declared temporary table that is active (the table was not explicitly dropped through the DROP TABLE statement or the ON COMMIT DROP TABLE clause on the DECLARE GLOBAL TEMPORARY TABLE statement)	No

When the conditions listed in Table 121 are true, the thread can be pooled when a COMMIT is issued. After a ROLLBACK, a thread can be pooled even if it had open cursors defined WITH HOLD or a held LOB locator because ROLLBACK

closes all cursors and LOB locators. ROLLBACK is also the only way to use the KEEPDYNAMIC(YES) bind option to clear information about a dynamic SQL statement.

Understanding the advantages of database access threads in INACTIVE MODE

Letting threads be pooled has the following advantages:

- You can leave an application that is running on a workstation connected to DB2 from the time the application is first activated until the workstation is shut down and thus avoid the delay of repeated connections.
- DB2 can support a larger number of DDF connections (150 000 instead of 4999).
- Less storage is used for each DDF connection. (A connection uses significantly less storage than a database access thread.)
- You get an accounting trace record each time that a thread is pooled rather than once for the entire time that you are connected. When a pooled thread becomes active, the accounting fields for that thread are re-initialized. As a result, the accounting record contains information about active threads only, which makes it easier to study how distributed applications are performing. If a pooled mode thread remains active because of KeepDynamic sections, an accounting record is still written.

  Exception: If you employ account record accumulation, an accounting trace is not produced each time that a thread becomes pooled. Accounting records can be rolled up by concatenated combinations of the following values:
  - End user ID
  - End transaction name
  - End user workstation name
- Each time that a thread is pooled, workload manager resets the information that it maintains on that thread. The next time that thread is activated, workload manager begins managing to the goals that you have set for the transactions that run in that service class. If you use multiple performance periods, it is possible to favor short-running units of work that use fewer resources while giving fewer resources over time to long running units of work. See “Establishing performance periods for DDF threads” on page 741 for more information.
- You can use response time goals, which is not recommended when using ACTIVE MODE threads.
- INACTIVE MODE threads can better take advantage of the ability to time out idle active threads, as described in “Timing out idle active threads.”
- Thread reuse lets DDF use a small pool of database access threads to support a large group of network clients.
- The response times reported by RMF include periods between requests and within the same unit of work. These times are shown as idle delays.

Enabling threads to be pooled

You must specify INACTIVE on the DDF THREADS field of installation panel DSNTIPR to allow threads to be pooled.

Timing out idle active threads

Active server threads that have remained idle for a specified period of time (in seconds) can be canceled by DB2. When you install DB2, you choose a maximum IDLE THREAD TIMEOUT period, from 0 to 9999 seconds. The timeout period is an approximation. If a server thread has been waiting for a request from the requesting site for this period of time, it is canceled unless the thread is currently pooled or in doubt thread. A value of 0, the default, means that the server threads cannot be canceled because of an idle thread timeout.

Recommendation: Use the default option with the option ACTIVE for the DDF THREADS field on DSNTIPR. If you specify a timeout interval with ACTIVE, an application would have to start its next unit of work within the timeout period specification, or risk being canceled.

TCP/IP keep_alive interval for the DB2 subsystem: For TCP/IP connections, it is a good idea to specify the IDLE THREAD TIMEOUT value in conjunction with a TCP/IP keep_alive interval of 5 minutes or less to make sure that resources are not locked for a long time when a network outage occurs. It is recommended that you override the TCP/IP stack keep_alive interval on a single DB2 subsystem by specifying a numeric value in the field TCP/IP KEEPALIVE on installation panel DSNTIPS.

Using threads with private-protocol connections

Database access threads with private-protocol connections behave differently than threads with DRDA-only connections. When a database access thread is associated with a private-protocol connection, whether inbound, outbound, or both, the database access thread remains with the connection or connections. During periods of inactivity, DB2 reduces the memory footprint of threads until the next unit of work begins.

The MAX INACTIVE DBATS setting determines whether DB2 will reduce the memory footprint of a private-protocol connection that involves a database access thread. When a private-protocol connection ends a unit of work, DB2 first compares the number of current inactive database access threads to the value that is specified for your installation for MAX INACTIVE DBATS. Based on these values, DB2 either makes the thread inactive or allows it to remain active:

- If the current number of inactive database access threads is below the value in MAX INACTIVE DBATS, the thread becomes inactive. It cannot be used by another connection.
- If the current number of inactive database access threads meets or exceeds the value in MAX INACTIVE DBATS, the thread remains active. However, too many active threads (that is, more than MAX REMOTE ACTIVE) can cause the thread and its connection to be terminated.

To limit the number of inactive database access threads that can be created, specify a value in the MAX INACTIVE DBATS field of installation panel DSNTIPR. The default is 0, which means that any database access thread that involves a private-protocol-connection remains active.

While using a private-protocol connection, DB2 always tries to make threads inactive, but in some cases cannot do so. If MAX INACTIVE DBATS is greater than 0 and the value has not been exceeded, the conditions listed in Table 122 determine if a thread can be inactive.

Table 122. Requirements for inactive threads

If there is...	Thread can be inactive?
A hop to another location	Yes
A package that is bound with RELEASE(COMMIT)	Yes
A package that is bound with RELEASE(DEALLOCATE)	No
A held cursor, a held LOB locator, or a package bound with KEEPDYNAMIC(YES)	No

Table 122. Requirements for inactive threads (continued)

If there is...	Thread can be inactive?
A declared temporary table that is active (the table was not explicitly dropped through the DROP TABLE statement)	No

Reusing threads for remote connections

The cost to create a thread can be significant and, as described in "Providing for thread reuse" on page 733, reusing threads is a way to avoid that cost. DB2 UDB for z/OS can reuse threads at the requester and at the server. At the requester, a thread can be reused for an application that uses the CICS, IMS, or RRS attachment facility as described later in this chapter. As a server, DB2 can assign that connection to a pooled database access thread. Those threads can be shared and reused by thousands of client connections, which lets DB2 support very large client networks at minimal cost. (Inactive threads are only eligible to be reused by the same connection.)

If your server is not DB2 UDB for z/OS, or some other server that can reuse threads, then reusing threads for your requesting CICS, IMS, or RRS applications is not a benefit for distributed access. Thread reuse occurs when sign-on occurs with a new authorization ID. If that request is bound for a server that does not support thread reuse, that change in the sign-on ID causes the connection between the requester and server to be released so that it can be rebuilt again for the new ID.

Using z/OS workload management to set performance objectives

z/OS supports enclave system request blocks (SRBs). A z/OS enclave lets each thread have its own performance objective. Using z/OS workload mangement (WLM) support, you can establish z/OS performance objectives for individual DDF server threads. For details on using workload management, see *z/OS MVS Planning: Workload Management*.

The z/OS performance objective of the DDF address space does not govern the performance objective of the user thread. As described in "z/OS performance options for DB2" on page 711, you should assign the DDF address space to a z/OS performance objective that is similar to the DB2 database services address space (*ssnm*DBM1). The z/OS performance objective of the DDF address space determines how quickly DB2 is able to perform operations associated with managing the distributed DB2 work load, such as adding new users or removing users that have terminated their connections. This performance objective should be a service class with a single velocity goal. This performance objective is assigned by modifying the WLM Classification Rules for started tasks (STC).

Classifying DDF threads

You can classify DDF threads by, among other things, authorization ID and stored procedure name. The stored procedure name is only used as a classification if the first statement issued by the client after the CONNECT is an SQL CALL statement. Use the WLM administrative application to define the service classes you want z/OS to manage. These service classes are associated with performance objectives. When a WLM-established stored procedure call originates locally, it inherits the performance objective of the caller, such as TSO or CICS.

Important: If you do not classify your DDF transactions into service classes, they are assigned to the default class, the *discretionary* class, which is at a very low priority.

Classification attributes: Each of the WLM classification attributes has a two or three character abbreviation that you can use when entering the attribute on the WLM menus. The following WLM classification attributes pertain to DB2 DDF threads:

AI Accounting information. The value of the DB2 accounting string associated with the DDF server thread, described by QMDAAINF in the DSNDQMDA mapping macro. WLM imposes a maximum length of 143 bytes for accounting information.

CI The DB2 correlation ID of the DDF server thread, described by QWHCCV in the DSNDQWHC mapping macro.

CN The DB2 collection name of the *first* SQL package accessed by the DRDA requester in the unit of work.

LU The VTAM LUNAME of the system that issued the SQL request.

NET The VTAM NETID of the system that issued the SQL request.

PC Process name. This attribute can be used to classify the application name or the transaction name. The value is defined by QWHCEUTX in the DSNDQWHC mapping macro.

PK The name of the **first** DB2 package accessed by the DRDA requester in the unit of work.

PN The DB2 plan name associated with the DDF server thread. For DB2 private protocol requesters and DB2 DRDA requesters that are at Version 3 or subsequent releases, this is the DB2 plan name of the requesting application. For other DRDA requesters, use 'DISTSERV' for PN.

PR Stored procedure name. This classification only applies if the first SQL statement from the client is a CALL statement.

SI Subsystem instance. The DB2 server's z/OS subsystem name.

SPM Subsystem parameter. This qualifier has a maximum length of 255 bytes. The first 16 bytes contain the client's user ID. The next 18 bytes contain the client's workstation name. The remaining 221 bytes are reserved.

Important: If the length of the client's user ID is less than 16 bytes, uses blanks after the user ID to pad the length. If the length of the client's workstation name is less than 18 bytes, uses blanks after the workstation name to pad the length.

SSC Subsystem collection name. When the DB2 subsystem is a member of a DB2 data sharing group, this attribute can be used to classify the data sharing group name. The value is defined by QWHADSGN in the DSNDQWHA mapping macro.

UI User ID. The DDF server thread's primary authorization ID, after inbound name translation.

Figure 75 on page 741 shows how you can associate DDF threads with service classes.

```
  Subsystem-Type  Xref  Notes  Options  Help
-------------------------------------------------------------------------------
                Create Rules for the Subsystem Type        Row 1 to 5 of 5

Subsystem Type . . . . . . . . DDF    (Required)
Description  . . . . . . . . . Distributed DB2
Fold qualifier names?  . . . . Y  (Y or N)

Enter one or more action codes: A=After  B=Before  C=Copy  D=Delete
M=Move I=Insert rule IS=Insert Sub-rule  R=Repeat

          -------Qualifier-------------            -------Class--------
Action    Type       Name     Start                Service     Report
                                           DEFAULTS: PRDBATCH    ________
 ____  1  SI         DB2P     ___                  PRDBATCH    ________
 ____  2    CN         ONLINE   ___                 PRDONLIN    ________
 ____  2    PRC        PAYPROC  ___                 PRDONLIN    ________
 ____  2    UI         SYSADM   ___                 PRDONLIN    ________
 ____  2    PK         QMFOS2   ___                 PRDQUERY    ________
 ____  1  SI         DB2T     ___                  TESTUSER    ________
 ____  2    PR         PAYPROCT ___                 TESTPAYR    ________
***************************** BOTTOM OF DATA ******************************
```

Figure 75. Classifying DDF threads using z/OS workload management. You assign performance goals to service classes using the services classes menu of WLM.

In Figure 75, the following classifications are shown:

- All DB2P applications accessing their first SQL package in the collection ONLINE are in service class PRDONLIN.
- All DB2P applications that call stored procedure PAYPROC first are in service class PRDONLIN.
- All work performed by DB2P user SYSADM is in service class PRDONLIN.
- Users other than SYSADM that run the DB2P PACKAGE QMFOS2 are in the PRDQUERY class. (The QMFOS2 package is not in collection ONLINE.
- All other work on the production system is in service class PRBBATCH.
- All users of the test DB2 system are assigned to the TESTUSER class except for work that first calls stored procedure PAYPROCT, which is in service class TESTPAYR.

Establishing performance periods for DDF threads

You can establish performance periods for DDF threads, including threads that run in the WLM-established stored procedures address space. By establishing multiple performance periods, you can cause the thread's performance objectives to change based upon the thread's processor consumption. Thus, a long-running unit of work can move down the priority order and let short-running transactions get in and out at a higher priority.

To design performance strategies for these threads, take into account the events that cause a DDF thread to reset its z/OS performance period. The z/OS performance period is reset by terminating the z/OS enclave for the thread and creating a new z/OS enclave for the thread, as described in "Monitoring distributed processing with RMF" on page 1017.

Because threads that are always active do not terminate the enclave and thus do not reset the performance period to period 1, a long-running thread always ends up in the last performance period. Any new business units of work that use that thread will suffer the performance consequences. This makes performance periods unattractive for long-running threads. For always active threads, therefore, use velocity goals and use a single-period service class.

Basic procedure for establishing performance objectives

To establish performance objectives for DDF threads and the related address spaces, use the following steps:

1. Create a WLM service definition that assigns service classes to the DDF threads under subsystem type DDF and to the DDF address space under subsystem type STC.
2. Install the service definition using the WLM menus and activate a policy (VARY WLM,POLICY=*policy*).

Threads are assigned a service class by the classification rules in the active WLM service policy. Each service class period has a performance objective (goal), and WLM raises or lowers that period's access to system resources as needed to meet the specified goal. For example, the goal might be "application APPL8 should run in less than 3 seconds of elapsed time 90% of the time".

No matter what your service class goals are, a request to start an address space might time out, depending on the timeout value that is specified in the TIMEOUT VALUE field of installation DSNTIPX. If the timeout value is too small, you might need to increase it to account for a busy system.

CICS design options

The CICS attachment facility provides a multithread connection to DB2 to allow you to operate DB2 with CICS. Threads allow each CICS application transaction or DB2 command to access DB2 resources. A thread runs under its own subtask task control block (TCB).

When a transaction needs a thread, an existing thread can be reused or a new thread can be created. If no existing thread is available, and if the maximum number of threads has not been reached, a thread is created. Use the CICS resource definition online (RDO) to tune the CICS attachment facility and define the characteristics of your threads. For more information, see:

- *CICS Transaction Server for OS/390 Resource Definition Guide* for information about RDO
- *CICS Transaction Server for z/OS DB2 Guide* for information about DB2 performance considerations and setup of the CICS attachment facility

IMS design options

The IMS attachment facility gives you the following design options:

- Control the number of IMS regions connected to DB2. For IMS, this is also the maximum number of concurrent threads.
- Optimize the number of concurrent threads used by IMS.

  A dependent region with a subsystem member (SSM) that is not empty is connected to DB2 at start up time. Regions with a null SSM cannot create a thread to DB2. A thread to DB2 is created at the first execution of an SQL statement in an IMS application schedule; it is terminated when the application terminates.

  The maximum number of concurrent threads used by IMS can be controlled by the number of IMS regions that can connect to DB2 by transaction class assignments. You can control the number by doing the following:

  – Minimize the number of regions needing a thread by the way in which you assign applications to regions.

– Provide an empty SSM member for regions that will not connect to DB2.

- Provide efficient thread reuse for high volume transactions.

  Thread creation and termination is a significant cost in IMS transactions. IMS transactions identified as wait for input (WFI) can reuse threads: they create a thread at the first execution of an SQL statement and reuse it until the region is terminated. In general, though, use WFI only for transactions that reach a region utilization of at least 75%.

  Some degree of thread reuse can also be achieved with IMS class scheduling, queuing, and a PROCLIM count greater than one. IMS Fast Path (IFP) dependent regions always reuse the DB2 thread.

TSO design options

You can tune your TSO attachment facility by choosing values for the following parameters on the Storage Sizes installation panel (DSNTIPE):

MAX TSO CONNECT The maximum number of TSO foreground connections (including DB2I, QMF, and foreground applications)

MAX BATCH CONNECT The maximum number of TSO background connections (including batch jobs and utilities)

Because DB2 must be stopped to set new values, consider setting a higher MAX BATCH CONNECT for batch periods. The statistics record (IFCID 0001) provides information on the create thread queue. The OMEGAMON statistics report (in Figure 76 on page 744) shows that information under the SUBSYSTEM SERVICES section.

For TSO or batch environments, having 1% of the requests queued is probably a good number to aim for by adjusting the MAX USERS value of installation panel DSNTIPE. Queuing at create thread time is not desirable in the CICS and IMS environments. If you are running IMS or CICS in the same DB2 subsystem as TSO and batch, use MAX BATCH CONNECT and MAX TSO CONNECT to limit the number of threads taken by the TSO and batch environments. The goal is to allow enough threads for CICS and IMS so that their threads do not queue. To determine the number of allied threads queued, see the QUEUED AT CREATE THREAD field (**A**) of the OMEGAMON statistics report.

```
SUBSYSTEM SERVICES           QUANTITY
---------------------------  --------
IDENTIFY                     30757.00
CREATE THREAD                30889.00
SIGNON                           0.00
TERMINATE                    61661.00
ROLLBACK                       644.00

COMMIT PHASE 1                   0.00
COMMIT PHASE 2                   0.00
READ ONLY COMMIT                 0.00

UNITS OF RECOVERY INDOUBT        0.00
UNITS OF REC.INDBT RESOLVED      0.00

SYNCHS(SINGLE PHASE COMMIT)  30265.00
QUEUED AT CREATE THREAD  A       0.00
SUBSYSTEM ALLIED MEMORY EOT      1.00
SUBSYSTEM ALLIED MEMORY EOM      0.00
SYSTEM EVENT CHECKPOINT          0.00
```

Figure 76. Thread queuing in the OMEGAMON statistics report

DB2 QMF design options

DB2 QMF has the following significant performance options:

- The DSQSIROW parameter of the ISPSTART command
- SPACE parameter of the user QMF profile (Q.PROFILES)
- QMF region size and the spill file attributes
- TRACE parameter of the user QMF profile (Q.PROFILES)

For more information on these aspects of DB2 QMF and how they affect performance, see *DB2 Query Management Facility: Installing and Managing DB2 QMF for TSO/CICS*.

Chapter 30. Tuning your queries

The information under this heading, up to the end of this chapter, is Product-sensitive Programming Interface and Associated Guidance Information, as defined in "Notices" on page 1301.

This chapter tells you how to improve the performance of your queries. It begins with "General tips and questions."

For more detailed information and suggestions, see:

- "Writing efficient predicates" on page 749
- "General rules about predicate evaluation" on page 753
- "Using host variables efficiently" on page 774
- "Writing efficient subqueries" on page 780
- "Using scrollable cursors efficiently" on page 785
- "Writing efficient queries on tables with data-partitioned secondary indexes" on page 786

If you still have performance problems after you have tried the suggestions in these sections, you can use other, more risky techniques. See "Special techniques to influence access path selection" on page 788 for information.

General tips and questions

Recommendation: If you have a query that is performing poorly, first go over the following checklist to see that you have not overlooked some of the basics:

- "Is the query coded as simply as possible?"
- "Are all predicates coded correctly?"
- "Are there subqueries in your query?" on page 746
- "Does your query involve aggregate functions?" on page 747
- "Do you have an input variable in the predicate of an SQL query?" on page 748
- "Do you have a problem with column correlation?" on page 748
- "Can your query be written to use a noncolumn expression?" on page 748
- "Can materialized query tables help your query performance?" on page 748
- "Does the query contain encrypted data?" on page 749

Is the query coded as simply as possible?

Make sure the SQL query is coded as simply and efficiently as possible. Make sure that no unused columns are selected and that there is no unneeded ORDER BY or GROUP BY.

Are all predicates coded correctly?

Indexable predicates: Make sure all the predicates that you think should be indexable are coded so that they can be indexable. Refer to Table 124 on page 754 to see which predicates are indexable and which are not.

Unintentionally redundant or unnecessary predicates: Try to remove any predicates that are unintentionally redundant or not needed; they can slow down performance.

© Copyright IBM Corp. 1982, 2008

Declared lengths of host variables: For string comparisons other than equal comparisons, ensure that the declared length of a host variable is less than or equal to the length attribute of the table column that it is compared to. For languages in which character strings are nul-terminated, the string length can be less than or equal to the column length plus 1. If the declared length of the host variable is greater than the column length, the predicate is stage 1 but cannot be a matching predicate for an index scan.

For example, assume that a host variable and an SQL column are defined as follows:

C language declaration	SQL definition
char string_hv[15]	STRING_COL CHAR(12)

A predicate such as `WHERE STRING_COL > :string_hv` is not a matching predicate for an index scan because the length of string_hv is greater than the length of STRING_COL. One way to avoid an inefficient predicate using character host variables is to declare the host variable with a length that is less than or equal to the column length:

```
char string_hv[12]
```

Because this is a C language example, the host variable length could be 1 byte greater than the column length:

```
char string_hv[13]
```

For numeric comparisons, a comparison between a DECIMAL column and a float or real host variable is stage 2 if the precision of the DECIMAL column is greater than 15. For example, assume that a host variable and an SQL column are defined as follows:

C language declaration	SQL definition
float float_hv	DECIMAL_COL DECIMAL(16,2)

A predicate such as `WHERE DECIMAL_COL = :float_hv` is not a matching predicate for an index scan because the length of DECIMAL_COL is greater than 15. However, if DECIMAL_COL is defined as DECIMAL(15,2), the predicate is stage 1 and indexable.

Are there subqueries in your query?

If your query uses subqueries, see "Writing efficient subqueries" on page 780 to understand how DB2 executes subqueries. There are no absolute rules to follow when deciding how or whether to code a subquery. But these are general guidelines:

- If efficient indexes are available on the tables in the subquery, then a correlated subquery is likely to be the most efficient kind of subquery.
- If no efficient indexes are available on the tables in the subquery, then a noncorrelated subquery would be likely to perform better.
- If multiple subqueries are in any parent query, make sure that the subqueries are ordered in the most efficient manner.

Example: Assume that MAIN_TABLE has 1000 rows:

```
SELECT * FROM MAIN_TABLE
  WHERE TYPE IN (subquery 1) AND
        PARTS IN (subquery 2);
```

Assuming that subquery 1 and subquery 2 are the same type of subquery (either correlated or noncorrelated) and the subqueries are stage 2, DB2 evaluates the subquery predicates in the order they appear in the WHERE clause. Subquery 1 rejects 10% of the total rows, and subquery 2 rejects 80% of the total rows.

The predicate in subquery 1 (which is referred to as P1) is evaluated 1000 times, and the predicate in subquery 2 (which is referred to as P2) is evaluated 900 times, for a total of 1900 predicate checks. However, if the order of the subquery predicates is reversed, P2 is evaluated 1000 times, but P1 is evaluated only 200 times, for a total of 1200 predicate checks.

Coding P2 before P1 appears to be more efficient if P1 and P2 take an equal amount of time to execute. However, if P1 is 100 times faster to evaluate than P2, then coding subquery 1 first might be advisable. If you notice a performance degradation, consider reordering the subqueries and monitoring the results. Consult "Writing efficient subqueries" on page 780 to help you understand what factors make one subquery run more slowly than another.

If you are unsure, run EXPLAIN on the query with both a correlated and a noncorrelated subquery. By examining the EXPLAIN output and understanding your data distribution and SQL statements, you should be able to determine which form is more efficient.

This general principle can apply to all types of predicates. However, because subquery predicates can potentially be thousands of times more processor- and I/O-intensive than all other predicates, the order of subquery predicates is particularly important.

Regardless of coding order, DB2 performs noncorrelated subquery predicates before correlated subquery predicates, unless the subquery is transformed into a join.

Refer to "DB2 predicate manipulation" on page 770 to see in what order DB2 will evaluate predicates and when you can control the evaluation order.

Does your query involve aggregate functions?

If your query involves aggregate functions, make sure that they are coded as simply as possible; this increases the chances that they will be evaluated when the data is retrieved, rather than afterward. In general, a aggregate function performs best when evaluated during data access and next best when evaluated during DB2 sort. Least preferable is to have a aggregate function evaluated after the data has been retrieved. Refer to "When are aggregate functions evaluated? (COLUMN_FN_EVAL)" on page 946 for help in using EXPLAIN to get the information you need.

For aggregate functions to be evaluated during data retrieval, the following conditions must be met for all aggregate functions in the query:

- No sort is needed for GROUP BY. Check this in the EXPLAIN output.
- No stage 2 (residual) predicates exist. Check this in your application.
- No distinct set functions exist, such as COUNT(DISTINCT C1).
- If the query is a join, all set functions must be on the last table joined. Check this by looking at the EXPLAIN output.
- All aggregate functions must be on single columns with no arithmetic expressions.

- The aggregate function is not one of the following aggregate functions:
  - STDDEV
  - STDDEV_SAMP
  - VAR
  - VAR_SAMP

If your query involves the functions MAX or MIN, refer to "One-fetch access (ACCESSTYPE=I1)" on page 953 to see whether your query could take advantage of that method.

Do you have an input variable in the predicate of an SQL query?

When host variables or parameter markers are used in a query, the actual values are not known when you bind the package or plan that contains the query. DB2 therefore uses a default filter factor to determine the best access path for an SQL statement. If that access path proves to be inefficient, you can do several things to obtain a better access path.

See "Using host variables efficiently" on page 774 for more information.

Do you have a problem with column correlation?

Two columns in a table are said to be correlated if the values in the columns do not vary independently.

DB2 might not determine the best access path when your queries include correlated columns. If you think you have a problem with column correlation, see "Column correlation" on page 766 for ideas on what to do about it.

Can your query be written to use a noncolumn expression?

The following predicate combines a column, SALARY, with values that are not from columns on one side of the operator:

```
WHERE SALARY + (:hv1 * SALARY) > 50000
```

If you rewrite the predicate in the following way, DB2 can evaluate it more efficiently:

```
WHERE SALARY > 50000/(1 + :hv1)
```

In the second form, the column is by itself on one side of the operator, and all the other values are on the other side of the operator. The expression on the right is called a *noncolumn expression*. DB2 can evaluate many predicates with noncolumn expressions at an earlier stage of processing called *stage 1*, so the queries take less time to run.

For more information on noncolumn expressions and stage 1 processing, see "Properties of predicates" on page 749.

Can materialized query tables help your query performance?

Dynamic queries that operate on very large amounts of data and involve multiple joins might take a long time to run. One way to improve the performance of these queries is to generate the results of all or parts of the queries in advance, and store the results in *materialized query tables*.

Materialized query tables are user-created tables. Depending on how the tables are defined, they are user-maintained or system-maintained. If you have set subsystem parameters or an application sets special registers to tell DB2 to use materialized query tables, when DB2 executes a dynamic query, DB2 uses the contents of applicable materialized query tables if DB2 finds a performance advantage to doing so.

For information about materialized query tables, see Chapter 32, "Using materialized query tables," on page 881.

Does the query contain encrypted data?

Encryption and decryption can degrade the performance of some queries. However, you can lessen the performance impact of encryption and decryption by writing your queries carefully and designing your database with encrypted data in mind. For more information about avoiding performance degradation while using encrypted data, see "Performance recommendations for data encryption" on page 212.

Writing efficient predicates

Definition: *Predicates* are found in the clauses WHERE, HAVING or ON of SQL statements; they describe attributes of data. They are usually based on the columns of a table and either qualify rows (through an index) or reject rows (returned by a scan) when the table is accessed. The resulting qualified or rejected rows are independent of the access path chosen for that table.

Example: The following query has three predicates: an equal predicate on C1, a BETWEEN predicate on C2, and a LIKE predicate on C3.

```
SELECT * FROM T1
  WHERE C1 = 10 AND
        C2 BETWEEN 10 AND 20 AND
        C3 NOT LIKE 'A%'
```

Effect on access paths: This section explains the effect of predicates on access paths. Because SQL allows you to express the same query in different ways, knowing how predicates affect path selection helps you write queries that access data efficiently.

This section describes:
- "Properties of predicates"
- "General rules about predicate evaluation" on page 753
- "Predicate filter factors" on page 760
- "DB2 predicate manipulation" on page 770
- "Column correlation" on page 766

Properties of predicates

Predicates in a HAVING clause are not used when selecting access paths; hence, in this section the term 'predicate' means a predicate after WHERE or ON.

A predicate influences the selection of an access path because of:
- Its **type**, as described in "Predicate types" on page 750
- Whether it is **indexable**, as described in "Indexable and nonindexable predicates" on page 751
- Whether it is **stage 1** or **stage 2**

- Whether it contains a ROWID column, as described in "Is direct row access possible? (PRIMARY_ACCESSTYPE = D)" on page 942

There are special considerations for "Predicates in the ON clause" on page 752.

Predicate definitions: Predicates are identified as:

Simple or compound
A *compound* predicate is the result of two predicates, whether simple or compound, connected together by AND or OR Boolean operators. All others are *simple*.

Local or join
Local predicates reference only one table. They are local to the table and restrict the number of rows returned for that table. *Join predicates* involve more than one table or correlated reference. They determine the way rows are joined from two or more tables. For examples of their use, see "Interpreting access to two or more tables (join)" on page 955.

Boolean term
Any predicate that is not contained by a compound OR predicate structure is a *Boolean term*. If a Boolean term is evaluated false for a particular row, the whole WHERE clause is evaluated false for that row.

Predicate types

The type of a predicate depends on its operator or syntax. The type determines what type of processing and filtering occurs when the predicate is evaluated. Table 123 shows the different predicate types.

Table 123. Definitions and examples of predicate types

Type	Definition	Example
Subquery	Any predicate that includes another SELECT statement.	C1 IN (SELECT C10 FROM TABLE1)
Equal	Any predicate that is not a subquery predicate and has an equal operator and no NOT operator. Also included are predicates of the form C1 IS NULL and C IS NOT DISTINCT FROM.	C1=100
Range	Any predicate that is not a subquery predicate and has an operator in the following list: >, >=, <, <=, LIKE, or BETWEEN.	C1>100
IN-list	A predicate of the form column IN (list of values).	C1 IN (5,10,15)
NOT	Any predicate that is not a subquery predicate and contains a NOT operator. Also included are predicates of the form C1 IS DISTINCT FROM.	COL1 <> 5 or COL1 NOT BETWEEN 10 AND 20

Example: Influence of type on access paths: The following two examples show how the predicate type can influence DB2's choice of an access path. In each one, assume that a unique index I1 (C1) exists on table T1 (C1, C2), and that all values of C1 are positive integers.

The following query has a range predicate:

```
SELECT C1, C2 FROM T1 WHERE C1 >= 0;
```

However, the predicate does not eliminate any rows of T1. Therefore, it could be determined during bind that a table space scan is more efficient than the index scan.

The following query has an equal predicate:

```
SELECT * FROM T1 WHERE C1 = 0;
```

DB2 chooses the index access in this case because the index is highly selective on column C1.

Indexable and nonindexable predicates

Definition: *Indexable* predicate types can match index entries; other types cannot. Indexable predicates might not become matching predicates of an index; it depends on the indexes that are available and the access path chosen at bind time.

Examples: If the employee table has an index on the column LASTNAME, the following predicate can be a matching predicate:

```
SELECT * FROM DSN8810.EMP WHERE LASTNAME = 'SMITH';
```

The following predicate cannot be a matching predicate, because it is not indexable.

```
SELECT * FROM DSN8810.EMP WHERE SEX <> 'F';
```

Recommendation: To make your queries as efficient as possible, use indexable predicates in your queries and create suitable indexes on your tables. Indexable predicates allow the possible use of a matching index scan, which is often a very efficient access path.

Stage 1 and stage 2 predicates

Definition: Rows retrieved for a query go through two stages of processing.

1. *Stage 1* predicates (sometimes called *sargable*) can be applied at the first stage.
2. *Stage 2* predicates (sometimes called *nonsargable* or *residual*) cannot be applied until the second stage.

The following items determine whether a predicate is stage 1:

- Predicate syntax

  See Table 124 on page 754 for a list of simple predicates and their types. See Examples of predicate properties for information on compound predicate types.
- Type and length of constants or columns in the predicate

  A simple predicate whose syntax classifies it as indexable and stage 1 might not be indexable or stage 1 because it contains constants and columns whose lengths are too long.

  Example: The following predicate is not indexable:

  ```
  CHARCOL<'ABCDEFG', where CHARCOL is defined as CHAR(6)
  ```

  The predicate is not indexable because the length of the column is shorter than the length of the constant.

  Example: The following predicate is not stage 1:

  ```
  DECCOL>34.5, where DECCOL is defined as DECIMAL(18,2)
  ```

  The predicate is not stage 1 because the precision of the decimal column is greater than 15.

- Whether DB2 evaluates the predicate before or after a join operation. A predicate that is evaluated after a join operation is always a stage 2 predicate.
- Join sequence

  The same predicate might be stage 1 or stage 2, depending on the join sequence. *Join sequence* is the order in which DB2 joins tables when it evaluates a query. The join sequence is not necessarily the same as the order in which the tables appear in the predicate.

  Example: This predicate might be stage 1 or stage 2:

  ```
  T1.C1=T2.C1+1
  ```

  If T2 is the first table in the join sequence, the predicate is stage 1, but if T1 is the first table in the join sequence, the predicate is stage 2.

  You can determine the join sequence by executing EXPLAIN on the query and examining the resulting plan table. See Chapter 34, "Using EXPLAIN to improve SQL performance," on page 927 for details.

All indexable predicates are stage 1. The predicate C1 LIKE %BC is stage 1, but is not indexable.

Recommendation: Use stage 1 predicates whenever possible.

Boolean term (BT) predicates

Definition: A *Boolean term predicate*, or *BT predicate*, is a simple or compound predicate that, when it is evaluated false for a particular row, makes the entire WHERE clause false for that particular row.

Examples: In the following query P1, P2 and P3 are simple predicates:

```
SELECT * FROM T1 WHERE P1 AND (P2 OR P3);
```

- P1 is a simple BT predicate.
- P2 and P3 are simple non-BT predicates.
- P2 OR P3 is a compound BT predicate.
- P1 AND (P2 OR P3) is a compound BT predicate.

Effect on access paths: In single-index processing, only Boolean term predicates are chosen for matching predicates. Hence, only indexable Boolean term predicates are candidates for matching index scans. To match index columns by predicates that are not Boolean terms, DB2 considers multiple-index access.

In join operations, Boolean term predicates can reject rows at an earlier stage than can non-Boolean term predicates.

Recommendation: For join operations, choose Boolean term predicates over non-Boolean term predicates whenever possible.

Predicates in the ON clause

The ON clause supplies the join condition in an outer join. For a full outer join, the clause can use only equal predicates. For other outer joins, the clause can use any predicates except predicates that contain subqueries.

For left and right outer joins, and for inner joins, join predicates in the ON clause are treated the same as other stage 1 and stage 2 predicates. A stage 2 predicate in the ON clause is treated as a stage 2 predicate of the inner table.

For full outer join, the ON clause is evaluated during the join operation like a stage 2 predicate.

In an outer join, predicates that are evaluated after the join are stage 2 predicates. Predicates in a table expression can be evaluated before the join and can therefore be stage 1 predicates.

Example: In the following statement, the predicate "EDLEVEL > 100" is evaluated before the full join and is a stage 1 predicate:

```
SELECT * FROM (SELECT * FROM DSN8810.EMP
   WHERE EDLEVEL  > 100) AS X FULL JOIN DSN8810.DEPT
      ON X.WORKDEPT  = DSN8810.DEPT.DEPTNO;
```

For more information about join methods, see "Interpreting access to two or more tables (join)" on page 955.

General rules about predicate evaluation

Recommendations:

1. In terms of resource usage, the earlier a predicate is evaluated, the better.
2. Stage 1 predicates are better than stage 2 predicates because they disqualify rows earlier and reduce the amount of processing needed at stage 2.
3. When possible, try to write queries that evaluate the most restrictive predicates first. When predicates with a high filter factor are processed first, unnecessary rows are screened as early as possible, which can reduce processing cost at a later stage. However, a predicate's restrictiveness is only effective among predicates of the same type and the same evaluation stage. For information about filter factors, see "Predicate filter factors" on page 760.

This section contains the following topics:

- "Order of evaluating predicates"
- "Examples of predicate properties" on page 759
- "Predicate filter factors" on page 760
- "Column correlation" on page 766
- "DB2 predicate manipulation" on page 770
- "Predicates with encrypted data" on page 773

Order of evaluating predicates

Two sets of rules determine the order of predicate evaluation.

The first set:

1. Indexable predicates are applied first. All matching predicates on index key columns are applied first and evaluated when the index is accessed.

   Next, stage 1 predicates that have not been picked as matching predicates but still refer to index columns are applied to the index. This is called *index screening*.
2. Other stage 1 predicates are applied next.

   After data page access, stage 1 predicates are applied to the data.
3. Finally, the stage 2 predicates are applied on the returned data rows.

The second set of rules describes the order of predicate evaluation within each of the stages:

1. All equal predicates (including column IN *list*, where *list* has only one element, or column BETWEEN *value1* AND *value1*) are evaluated.
2. All range predicates and predicates of the form *column* IS NOT NULL are evaluated.
3. All other predicate types are evaluated.

After both sets of rules are applied, predicates are evaluated in the order in which they appear in the query. Because you specify that order, you have some control over the order of evaluation.

Exception: Regardless of coding order, non-correlated subqueries are evaluated before correlated subqueries, unless DB2 transforms the subquery into a join.

Summary of predicate processing

Table 124 lists many of the simple predicates and tells whether those predicates are indexable or stage 1. The following terms are used:

- *subq* means a correlated or noncorrelated subquery.
- *noncor subq* means a noncorrelated subquery.
- *cor subq* means a correlated subquery.
- *op* is any of the operators >, >=, <, <=, ¬>, ¬<.
- *value* is a constant, host variable, or special register.
- *pattern* is any character string that does *not* start with the special characters for percent (%) or underscore (_).
- *char* is any character string that does *not* include the special characters for percent (%) or underscore (_).
- *expression* is any expression that contains arithmetic operators, scalar functions, aggregate functions, concatenation operators, columns, constants, host variables, special registers, or date or time expressions.
- *noncol expr* is a noncolumn expression, which is any expression that does not contain a column. That expression can contain arithmetic operators, scalar functions, concatenation operators, constants, host variables, special registers, or date or time expressions.

  An example of a noncolumn expression is

  ```
  CURRENT DATE - 50 DAYS
  ```
- T*n* *col expr* is an expression that contains a column in table T*n*. The expression might be only that column.
- *predicate* is a predicate of any type.

In general, if you form a compound predicate by combining several simple predicates with OR operators, the result of the operation has the same characteristics as the simple predicate that is evaluated latest. For example, if two indexable predicates are combined with an OR operator, the result is indexable. If a stage 1 predicate and a stage 2 predicate are combined with an OR operator, the result is stage 2.

Table 124. Predicate types and processing

Predicate Type	Index-able?	Stage 1?	Notes
COL = *value*	Y	Y	16
COL = *noncol expr*	Y	Y	9, 11, 12, 15
COL IS NULL	Y	Y	20, 21

Table 124. Predicate types and processing (continued)

Predicate Type	Index-able?	Stage 1?	Notes
COL *op value*	Y	Y	13
COL *op noncol expr*	Y	Y	9, 11, 12, 13
COL BETWEEN *value1* AND *value2*	Y	Y	13
COL BETWEEN *noncol expr1* AND *noncol expr2*	Y	Y	9, 11, 12, 13, 15,23
value BETWEEN COL1 AND COL2	N	N	
COL BETWEEN COL1 AND COL2	N	N	10
COL BETWEEN *expression1* AND *expression2*	Y	Y	6, 7, 11, 12, 13, 14
COL LIKE '*pattern*'	Y	Y	5
COL IN (*list*)	Y	Y	17, 18
COL <> *value*	N	Y	8, 11
COL <> *noncol expr*	N	Y	8, 11
COL IS NOT NULL	Y	Y	21
COL NOT BETWEEN *value1* AND *value2*	N	Y	
COL NOT BETWEEN *noncol expr1* AND *noncol expr2*	N	Y	
value NOT BETWEEN COL1 AND COL2	N	N	
COL NOT IN (*list*)	N	Y	
COL NOT LIKE ' *char*'	N	Y	5
COL LIKE '%*char*'	N	Y	1, 5
COL LIKE '_*char*'	N	Y	1, 5
COL LIKE *host variable*	Y	Y	2, 5
T1.COL = T2 *col expr*	Y	Y	6, 9, 11, 12, 14, 15, 25
T1.COL *op* T2 *col expr*	Y	Y	6, 9, 11, 12, 13, 14, 15
T1.COL <> T2 *col expr*	N	Y	8, 11
T1.COL1 = T1.COL2	N	N	3, 25
T1.COL1 *op* T1.COL2	N	N	3
T1.COL1 <> T1.COL2	N	N	3
COL=(*noncor subq*)	Y	Y	
COL = ANY (*noncor subq*)	N	N	22
COL = ALL (*noncor subq*)	N	N	
COL *op* (*noncor subq*)	Y	Y	26 on page 759
COL *op* ANY (*noncor subq*)	Y	Y	22
COL *op* ALL (*noncor subq*)	Y	Y	

Table 124. Predicate types and processing (continued)

Predicate Type	Index-able?	Stage 1?	Notes
COL <> (*noncor subq*)	N	Y	
# COL <> ANY (*noncor subq*)	N	N	22
COL <> ALL (*noncor subq*)	N	N	
# COL IN (*noncor subq*)	Y	Y	24
(COL1,...COL*n*) IN (*noncor subq*)	Y	Y	
COL NOT IN (*noncor subq*)	N	N	
(COL1,...COL*n*) NOT IN (*noncor subq*)	N	N	
COL = (*cor subq*)	N	N	4
# COL = ANY (*cor subq*)	N	N	22
COL = ALL (*cor subq*)	N	N	
COL *op* (*cor subq*)	N	N	4
# COL *op* ANY (*cor subq*)	N	N	22
COL *op* ALL (*cor subq*)	N	N	
COL <> (*cor subq*)	N	N	4
# COL <> ANY (*cor subq*)	N	N	22
COL <> ALL (*cor subq*)	N	N	
COL IN (*cor subq*)	N	N	19
(COL1,...COL*n*) IN (*cor subq*)	N	N	
COL NOT IN (*cor subq*)	N	N	
(COL1,...COL*n*) NOT IN (*cor subq*)	N	N	
COL IS DISTINCT FROM *value*	N	Y	8, 11
COL IS NOT DISTINCT FROM *value*	Y	Y	16
COL IS DISTINCT FROM *noncol expr*	N	Y	8, 11
COL IS NOT DISTINCT FROM *noncol expr*	Y	Y	9, 11, 12, 15
T1.COL1 IS DISTINCT FROM T2.COL2	N	N	3
T1.COL1 IS NOT DISTINCT FROM T2.COL2	N	N	3
T1.COL1 IS DISTINCT FROM T2 *col expr*	N	Y	8, 11
T1.COL1 IS NOT DISTINCT FROM T2 *col expr*	Y	Y	6, 9, 11, 12, 14, 15
COL IS DISTINCT FROM *(noncor subq)*	N	Y	
COL IS NOT DISTINCT FROM *(noncor subq)*	Y	Y	
# COL IS DISTINCT FROM ANY *(noncor subq)*	N	N	22
# COL IS NOT DISTINCT FROM ANY *(noncor subq)*	N	N	22
COL IS DISTINCT FROM ALL *(noncor subq)*	N	N	
COL IS NOT DISTINCT FROM ALL *(noncor subq)*	N	N	4
COL IS NOT DISTINCT FROM *(cor subq)*	N	N	4
# COL IS DISTINCT FROM ANY *(cor subq)*	N	N	22
# COL IS DISTINCT FROM ANY *(cor subq)*	N	N	22
# COL IS NOT DISTINCT FROM ANY *(cor subq)*	N	N	22

Table 124. Predicate types and processing (continued)

Predicate Type	Index-able?	Stage 1?	Notes
COL IS DISTINCT FROM ALL *(cor subq)*	N	N	
COL IS NOT DISTINCT FROM ALL *(cor subq)*	N	N	
EXISTS (*subq*)	N	N	19
NOT EXISTS (*subq*)	N	N	
expression = *value*	N	N	
expression <> *value*	N	N	
expression op value	N	N	
expression op (*subq*)	N	N	

Notes to Table 124 on page 754:

1. Indexable only if an ESCAPE character is specified and used in the LIKE predicate. For example, COL LIKE '+%*char*' ESCAPE '+' is indexable.
2. Indexable only if the pattern in the host variable is an indexable constant (for example, host variable='*char*%').
3. If both COL1 and COL2 are from the same table, access through an index on either one is not considered for these predicates. However, the following query is an exception:

   ```
   SELECT *  FROM T1 A, T1 B WHERE A.C1 = B.C2;
   ```

   By using correlation names, the query treats one table as if it were two separate tables. Therefore, indexes on columns C1 and C2 are considered for access.
4. If the subquery has already been evaluated for a given correlation value, then the subquery might not have to be reevaluated.
5. Not indexable or stage 1 if a field procedure exists on that column.
6. The column on the left side of the join sequence must be in a different table from any columns on the right side of the join sequence.
7. The tables that contain the columns in *expression1* or *expression2* must already have been accessed.
8. The processing for WHERE NOT COL = *value* is like that for WHERE COL <> *value*, and so on.
9. If *noncol expr*, *noncol expr1*, or *noncol expr2* is a noncolumn expression of one of these forms, then the predicate is not indexable:
   - *noncol expr* + 0
   - *noncol expr* - 0
   - *noncol expr* * 1
   - *noncol expr* / 1
   - *noncol expr* CONCAT *empty string*
10. COL, COL1, and COL2 can be the same column or different columns. The columns are in the same table.
11. Any of the following sets of conditions make the predicate stage 2:
    - The first value obtained before the predicate is evaluated is DECIMAL(*p*,*s*), where *p*>15, and the second value obtained before the predicate is evaluated is REAL or FLOAT.

- The first value obtained before the predicate is evaluated is CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC, and the second value obtained before the predicate is evaluated is DATE, TIME, or TIMESTAMP.

12. The predicate is stage 1 but not indexable if the first value obtained before the predicate is evaluated is CHAR or VARCHAR, the second value obtained before the predicate is evaluated is GRAPHIC or VARGRAPHIC, and the first value obtained before the predicate is evaluated is not Unicode mixed.
13. If both sides of the comparison are strings, any of the following sets of conditions makes the predicate stage 1 but not indexable:
    - The first value obtained before the predicate is evaluated is CHAR or VARCHAR, and the second value obtained before the predicate is evaluated is GRAPHIC or VARGRAPHIC.
    - Both of the following conditions are true:
      - Both sides of the comparison are CHAR or VARCHAR, or both sides of the comparison are BINARY or VARBINARY
      - The length the first value obtained before the predicate is evaluated is less than the length of the second value obtained before the predicate is evaluated.
    - Both of the following conditions are true:
      - Both sides of the comparison are GRAPHIC or VARGRAPHIC.
      - The length of the first value obtained before the predicate is evaluated is less than the length of the second value obtained before the predicate is evaluated.
    - Both of the following conditions are true:
      - The first value obtained before the predicate is evaluated is GRAPHIC or VARGRAPHIC, and the second value obtained before the predicate is evaluated is CHAR or VARCHAR.
      - The length of the first value obtained before the predicate is evaluated is less than the length of the second value obtained before the predicate is evaluated.
14. If both sides of the comparison are strings, but the two sides have different CCSIDs, the predicate is stage 1 and indexable only if the first value obtained before the predicate is evaluated is Unicode and the comparison does not meet any of the conditions in note 13.
15. Under either of these circumstances, the predicate is stage 2:
    - *noncol expr* is a case expression.
    - All of the following conditions are true:
      - *noncol expr* is the product or the quotient of two noncolumn expressions
      - *noncol expr* is an integer value
      - COL is a FLOAT or a DECIMAL column
16. If COL has the ROWID data type, DB2 tries to use direct row access instead of index access or a table space scan.
17. If COL has the ROWID data type, and an index is defined on COL, DB2 tries to use direct row access instead of index access.
18. IN-list predicates are indexable and stage 1 if the following conditions are true:
    - The IN list contains only simple items. For example, constants, host variables, parameter markers, and special registers.
    - The IN list does not contain any aggregate functions or scalar functions.
    - The IN list is not contained in a trigger's WHEN clause.

- For numeric predicates where the left side column is DECIMAL with precision greater than 15, none of the items in the IN list are FLOAT.
- For string predicates, the coded character set identifier is the same as the identifier for the left side column.
- For DATE, TIME, and TIMESTAMP predicates, the left side column must be DATE, TIME, or TIMESTAMP.

19. COL IN (*corr subq*) and EXISTS (*corr subq*) predicates might become indexable and stage 1 if they are transformed to a join during processing.
20. The predicate types COL IS NULL and COL IS NOT NULL are stage 2 predicates when they query a column that is defined as NOT NULL.
21. If the predicate type is COL IS NULL and the column is defined as NOT NULL, the table is not accessed because C1 cannot be NULL.
22. The ANY and SOME keywords behave similarly. If a predicate with the ANY keyword is not indexable and not stage 1, a similar predicate with the SOME keyword is not indexable and not stage 1.
23. Under either of these circumstances, the predicate is stage 2:
    - *noncol expr* is a case expression.
    - *noncol expr* is the product or the quotient of two noncolumn expressions, that product or quotient is an integer value, and COL is a FLOAT or a DECIMAL column.
24. COL IN (*noncor subq*) is stage 1 for type N access only. Otherwise, it is stage 2.
25. If the inner table is as EBCDIC or ASCII column and the outer table is a Unicode column, the predicate is stage 1 and indexable.
26. This type of predicate is not stage 1 when a nullability mismatch is possible.

Examples of predicate properties

Assume that predicate P1 and P2 are simple, stage 1, indexable predicates:

P1 AND P2 is a compound, stage 1, indexable predicate.

P1 OR P2 is a compound, stage 1 predicate, not indexable except by a union of RID lists from two indexes.

The following examples of predicates illustrate the general rules shown in Table 124 on page 754. In each case, assume that there is an index on columns (C1,C2,C3,C4) of the table and that 0 is the lowest value in each column.

- WHERE C1=5 AND C2=7

  Both predicates are stage 1 and the compound predicate is indexable. A matching index scan could be used with C1 and C2 as matching columns.
- WHERE C1=5 AND C2>7

  Both predicates are stage 1 and the compound predicate is indexable. A matching index scan could be used with C1 and C2 as matching columns.
- WHERE C1>5 AND C2=7

  Both predicates are stage 1, but only the first matches the index. A matching index scan could be used with C1 as a matching column.
- WHERE C1=5 OR C2=7

  Both predicates are stage 1 but not Boolean terms. The compound is indexable. Multiple-index access for the compound predicate is not possible because there is no index that has C2 as the leading column. For single-index access, C1 and C2 can be only index screening columns.
- WHERE C1=5 OR C2<>7

The first predicate is indexable and stage 1, and the second predicate is stage 1 but not indexable. The compound predicate is stage 1 and not indexable.

- WHERE C1>5 OR C2=7

  Both predicates are stage 1 but not Boolean terms. The compound is indexable. Multiple-index access for the compound predicate is not possible because no index has C2 as the leading column. For single-index access, C1 and C2 can be only index screening columns.

- WHERE C1 IN (*cor subq*) AND C2=C1

  Both predicates are stage 2 and not indexable. The index is not considered for matching-index access, and both predicates are evaluated at stage 2.

- WHERE C1=5 AND C2=7 AND (C3 + 5) IN (7,8)

  The first two predicates only are stage 1 and indexable. The index is considered for matching-index access, and all rows satisfying those two predicates are passed to stage 2 to evaluate the third predicate.

- WHERE C1=5 OR C2=7 OR (C3 + 5) IN (7,8)

  The third predicate is stage 2. The compound predicate is stage 2 and all three predicates are evaluated at stage 2. The simple predicates are not Boolean terms and the compound predicate is not indexable.

- WHERE C1=5 OR (C2=7 AND C3=C4)

  The third predicate is stage 2. The two compound predicates (C2=7 AND C3=C4) and (C1=5 OR (C2=7 AND C3=C4)) are stage 2. All predicates are evaluated at stage 2.

- WHERE (C1>5 OR C2=7) AND C3 = C4

  The compound predicate (C1>5 OR C2=7) is indexable and stage 1. The simple predicate C3=C4 is not stage1; so the index is not considered for matching-index access. Rows that satisfy the compound predicate (C1>5 OR C2=7) are passed to stage 2 for evaluation of the predicate C3=C4.

Predicate filter factors

Definition: The *filter factor* of a predicate is a number between 0 and 1 that estimates the proportion of rows in a table for which the predicate is true. Those rows are said to *qualify* by that predicate.

Example: Suppose that DB2 can determine that column C1 of table T contains only five distinct values: A, D, Q, W and X. In the absence of other information, DB2 estimates that one-fifth of the rows have the value D in column C1. Then the predicate `C1='D'` has the filter factor 0.2 for table T.

How DB2 uses filter factors: Filter factors affect the choice of access paths by estimating the number of rows qualified by a set of predicates.

For simple predicates, the filter factor is a function of three variables:

1. The literal value in the predicate; for instance, 'D' in the previous example.
2. The operator in the predicate; for instance, '=' in the previous example and '<>' in the negation of the predicate.
3. Statistics on the column in the predicate. In the previous example, those include the information that column T.C1 contains only five values.

Recommendation: Control the first two of those variables when you write a predicate. Your understanding of how DB2 uses filter factors should help you write more efficient predicates.

Values of the third variable, statistics on the column, are kept in the DB2 catalog. You can update many of those values, either by running the utility RUNSTATS or by executing UPDATE for a catalog table. For information about using RUNSTATS, see "Gathering monitor statistics and update statistics" on page 912. For information on updating the catalog manually, see "Updating catalog statistics" on page 799.

If you intend to update the catalog with statistics of your own choice, you should understand how DB2 uses:
- "Default filter factors for simple predicates"
- "Filter factors for uniform distributions"
- "Interpolation formulas" on page 762
- "Filter factors for all distributions" on page 763

Default filter factors for simple predicates

Table 125 lists default filter factors for different types of predicates. DB2 uses those values when no other statistics exist.

Example: The default filter factor for the predicate C1 = 'D' is 1/25 (0.04). If D is actually not close to 0.04, the default probably does not lead to an optimal access path.

Table 125. DB2 default filter factors by predicate type

Predicate Type	Filter Factor
Col = literal	1/25
Col <> literal	1 – (1/25)
Col IS NULL	1/25
Col IS NOT DISTINCT FROM	1/25
Col IS DISTINCT FROM	1 – (1/25)
Col IN (literal list)	(number of literals)/25
Col *Op* literal	1/3
Col LIKE literal	1/10
Col BETWEEN literal1 and literal2	1/10

Note:
Op is one of these operators: <, <=, >, >=.
Literal is any constant value that is known at bind time.

Filter factors for uniform distributions

DB2 uses the filter factors in Table 126 if:

- There is a positive value in column COLCARDF of catalog table SYSIBM.SYSCOLUMNS for the column "Col".
- There are no additional statistics for "Col" in SYSIBM.SYSCOLDIST.

Example: If D is one of only five values in column C1, using RUNSTATS puts the value 5 in column COLCARDF of SYSCOLUMNS. If there are no additional statistics available, the filter factor for the predicate C1 = 'D' is 1/5 (0.2).

Table 126. DB2 uniform filter factors by predicate type

Predicate type	Filter factor
Col = literal	1/COLCARDF
Col <> literal	1 – (1/COLCARDF)

Table 126. DB2 uniform filter factors by predicate type (continued)

Predicate type	Filter factor
Col IS NULL	1/COLCARDF
Col IS NOT DISTINCT FROM	1/COLCARDF
Col IS DISTINCT FROM	1 – (1/COLCARDF)
Col IN (literal list)	number of literals /COLCARDF
Col *Op1* literal	interpolation formula
Col *Op2* literal	interpolation formula
Col LIKE literal	interpolation formula
Col BETWEEN literal1 and literal2	interpolation formula

Note:
Op1 is < or <=, and the literal is not a host variable.
Op2 is > or >=, and the literal is not a host variable.
Literal is any constant value that is known at bind time.

Filter factors for other predicate types: The examples selected in Table 125 on page 761 and Table 126 on page 761 represent only the most common types of predicates. If P1 is a predicate and F is its filter factor, then the filter factor of the predicate NOT P1 is (1 - F). But, filter factor calculation is dependent on many things, so a specific filter factor cannot be given for all predicate types.

Interpolation formulas

Definition: For a predicate that uses a range of values, DB2 calculates the filter factor by an *interpolation formula*. The formula is based on an estimate of the ratio of the number of values in the range to the number of values in the entire column of the table.

The formulas: The formulas that follow are rough estimates, subject to further modification by DB2. They apply to a predicate of the form `col op. literal`. The value of (Total Entries) in each formula is estimated from the values in columns HIGH2KEY and LOW2KEY in catalog table SYSIBM.SYSCOLUMNS for column *col*: Total Entries = (HIGH2KEY value - LOW2KEY value).

- For the operators < and <=, where the literal is not a host variable:
  (Literal value - LOW2KEY value) / (Total Entries)
- For the operators > and >=, where the literal is not a host variable:
  (HIGH2KEY value - Literal value) / (Total Entries)
- For LIKE or BETWEEN:
  (High literal value - Low literal value) / (Total Entries)

Example: For column C2 in a predicate, suppose that the value of HIGH2KEY is 1400 and the value of LOW2KEY is 200. For C2, DB2 calculates (Total Entries) = 1200.

For the predicate `C1 BETWEEN 800 AND 1100`, DB2 calculates the filter factor F as:

```
F = (1100 - 800)/1200 = 1/4 = 0.25
```

Interpolation for LIKE: DB2 treats a LIKE predicate as a type of BETWEEN predicate. Two values that bound the range qualified by the predicate are generated from the literal string in the predicate. Only the leading characters found before the escape character ('%' or '_') are used to generate the bounds. So if the

escape character is the first character of the string, the filter factor is estimated as 1, and the predicate is estimated to reject no rows.

Defaults for interpolation: DB2 might not interpolate in some cases; instead, it can use a default filter factor. Defaults for interpolation are:

- Relevant only for ranges, including LIKE and BETWEEN predicates
- Used only when interpolation is not adequate
- Based on the value of COLCARDF
- Used whether uniform or additional distribution statistics exist on the column if either of the following conditions is met:
  - The predicate does not contain constants
  - COLCARDF < 4.

Table 127 shows interpolation defaults for the operators <, <=, >, >= and for LIKE and BETWEEN.

Table 127. Default filter factors for interpolation

COLCARDF	Factor for Op	Factor for LIKE or BETWEEN
>=100000000	1/10,000	3/100000
>=10000000	1/3,000	1/10000
>=1000000	1/1,000	3/10000
>=100000	1/300	1/1000
>=10000	1/100	3/1000
>=1000	1/30	1/100
>=100	1/10	3/100
>=2	1/3	1/10
=1	1/1	1/1
<=0	1/3	1/10

Note: Op is one of these operators: <, <=, >, >=.

Filter factors for all distributions

RUNSTATS can generate additional statistics for a column or set of columns. DB2 can use that information to calculate filter factors. DB2 collects two kinds of distribution statistics:

Frequency
: The percentage of rows in the table that contain a value for a column or set of columns

Cardinality
: The number of distinct values in a set of columns

When they are used: Table 128 on page 764 lists the types of predicates on which these statistics are used.

Table 128. Predicates for which distribution statistics are used

Type of statistic	Single column or concatenated columns	Predicates
Frequency	Single	COL=*literal* COL IS NULL COL IN (*literal-list*) COL *op literal* COL BETWEEN *literal* AND *literal* COL=*host-variable* COL1=COL2 T1.COL=T2.COL COL IS NOT DISTINCT FROM
Frequency	Concatenated	COL=*literal* COL IS NOT DISTINCT FROM
Cardinality	Single	COL=*literal* COL IS NULL COL IN (*literal-list*) COL *op literal* COL BETWEEN *literal* AND *literal* COL=*host-variable* COL1=COL2 T1.COL=T2.COL COL IS NOT DISTINCT FROM
Cardinality	Concatenated	COL=*literal* COL=:*host-variable* COL1=COL2 COL IS NOT DISTINCT FROM

Note: *op* is one of these operators: <, <=, >, >=.

How they are used: Columns COLVALUE and FREQUENCYF in table SYSCOLDIST contain distribution statistics. Regardless of the number of values in those columns, running RUNSTATS deletes the existing values and inserts rows for frequent values.

You can run RUNSTATS without the FREQVAL option, with the FREQVAL option in the *correl-spec*, with the FREQVAL option in the *colgroup-spec*, or in both, with the following effects:

- If you run RUNSTATS without the FREQVAL option, RUNSTATS inserts rows for the 10 most frequent values for the first column of the specified index.
- If you run RUNSTATS with the FREQVAL option in the *correl-spec*, RUNSTATS inserts rows for concatenated columns of an index. The NUMCOLS option specifies the number of concatenated index columns. The COUNT option specifies the number of frequent values. You can collect most-frequent values, least-frequent values, or both.
- If you run RUNSTATS with the FREQVAL option in the *colgroup-spec*, RUNSTATS inserts rows for the columns in the column group that you specify. The COUNT option specifies the number of frequent values. You can collect most-frequent values, least-frequent values, or both.
- If you specify the FREQVAL option, RUNSTATS inserts rows for columns of the specified index and for columns in a column group.

See Part 2 of *DB2 Utility Guide and Reference* for more information about RUNSTATS. DB2 uses the frequencies in column FREQUENCYF for predicates that use the values in column COLVALUE and assumes that the remaining data are uniformly distributed.

Example: Filter factor for a single column

Suppose that the predicate is C1 IN ('3','5') and that SYSCOLDIST contains these values for column C1:

```
COLVALUE    FREQUENCYF
  '3'         .0153
  '5'         .0859
  '8'         .0627
```

The filter factor is .0153 + .0859 = .1012.

Example: Filter factor for correlated columns

Suppose that columns C1 and C2 are correlated. Suppose also that the predicate is C1='3' AND C2='5' and that SYSCOLDIST contains these values for columns C1 and C2:

```
COLVALUE    FREQUENCYF
'1' '1'       .1176
'2' '2'       .0588
'3' '3'       .0588
'3' '5'       .1176
'4' '4'       .0588
'5' '3'       .1764
'5' '5'       .3529
'6' '6'       .0588
```

The filter factor is .1176.

Using multiple filter factors to determine the cost of a query

When DB2 estimates the cost of a query, it determines the filter factor repeatedly and at various levels. For example, suppose that you execute the following query:

```
SELECT COLS FROM T1
  WHERE C1 = 'A'
  AND   C3 = 'B'
  AND   C4 = 'C';
```

Table T1 consists of columns C1, C2, C3, and C4. Index I1 is defined on table T1 and contains columns C1, C2, and C3.

Suppose that the simple predicates in the compound predicate have the following characteristics:

C1='A'	Matching predicate
C3='B'	Screening predicate
C4='C'	Stage 1, nonindexable predicate

To determine the cost of accessing table T1 through index I1, DB2 performs these steps:

1. Estimates the matching index cost. DB2 determines the index matching filter factor by using single-column cardinality and single-column frequency statistics because only one column can be a matching column.

2. Estimates the total index filtering. This includes matching and screening filtering. If statistics exist on column group (C1,C3), DB2 uses those statistics. Otherwise DB2 uses the available single-column statistics for each of these columns.

   DB2 will also use FULLKEYCARDF as a bound. Therefore, it can be critical to have column group statistics on column group (C1, C3) to get an accurate estimate.
3. Estimates the table-level filtering. If statistics are available on column group (C1,C3,C4), DB2 uses them. Otherwise, DB2 uses statistics that exist on subsets of those columns.

Important: If you supply appropriate statistics at each level of filtering, DB2 is more likely to choose the most efficient access path.

You can use RUNSTATS to collect any of the needed statistics.

Column correlation

Two columns of data, A and B of a single table, are correlated if the values in column A do not vary independently of the values in column B.

Example: Table 129 is an excerpt from a large single table. Columns CITY and STATE are highly correlated, and columns DEPTNO and SEX are entirely independent.

Table 129. Data from the CREWINFO table

CITY	STATE	DEPTNO	SEX	EMPNO	ZIPCODE
Fresno	CA	A345	F	27375	93650
Fresno	CA	J123	M	12345	93710
Fresno	CA	J123	F	93875	93650
Fresno	CA	J123	F	52325	93792
New York	NY	J123	M	19823	09001
New York	NY	A345	M	15522	09530
Miami	FL	B499	M	83825	33116
Miami	FL	A345	F	35785	34099
Los Angeles	CA	X987	M	12131	90077
Los Angeles	CA	A345	M	38251	90091

In this simple example, for every value of column CITY that equals 'FRESNO', there is the same value in column STATE ('CA').

How to detect column correlation

The first indication that column correlation is a problem is because of poor response times when DB2 has chosen an inappropriate access path. If you suspect two columns in a table (CITY and STATE in table CREWINFO) are correlated, then you can issue the following SQL queries that reflect the relationships between the columns:

```
SELECT COUNT (DISTINCT CITY) AS CITYCOUNT,
  COUNT (DISTINCT STATE) AS STATECOUNT FROM CREWINFO;
```

The result of the count of each distinct column is the value of COLCARDF in the DB2 catalog table SYSCOLUMNS. Multiply the previous two values together to get a preliminary result:

```
CITYCOUNT  x  STATECOUNT    =      ANSWER1
```

Then issue the following SQL statement:

```
SELECT COUNT(*) FROM
  (SELECT DISTINCT CITY, STATE
   FROM CREWINFO) AS V1;           (ANSWER2)
```

Compare the result of the previous count (ANSWER2) with ANSWER1. If ANSWER2 is less than ANSWER1, then the suspected columns are correlated.

Impacts of column correlation

DB2 might not determine the best access path, table order, or join method when your query uses columns that are highly correlated. Column correlation can make the estimated cost of operations cheaper than they actually are. Column correlation affects both single table queries and join queries.

Column correlation on the best matching columns of an index: The following query selects rows with females in department A345 from Fresno, California. Two indexes are defined on the table, Index 1 (CITY,STATE,ZIPCODE) and Index 2 (DEPTNO,SEX).

Query 1

```
SELECT ... FROM CREWINFO WHERE
  CITY = 'FRESNO' AND STATE = 'CA'         (PREDICATE1)
  AND DEPTNO = 'A345' AND SEX = 'F';       (PREDICATE2)
```

Consider the two compound predicates (labeled PREDICATE1 and PREDICATE2), their actual filtering effects (the proportion of rows they select), and their DB2 filter factors. Unless the proper catalog statistics are gathered, the filter factors are calculated as if the columns of the predicate are entirely independent (not correlated).

When the columns in a predicate correlate but the correlation is not reflected in catalog statistics, the actual filtering effect to be significantly different from the DB2 filter factor. Table 130 on page 768 shows how the actual filtering effect and the DB2 filter factor can differ, and how that difference can affect index choice and performance.

Table 130. Effects of column correlation on matching columns

	INDEX 1	**INDEX 2**
Matching predicates	Predicate1 CITY=FRESNO AND STATE=CA	Predicate2 DEPTNO=A345 AND SEX=F
Matching columns	2	2
DB2 estimate for matching columns (Filter Factor)	column=CITY, COLCARDF=4 Filter Factor=1/4 column=STATE, COLCARDF=3 Filter Factor=1/3	column=DEPTNO, COLCARDF=4 Filter Factor=1/4 column=SEX, COLCARDF=2 Filter Factor=1/2
Compound filter factor for matching columns	1/4 × 1/3 = 0.083	1/4 × 1/2 = 0.125
Qualified leaf pages based on DB2 estimations	0.083 × 10 = 0.83 INDEX CHOSEN (.8 < 1.25)	0.125 × 10 = 1.25
Actual filter factor based on data distribution	4/10	2/10
Actual number of qualified leaf pages based on compound predicate	4/10 × 10 = 4	2/10 × 10 = 2 BETTER INDEX CHOICE (2 < 4)

DB2 chooses an index that returns the fewest rows, partly determined by the smallest filter factor of the matching columns. Assume that filter factor is the only influence on the access path. The combined filtering of columns CITY and STATE seems very good, whereas the matching columns for the second index do not seem to filter as much. Based on those calculations, DB2 chooses Index 1 as an access path for Query 1.

The problem is that the filtering of columns CITY and STATE should not look good. Column STATE does almost no filtering. Since columns DEPTNO and SEX do a better job of filtering out rows, DB2 should favor Index 2 over Index 1.

Column correlation on index screening columns of an index: Correlation might also occur on nonmatching index columns, used for index screening. See "Nonmatching index scan (ACCESSTYPE=I and MATCHCOLS=0)" on page 951 for more information. Index screening predicates help reduce the number of data rows that qualify while scanning the index. However, if the index screening predicates are correlated, they do not filter as many data rows as their filter factors suggest. To illustrate this, use "Query 1" on page 767 with the following indexes on Table 129 on page 766:

```
Index 3 (EMPNO,CITY,STATE)
Index 4 (EMPNO,DEPTNO,SEX)
```

In the case of Index 3, because the columns CITY and STATE of Predicate 1 are correlated, the index access is not improved as much as estimated by the screening predicates and therefore Index 4 might be a better choice. (Note that index screening also occurs for indexes with matching columns greater than zero.)

Multiple table joins: In Query 2, Table 131 on page 769 is added to the original query (see "Query 1" on page 767) to show the impact of column correlation on join queries.

Table 131. Data from the DEPTINFO table

CITY	STATE	MANAGER	DEPT	DEPTNAME
Fresno	CA	Smith	J123	ADMIN
Los Angeles	CA	Jones	A345	LEGAL

Query 2

```
SELECT ... FROM CREWINFO T1,DEPTINFO T2
   WHERE T1.CITY = 'FRESNO' AND T1.STATE='CA'          (PREDICATE 1)
   AND T1.DEPTNO = T2.DEPT AND T2.DEPTNAME = 'LEGAL';
```

The order that tables are accessed in a join statement affects performance. The estimated combined filtering of Predicate1 is lower than its actual filtering. So table CREWINFO might look better as the first table accessed than it should.

Also, due to the smaller estimated size for table CREWINFO, a nested loop join might be chosen for the join method. But, if many rows are selected from table CREWINFO because Predicate1 does not filter as many rows as estimated, then another join method or join sequence might be better.

What to do about column correlation

If column correlation is causing DB2 to choose an inappropriate access path, try one of these techniques to alter the access path:

- For leading indexed columns, run the RUNSTATS utility with the KEYCARD option determine the column correlation. For all other column groups, run the RUNSTATS utility with the COLGROUP option.
- Run the RUNSTATS utility to collect column correlation information for any column group with the COLGROUP option.
- Update the catalog statistics manually.
- Use SQL that forces access through a particular index.

The last two techniques are discussed in “Special techniques to influence access path selection” on page 788.

The RUNSTATS utility collects the statistics DB2 needs to make proper choices about queries. With RUNSTATS, you can collect statistics on the concatenated key columns of an index and the number of distinct values for those concatenated columns. This gives DB2 accurate information to calculate the filter factor for the query.

Example: RUNSTATS collects statistics that benefit queries like this:

```
SELECT * FROM T1
 WHERE C1 = 'a' AND C2 = 'b' AND C3 = 'c' ;
```

where:

- The first three index keys are used (MATCHCOLS = 3).
- An index exists on C1, C2, C3, C4, C5.
- Some or all of the columns in the index are correlated in some way.

See “Using RUNSTATS to keep access path statistics current” on page 651 for information on using RUNSTATS to influence access path selection. See “Updating catalog statistics” on page 799 for information on updating catalog statistics manually.

DB2 predicate manipulation

In some specific cases, DB2 either modifies some predicates, or generates extra predicates. Although these modifications are transparent to you, they have a direct impact on the access path selection and your PLAN_TABLE results. This is because DB2 always uses an index access path when it is cost effective. Generating extra predicates provides more indexable predicates potentially, which creates more chances for an efficient index access path.

Therefore, to understand your PLAN_TABLE results, you must understand how DB2 manipulates predicates. The information in Table 124 on page 754 is also helpful.

Predicate modifications for IN-list predicates

If an IN-list predicate has only one item in its list, the predicate becomes an EQUAL predicate.

A set of simple, Boolean term, equal predicates on the same column that are connected by OR predicates can be converted into an IN-list predicate. For example: C1=5 or C1=10 or C1=15 converts to C1 IN (5,10,15).

When DB2 simplifies join operations

Because full outer joins are less efficient than left or right joins, and left and right joins are less efficient than inner joins, you should always try to use the simplest type of join operation in your queries. However, if DB2 encounters a join operation that it can simplify, it attempts to do so. In general, DB2 can simplify a join operation when the query contains a predicate or an ON clause that eliminates the null values that are generated by the join operation.

Example: Consider this query:

```
SELECT * FROM T1 X FULL JOIN T2 Y
  ON X.C1=Y.C1
  WHERE X.C2 > 12;
```

The outer join operation gives you these result table rows:

- The rows with matching values of C1 in tables T1 and T2 (the inner join result)
- The rows from T1 where C1 has no corresponding value in T2
- The rows from T2 where C1 has no corresponding value in T1

However, when you apply the predicate, you remove all rows in the result table that came from T2 where C1 has no corresponding value in T1. DB2 transforms the full join into a left join, which is more efficient:

```
SELECT * FROM T1 X LEFT JOIN T2 Y
  ON X.C1=Y.C1
  WHERE X.C2 > 12;
```

Example: The predicate, X.C2>12, filters out all null values that result from the right join:

```
SELECT * FROM T1 X RIGHT JOIN T2 Y
  ON X.C1=Y.C1
  WHERE X.C2>12;
```

Therefore, DB2 can transform the right join into a more efficient inner join without changing the result:

```
SELECT * FROM T1 X INNER JOIN T2 Y
  ON X.C1=Y.C1
  WHERE X.C2>12;
```

The predicate that follows a join operation must have the following characteristics before DB2 transforms an outer join into a simpler outer join or into an inner join:

- The predicate is a Boolean term predicate.
- The predicate is false if one table in the join operation supplies a null value for all of its columns.

These predicates are examples of predicates that can cause DB2 to simplify join operations:

```
T1.C1 > 10
T1.C1 IS NOT NULL
T1.C1 > 10 OR T1.C2 > 15
T1.C1 > T2.C1
T1.C1 IN (1,2,4)
T1.C1 LIKE 'ABC%'
T1.C1 BETWEEN 10 AND 100
12 BETWEEN T1.C1 AND 100
```

Example: This examples shows how DB2 can simplify a join operation because the query contains an ON clause that eliminates rows with unmatched values:

```
SELECT * FROM T1 X LEFT JOIN T2 Y
  FULL JOIN T3 Z ON Y.C1=Z.C1
  ON X.C1=Y.C1;
```

Because the last ON clause eliminates any rows from the result table for which column values that come from T1 or T2 are null, DB2 can replace the full join with a more efficient left join to achieve the same result:

```
SELECT * FROM T1 X LEFT JOIN T2 Y
  LEFT JOIN T3 Z ON Y.C1=Z.C1
  ON X.C1=Y.C1;
```

In one case, DB2 transforms a full outer join into a left join when you cannot write code to do it. This is the case where a view specifies a full outer join, but a subsequent query on that view requires only a left outer join.

Example: Consider this view:

```
CREATE VIEW V1 (C1,T1C2,T2C2) AS
  SELECT COALESCE(T1.C1, T2.C1), T1.C2, T2.C2
  FROM T1 X FULL JOIN T2 Y
  ON T1.C1=T2.C1;
```

This view contains rows for which values of C2 that come from T1 are null. However, if you execute the following query, you eliminate the rows with null values for C2 that come from T1:

```
SELECT * FROM V1
  WHERE T1C2 > 10;
```

Therefore, for this query, a left join between T1 and T2 would have been adequate. DB2 can execute this query as if the view V1 was generated with a left outer join so that the query runs more efficiently.

Predicates generated through transitive closure

When the set of predicates that belong to a query logically imply other predicates, DB2 can generate additional predicates to provide more information for access path selection.

Rules for generating predicates: For single-table or inner join queries, DB2 generates predicates for transitive closure if:

- The query has an equal type predicate: COL1=COL2. This could be:
  - A local predicate
  - A join predicate
- The query also has a Boolean term predicate on one of the columns in the first predicate with one of the following formats:
  - COL1 *op value*

    op is =, <>, >, >=, <, or <=.

    value is a constant, host variable, or special register.
  - COL1 (NOT) BETWEEN *value1* AND *value2*
  - COL1=COL3

For outer join queries, DB2 generates predicates for transitive closure if the query has an ON clause of the form COL1=COL2 and a before join predicate that has one of the following formats:

- COL1 *op value*

  op is =, <>, >, >=, <, or <=
- COL1 (NOT) BETWEEN *value1* AND *value2*

DB2 generates a transitive closure predicate for an outer join query only if the generated predicate does not reference the table with unmatched rows. That is, the generated predicate cannot reference the left table for a left outer join or the right table for a right outer join.

For a multiple-CCSID query, DB2 does not generate a transitive closure predicate if the predicate that would be generated has these characteristics:

- The generated predicate is a range predicate (*op* is >, >=, <, or <=).
- Evaluation of the query with the generated predicate results in different CCSID conversion from evaluation of the query without the predicate. See Chapter 4 of *DB2 SQL Reference* for information on CCSID conversion.

When a predicate meets the transitive closure conditions, DB2 generates a new predicate, whether or not it already exists in the WHERE clause.

The generated predicates have one of the following formats:

- COL *op value*

  op is =, <>, >, >=, <, or <=.

  value is a constant, host variable, or special register.
- COL (NOT) BETWEEN *value1* AND *value2*
- COL1=COL2 (for single-table or inner join queries only)

Example of transitive closure for an inner join: Suppose that you have written this query, which meets the conditions for transitive closure:

```
SELECT * FROM T1, T2
  WHERE T1.C1=T2.C1 AND
  T1.C1>10;
```

DB2 generates an additional predicate to produce this query, which is more efficient:

```
SELECT * FROM T1, T2
  WHERE T1.C1=T2.C1 AND
  T1.C1>10 AND
  T2.C1>10;
```

Example of transitive closure for an outer join: Suppose that you have written this outer join query:

```
SELECT * FROM
  (SELECT T1.C1 FROM T1 WHERE T1.C1>10) X
  LEFT JOIN
  (SELECT T2.C1 FROM T2)                Y
  ON X.C1 = Y.C1;
```

The before join predicate, T1.C1>10, meets the conditions for transitive closure, so DB2 generates a query that has the same result as this more-efficient query:

```
SELECT * FROM
  (SELECT T1.C1 FROM T1 WHERE T1.C1>10) X
  LEFT JOIN
  (SELECT T2.C1 FROM T2 WHERE T2.C1>10) Y
  ON X.C1 = Y.C1;
```

Predicate redundancy: A predicate is redundant if evaluation of other predicates in the query already determines the result that the predicate provides. You can specify redundant predicates or DB2 can generate them. DB2 does not determine that any of your query predicates are redundant. All predicates that you code are evaluated at execution time regardless of whether they are redundant. If DB2 generates a redundant predicate to help select access paths, that predicate is ignored at execution.

Adding extra predicates: DB2 performs predicate transitive closure only on equal and range predicates. However, you can help DB2 to choose a better access path by adding transitive closure predicates for other types of operators, such as IN or LIKE. For example, consider the following SELECT statement:

```
SELECT * FROM T1,T2
   WHERE T1.C1=T2.C1
     AND T1.C1 LIKE 'A%';
```

If `T1.C1=T2.C1` is true, and `T1.C1 LIKE 'A%'` is true, then `T2.C1 LIKE 'A%'` must also be true. Therefore, you can give DB2 extra information for evaluating the query by adding `T2.C1 LIKE 'A%'`:

```
SELECT * FROM T1,T2
   WHERE T1.C1=T2.C1
     AND T1.C1 LIKE 'A%'
     AND T2.C1 LIKE 'A%';
```

Predicates with encrypted data

DB2 provides built-in functions for data encryption and decryption. These functions can secure sensitive data, but they can also degrade the performance of some statements if they are not used carefully. If a predicate contains any operator other than = and <>, encrypted data must be decrypted before comparisons can be made. Decryption makes the predicates stage 2. For advice on avoiding unnecessary encryption and decryption for predicate evaluation, see “Performance recommendations for data encryption” on page 212.

Using host variables efficiently

Host variables require default filter factors: When you bind a static SQL statement that contains host variables, DB2 uses a default filter factor to determine the best access path for the SQL statement. For more information on filter factors, including default values, see "Predicate filter factors" on page 760.

DB2 often chooses an access path that performs well for a query with several host variables. However, in a new release or after maintenance has been applied, DB2 might choose a new access path that does not perform as well as the old access path. In many cases, the change in access paths is due to the default filter factors, which might lead DB2 to optimize the query in a different way.

The two ways to change the access path for a query that contains host variables are:

- Bind the package or plan that contains the query with the option REOPT(ALWAYS) or the option REOPT(ONCE).
- Rewrite the query.

Changing the access path at run time

You can use the following bind options to control how DB2 determines the access path for SQL statements with variable values:

REOPT(ALWAYS)	DB2 determines the access path for any SQL statement with variable values each time the statement is run.
REOPT(ONCE)	DB2 determines and caches the access path for any SQL statement with variable values only once at run time, using the first set of input variable values. If the statement is run multiple times, DB2 does not reoptimize each time unless the cached statement is invalidated or removed from the cache.
REOPT(NONE)	DB2 determines the access path at bind time, and does not change the access path at run time.

The REOPT(ALWAYS) bind option

Specify the REOPT(ALWAYS) bind option when you want DB2 to determine access paths at both bind time and run time for statements that contain host variables and special registers: At run time, DB2 uses the values in these variables to determine the access paths. If the statement runs multiple times, DB2 determines the access path each time that the statement runs.

Consider using the REOPT(ALWAYS) bind option in the following circumstances:

- The SQL statement does not perform well with the access path that is chosen at bind time.
- The SQL statement takes a relatively long time to execute. For long-running SQL statements, the performance gain from the better access path that is chosen based on the input variable values for each run can be greater than the performance cost of reoptimizing the access path each time that the statement runs.

To use the REOPT(ALWAYS) bind option most efficiently, first determine which SQL statements in your applications perform poorly with the REOPT(NONE) bind

option and the REOPT(ONCE) bind option. Separate the code containing those statements into units that you bind into packages with the REOPT(ALWAYS) option. Bind the rest of the code into packages using the REOPT(NONE) bind option or the REOPT(ONCE) bind option, as appropriate. Then bind the plan with the REOPT(NONE) bind option. Statements in the packages bound with REOPT(ALWAYS) are candidates for repeated reoptimization at run time.

Example: To determine which queries in plans and packages that are bound with the REOPT(ALWAYS) bind option will be reoptimized at run time, execute the following SELECT statements:

```
SELECT PLNAME,
  CASE WHEN STMTNOI <> 0
   THEN STMTNOI
   ELSE STMTNO
  END AS STMTNUM,
  SEQNO, TEXT
  FROM   SYSIBM.SYSSTMT
  WHERE STATUS IN ('B','F','G','J')
  ORDER BY PLNAME, STMTNUM, SEQNO;
```

```
SELECT COLLID, NAME, VERSION,
  CASE WHEN STMTNOI <> 0
   THEN STMTNOI
   ELSE STMTNO
  END AS STMTNUM,
  SEQNO, STMT
  FROM SYSIBM.SYSPACKSTMT
  WHERE STATUS IN ('B','F','G','J')
  ORDER BY COLLID, NAME, VERSION, STMTNUM, SEQNO;
```

If you specify the bind option VALIDATE(RUN), and a statement in the plan or package is not bound successfully, that statement is incrementally bound at run time. If you also specify the bind option REOPT(ALWAYS), DB2 reoptimizes the access path during the incremental bind.

Example: To determine which plans and packages have statements that will be incrementally bound, execute the following SELECT statements:

```
SELECT DISTINCT NAME
  FROM SYSIBM.SYSSTMT
  WHERE STATUS = 'F' OR STATUS = 'H';
```

```
SELECT DISTINCT COLLID, NAME, VERSION
  FROM   SYSIBM.SYSPACKSTMT
  WHERE STATUS = 'F' OR STATUS = 'H';
```

The REOPT(ONCE) bind option

You can use the REOPT(ONCE) bind option to determine the access path for an SQL statement at run time. The REOPT(ONCE) bind option determines the access path for an SQL statement only once at run time and works only with dynamic SQL statements. The REOPT(ONCE) bind option allows DB2 to store the access path for dynamic SQL statements in the dynamic statement cache.

Consider using the REOPT(ONCE) bind option in the following circumstances:

- The SQL statement is a dynamic SQL statement.
- The SQL statement does not perform well with the access path that is chosen at bind time.
- The SQL statement is relatively simple and takes a relatively short time to execute. For simple SQL statements, reoptimizing the access path each time that the statement runs can degrade performance more than using the access path from the first run for each subsequent run.

- The same SQL statement is repeated many times in a loop, or is run by many threads. Because of the dynamic statement cache, the access path that DB2 chooses for the first set of input variables will perform well for subsequent executions of the same SQL statement, even if the input variable values are different each time.

To use the REOPT(ONCE) bind option most efficiently, first determine which dynamic SQL statements in your applications perform poorly with the REOPT(NONE) bind option and the REOPT(ALWAYS) bind option. Separate the code containing those statements into units that you bind into packages with the REOPT(ONCE) option. Bind the rest of the code into packages using the REOPT(NONE) bind option or the REOPT(ALWAYS) bind option, as appropriate. Then bind the plan with the REOPT(NONE) bind option. A dynamic statement in a package that is bound with REOPT(ONCE) is a candidate for reoptimization the first time that the statement is run.

Example: To determine which queries in plans and packages that are bound with the REOPT(ONCE) bind option will be reoptimized at run time, execute the following SELECT statements:

```
SELECT PLNAME,
  CASE WHEN STMTNOI <> 0
   THEN STMTNOI
   ELSE STMTNO
  END AS STMTNUM,
  SEQNO, TEXT
  FROM   SYSIBM.SYSSTMT
  WHERE STATUS IN ('J')
  ORDER BY PLNAME, STMTNUM, SEQNO;
```

```
SELECT COLLID, NAME, VERSION,
  CASE WHEN STMTNOI <> 0
   THEN STMTNOI
   ELSE STMTNO
  END AS STMTNUM,
  SEQNO, STMT
  FROM SYSIBM.SYSPACKSTMT
  WHERE STATUS IN ('J')
  ORDER BY COLLID, NAME, VERSION, STMTNUM, SEQNO;
```

If you specify the bind option VALIDATE(RUN), and a statement in the plan or package is not bound successfully, that statement is incrementally bound at run time.

Example: To determine which plans and packages have statements that will be incrementally bound, execute the following SELECT statements:

```
SELECT DISTINCT NAME
  FROM SYSIBM.SYSSTMT
  WHERE STATUS = 'F' OR STATUS = 'H';
```

```
SELECT DISTINCT COLLID, NAME, VERSION
  FROM   SYSIBM.SYSPACKSTMT
  WHERE STATUS = 'F' OR STATUS = 'H';
```

The REOPT(NONE) bind option

You should use the REOPT(NONE) bind option when an SQL statement with variable values performs well with the access path that is chosen at bind time. Keep in mind that an SQL statement that performs well with the REOPT(NONE) bind option might perform even better with the bind options that change the access path at run time.

Rewriting queries to influence access path selection

> **Important**
>
> This section describes tactics for rewriting queries to influence how DB2 selects access paths. The access path selection "tricks" that are described in the section might cause significant performance degradation if they are not carefully implemented and monitored.
>
> **Example:** The selection method might change in a later release of DB2, causing your changes to degrade performance.
>
> Before and after you make any permanent changes, take performance measurements. When you migrate to a new release, evaluate the performance again. Be prepared to back out any changes that have degraded performance.

The examples that follow identify potential performance problems and offer suggestions for tuning the queries. However, before you rewrite any query, you should consider whether the REOPT(ALWAYS) or REOPT(ONCE) bind options can solve your access path problems. See “Changing the access path at run time” on page 774 for more information about REOPT(ALWAYS) and REOPT(ONCE).

Example 1: An equal predicate

An equal predicate has a default filter factor of 1/COLCARDF. The actual filter factor might be quite different.

Query:

```
SELECT * FROM DSN8810.EMP
 WHERE SEX = :HV1;
```

Assumptions: Because the column SEX has only two different values, 'M' and 'F', the value COLCARDF for SEX is 2. If the numbers of male and female employees are not equal, the actual filter factor of 1/2 is larger or smaller than the default, depending on whether `:HV1` is set to 'M' or 'F'.

Recommendation: One of these two actions can improve the access path:

- Bind the package or plan that contains the query with the REOPT(ALWAYS) bind option. This action causes DB2 to reoptimize the query at run time, using the input values you provide. You might also consider binding the package or plan with the REOPT(ONCE) bind option.
- Write predicates to influence the DB2 selection of an access path, based on your knowledge of actual filter factors. For example, you can break the query into three different queries, two of which use constants. DB2 can then determine the exact filter factor for most cases when it binds the plan.

```
SELECT (HV1);
  WHEN ('M')
    DO;
      EXEC SQL SELECT * FROM DSN8810.EMP
       WHERE SEX = 'M';
    END;
  WHEN ('F')
    DO;
      EXEC SQL SELECT * FROM DSN8810.EMP
       WHERE SEX = 'F';
    END;
  OTHERWISE
```

```
    DO:
      EXEC SQL SELECT * FROM DSN8810.EMP
       WHERE SEX = :HV1;
    END;
END;
```

Example 2: Known ranges

Table T1 has two indexes: T1X1 on column C1 and T1X2 on column C2.

Query:

```
SELECT * FROM T1
       WHERE C1 BETWEEN :HV1 AND :HV2
         AND C2 BETWEEN :HV3 AND :HV4;
```

Assumptions: You know that:

- The application always provides a narrow range on C1 and a wide range on C2.
- The desired access path is through index T1X1.

Recommendation: If DB2 does not choose T1X1, rewrite the query as follows, so that DB2 does not choose index T1X2 on C2:

```
SELECT * FROM T1
       WHERE C1 BETWEEN :HV1 AND :HV2
         AND (C2 BETWEEN :HV3 AND :HV4 OR 0=1);
```

Example 3: Variable ranges

Table T1 has two indexes: T1X1 on column C1 and T1X2 on column C2.

Query:

```
SELECT * FROM T1
       WHERE C1 BETWEEN :HV1 AND :HV2
         AND C2 BETWEEN :HV3 AND :HV4;
```

Assumptions: You know that the application provides both narrow and wide ranges on C1 and C2. Hence, default filter factors do not allow DB2 to choose the best access path in all cases. For example, a small range on C1 favors index T1X1 on C1, a small range on C2 favors index T1X2 on C2, and wide ranges on both C1 and C2 favor a table space scan.

Recommendation: If DB2 does not choose the best access path, try either of the following changes to your application:

- Use a dynamic SQL statement and embed the ranges of C1 and C2 in the statement. With access to the actual range values, DB2 can estimate the actual filter factors for the query. Preparing the statement each time it is executed requires an extra step, but it can be worthwhile if the query accesses a large amount of data.
- Include some simple logic to check the ranges of C1 and C2, and then execute one of these static SQL statements, based on the ranges of C1 and C2:

```
SELECT * FROM T1 WHERE C1 BETWEEN :HV1 AND :HV2
                AND  (C2 BETWEEN :HV3 AND :HV4 OR 0=1);

SELECT * FROM T1 WHERE C2 BETWEEN :HV3 AND :HV4
                AND  (C1 BETWEEN :HV1 AND :HV2 OR 0=1);

SELECT * FROM T1 WHERE (C1 BETWEEN :HV1 AND :HV2 OR 0=1)
                AND   (C2 BETWEEN :HV3 AND :HV4 OR 0=1);
```

Example 4: ORDER BY

Table T1 has two indexes: T1X1 on column C1 and T1X2 on column C2.

Query:

```
SELECT * FROM T1
       WHERE C1 BETWEEN :HV1 AND :HV2
       ORDER BY C2;
```

In this example, DB2 could choose one of the following actions:

- Scan index T1X1 and then sort the results by column C2
- Scan the table space in which T1 resides and then sort the results by column C2
- Scan index T1X2 and then apply the predicate to each row of data, thereby avoiding the sort

Which choice is best depends on the following factors:

- The number of rows that satisfy the range predicate
- The cluster ratio of the indexes

If the actual number of rows that satisfy the range predicate is significantly different from the estimate, DB2 might not choose the best access path.

Assumptions: You disagree with the DB2 choice.

Recommendation: In your application, use a dynamic SQL statement and embed the range of C1 in the statement. That allows DB2 to use the actual filter factor rather than the default, but requires extra processing for the PREPARE statement.

Example 5: A join operation

Tables A, B, and C each have indexes on columns C1, C2, C3, and C4.

Assumptions: The actual filter factors on table A are much larger than the default factors. Hence, DB2 underestimates the number of rows selected from table A and wrongly chooses that as the first table in the join.

Recommendations: You can:

- Reduce the estimated size of Table A by adding predicates
- Disfavor any index on the join column by making the join predicate on table A nonindexable

Example: The following query illustrates the second of those choices.

```
SELECT * FROM T1 A, T1 B, T1 C
       WHERE (A.C1 = B.C1 OR 0=1)
         AND A.C2 = C.C2
         AND A.C2 BETWEEN :HV1 AND :HV2
         AND A.C3 BETWEEN :HV3 AND :HV4
         AND A.C4 < :HV5
         AND B.C2 BETWEEN :HV6 AND :HV7
         AND B.C3 < :HV8
         AND C.C2 < :HV9;
```

The result of making the join predicate between A and B a nonindexable predicate (which cannot be used in single index access) disfavors the use of the index on column C1. This, in turn, might lead DB2 to access table A or B first. Or, it might lead DB2 to change the access type of table A or B, thereby influencing the join sequence of the other tables.

Writing efficient subqueries

Definitions: A *subquery* is a SELECT statement within the WHERE or HAVING clause of another SQL statement.

Decision needed: You can often write two or more SQL statements that achieve identical results, particularly if you use subqueries. The statements have different access paths, however, and probably perform differently.

Topic overview: The topics that follow describe different methods to achieve the results intended by a subquery and tell what DB2 does for each method. The information should help you estimate what method performs best for your query.

The first two methods use different types of subqueries:
- "Correlated subqueries"
- "Noncorrelated subqueries" on page 781

A subquery can sometimes be transformed into a join operation. Sometimes DB2 does that to improve the access path, and sometimes you can get better results by doing it yourself. The third method is:
- "Conditions for DB2 to transform a subquery into a join" on page 782

Finally, for a comparison of the three methods as applied to a single task, see:
- "Subquery tuning" on page 784

Correlated subqueries

Definition: A *correlated* subquery refers to at least one column of the outer query.

Any predicate that contains a correlated subquery is a stage 2 predicate unless it is transformed to a join.

Example: In the following query, the correlation name, X, illustrates the subquery's reference to the outer query block.

```
SELECT * FROM DSN8810.EMP X
   WHERE  JOB = 'DESIGNER'
      AND  EXISTS (SELECT 1
                   FROM   DSN8810.PROJ
                   WHERE  DEPTNO = X.WORKDEPT
                      AND  MAJPROJ = 'MA2100');
```

What DB2 does: A correlated subquery is evaluated for each qualified row of the outer query that is referred to. In executing the example, DB2:

1. Reads a row from table EMP where JOB='DESIGNER'.
2. Searches for the value of WORKDEPT from that row, in a table stored in memory.

   The in-memory table saves executions of the subquery. If the subquery has already been executed with the value of WORKDEPT, the result of the subquery is in the table and DB2 does not execute it again for the current row. Instead, DB2 can skip to step 5.
3. Executes the subquery, if the value of WORKDEPT is not in memory. That requires searching the PROJ table to check whether there is any project, where MAJPROJ is 'MA2100', for which the current WORKDEPT is responsible.
4. Stores the value of WORKDEPT and the result of the subquery in memory.
5. Returns the values of the current row of EMP to the application.

DB2 repeats this whole process for each qualified row of the EMP table.

Notes on the in-memory table: The in-memory table is applicable if the operator of the predicate that contains the subquery is one of the following operators:

<, <=, >, >=, =, <>, EXISTS, NOT EXISTS

The table is not used, however, if:

- There are more than 16 correlated columns in the subquery
- The sum of the lengths of the correlated columns is more than 256 bytes
- There is a unique index on a subset of the correlated columns of a table from the outer query

The in-memory table is a wrap-around table and does not guarantee saving the results of all possible duplicated executions of the subquery.

Noncorrelated subqueries

Definition: A *noncorrelated* subquery makes no reference to outer queries.

Example:

```
SELECT * FROM DSN8810.EMP
  WHERE  JOB = 'DESIGNER'
    AND  WORKDEPT IN (SELECT DEPTNO
                      FROM   DSN8810.PROJ
                      WHERE  MAJPROJ = 'MA2100');
```

What DB2 does: A noncorrelated subquery is executed once when the cursor is opened for the query. What DB2 does to process it depends on whether it returns a single value or more than one value. The query in the preceding example can return more than one value.

Single-value subqueries

When the subquery is contained in a predicate with a simple operator, the subquery is required to return 1 or 0 rows. The simple operator can be one of the following operators:

<, <=, >, >=, =, <>, NOT <, NOT <=, NOT >, NOT >=

The following noncorrelated subquery returns a single value:

```
SELECT *
FROM   DSN8810.EMP
WHERE  JOB = 'DESIGNER'
  AND  WORKDEPT <= (SELECT MAX(DEPTNO)
                    FROM   DSN8810.PROJ);
```

What DB2 does: When the cursor is opened, the subquery executes. If it returns more than one row, DB2 issues an error. The predicate that contains the subquery is treated like a simple predicate with a constant specified, for example, WORKDEPT <= *'value'*.

Stage 1 and stage 2 processing: The rules for determining whether a predicate with a noncorrelated subquery that returns a single value is stage 1 or stage 2 are generally the same as for the same predicate with a single variable.

Multiple-value subqueries

A subquery can return more than one value if the operator is one of the following:

op ANY, *op* ALL , *op* SOME, IN, EXISTS

where *op* is any of the operators >, >=, <, <=, NOT <, NOT <=, NOT >, NOT >=.

What DB2 does: If possible, DB2 reduces a subquery that returns more than one row to one that returns only a single row. That occurs when there is a range comparison along with ANY, ALL, or SOME. The following query is an example:

```
SELECT * FROM DSN8810.EMP
   WHERE  JOB = 'DESIGNER'
     AND  WORKDEPT <= ANY (SELECT DEPTNO
                           FROM   DSN8810.PROJ
                           WHERE  MAJPROJ = 'MA2100');
```

DB2 calculates the maximum value for DEPTNO from table DSN8810.PROJ and removes the ANY keyword from the query. After this transformation, the subquery is treated like a single-value subquery.

That transformation can be made with a *maximum value* if the range operator is:

- > or >= with the quantifier ALL
- < or <= with the quantifier ANY or SOME

The transformation can be made with a *minimum value* if the range operator is:

- < or <= with the quantifier ALL
- > or >= with the quantifier ANY or SOME

The resulting predicate is determined to be stage 1 or stage 2 by the same rules as for the same predicate with a single-valued subquery.

When a subquery is sorted: A noncorrelated subquery is sorted when the comparison operator is IN, NOT IN, = ANY, <> ANY, = ALL, or <> ALL. The sort enhances the predicate evaluation, reducing the amount of scanning on the subquery result. When the value of the subquery becomes smaller or equal to the expression on the left side, the scanning can be stopped and the predicate can be determined to be true or false.

When the subquery result is a character data type and the left side of the predicate is a datetime data type, then the result is placed in a work file without sorting. For some noncorrelated subqueries that use IN, NOT IN, = ANY, <> ANY, = ALL, or <> ALL comparison operators, DB2 can more accurately pinpoint an entry point into the work file, thus further reducing the amount of scanning that is done.

Results from EXPLAIN: For information about the result in a plan table for a subquery that is sorted, see "When are aggregate functions evaluated? (COLUMN_FN_EVAL)" on page 946.

Conditions for DB2 to transform a subquery into a join

For a SELECT, UPDATE, or DELETE statement, DB2 can sometimes transform a subquery into a join between the result table of a subquery and the result table of an outer query.

For a SELECT statement, DB2 does the transformation if the following conditions are true:

- The transformation does not introduce redundancy.
- The subquery appears in a WHERE clause.

- The subquery does not contain GROUP BY, HAVING, or aggregate functions.
- The subquery has only one table in the FROM clause.
- For a correlated subquery, the comparison operator of the predicate containing the subquery is IN, = ANY, or = SOME.
- For a noncorrelated subquery, the comparison operator of the predicate containing the subquery is IN, EXISTS, = ANY, or = SOME.
- For a noncorrelated subquery, the subquery select list has only one column, guaranteed by a unique index to have unique values.
- For a noncorrelated subquery, the left side of the predicate is a single column with the same data type and length as the subquery's column. (For a correlated subquery, the left side can be any expression.)

For an UPDATE or DELETE statement, or a SELECT statement that does not meet the previous conditions for transformation, DB2 does the transformation of a correlated subquery into a join if the following conditions are true:

- The transformation does not introduce redundancy.
- The subquery is correlated to its immediate outer query.
- The FROM clause of the subquery contains only one table, and the outer query (for SELECT), UPDATE, or DELETE references only one table.
- If the outer predicate is a quantified predicate with an operator of =ANY or an IN predicate, the following conditions are true:
  - The left side of the outer predicate is a single column.
  - The right side of the outer predicate is a subquery that references a single column.
  - The two columns have the same data type and length.
- The subquery does not contain the GROUP BY or DISTINCT clauses.
- The subquery does not contain aggregate functions.
- The SELECT clause of the subquery does not contain a user-defined function with an external action or a user-defined function that modifies data.
- The subquery predicate is a Boolean term predicate.
- The predicates in the subquery that provide correlation are stage 1 predicates.
- The subquery does not contain nested subqueries.
- The subquery does not contain a self-referencing UPDATE or DELETE.
- For a SELECT statement, the query does not contain the FOR UPDATE OF clause.
- For an UPDATE or DELETE statement, the statement is a searched UPDATE or DELETE.
- For a SELECT statement, parallelism is not enabled.

For a statement with multiple subqueries, DB2 does the transformation only on the last subquery in the statement that qualifies for transformation.

Example: The following subquery can be transformed into a join because it meets the first set of conditions for transformation:

```
SELECT * FROM EMP
  WHERE DEPTNO IN
   (SELECT DEPTNO FROM DEPT
      WHERE LOCATION IN ('SAN JOSE', 'SAN FRANCISCO')
      AND DIVISION = 'MARKETING');
```

If there is a department in the marketing division which has branches in both San Jose and San Francisco, the result of the SQL statement is not the same as if a join were done. The join makes each employee in this department appear twice because it matches once for the department of location San Jose and again of location San Francisco, although it is the same department. Therefore, it is clear that to transform a subquery into a join, the uniqueness of the subquery select list must be guaranteed. For this example, a unique index on any of the following sets of columns would guarantee uniqueness:

- (DEPTNO)
- (DIVISION, DEPTNO)
- (DEPTNO, DIVISION).

The resultant query is:

```
SELECT EMP.* FROM EMP, DEPT
   WHERE EMP.DEPTNO = DEPT.DEPTNO AND
         DEPT.LOCATION IN ('SAN JOSE', 'SAN FRANCISCO') AND
         DEPT.DIVISION = 'MARKETING';
```

Example: The following subquery can be transformed into a join because it meets the second set of conditions for transformation:

```
UPDATE T1 SET T1.C1 = 1
  WHERE T1.C1 =ANY
    (SELECT T2.C1 FROM T2
      WHERE T2.C2 = T1.C2);
```

Results from EXPLAIN: For information about the result in a plan table for a subquery that is transformed into a join operation, see "Is a subquery transformed into a join?" on page 946.

Subquery tuning

The following three queries all retrieve the same rows. All three retrieve data about all designers in departments that are responsible for projects that are part of major project MA2100. These three queries show that there are several ways to retrieve a desired result.

Query A: A join of two tables

```
SELECT DSN8810.EMP.* FROM DSN8810.EMP, DSN8810.PROJ
   WHERE  JOB = 'DESIGNER'
     AND  WORKDEPT = DEPTNO
     AND  MAJPROJ = 'MA2100';
```

Query B: A correlated subquery

```
SELECT * FROM DSN8810.EMP X
   WHERE  JOB = 'DESIGNER'
     AND  EXISTS (SELECT 1 FROM DSN8810.PROJ
                  WHERE  DEPTNO = X.WORKDEPT
                    AND  MAJPROJ = 'MA2100');
```

Query C: A noncorrelated subquery

```
SELECT * FROM DSN8810.EMP
    WHERE  JOB = 'DESIGNER'
      AND  WORKDEPT IN (SELECT DEPTNO FROM DSN8810.PROJ
                        WHERE  MAJPROJ = 'MA2100');
```

If you need columns from both tables EMP and PROJ in the output, you must use a join.

PROJ might contain duplicate values of DEPTNO in the subquery, so that an equivalent join cannot be written.

In general, query A might be the one that performs best. However, if there is no index on DEPTNO in table PROJ, then query C might perform best. The IN-subquery predicate in query C is indexable. Therefore, if an index on WORKDEPT exists, DB2 might do IN-list access on table EMP. If you decide that a join cannot be used and there is an available index on DEPTNO in table PROJ, then query B might perform best.

When looking at a problem subquery, see if the query can be rewritten into another format or see if there is an index that you can create to help improve the performance of the subquery.

Knowing the sequence of evaluation is important, for the different subquery predicates and for all other predicates in the query. If the subquery predicate is costly, perhaps another predicate could be evaluated before that predicate so that the rows would be rejected before even evaluating the problem subquery predicate.

Using scrollable cursors efficiently

The following recommendations help you get the best performance from your scrollable cursors:

- Determine when scrollable cursors work best for you.

  Scrollable cursors are a valuable tool for writing applications such as screen-based applications, in which the result table is small and you often move back and forth through the data. However, scrollable cursors require more DB2 processing than non-scrollable cursors. If your applications require large result tables or you only need to move sequentially forward through the data, use non-scrollable cursors.
- Declare scrollable cursors as SENSITIVE only if you need to see the latest data.

  If you do not need to see updates that are made by other cursors or application processes, using a cursor that you declare as INSENSITIVE requires less processing by DB2.

  If you need to see only some of the latest updates, and you do not need to see the results of insert operations, declare scrollable cursors as SENSITIVE STATIC. See Chapter 5 of *DB2 SQL Reference* for information about which updates you can see with a scrollable cursor that is declared as SENSITIVE STATIC.

  If you need to see all of the latest updates and inserts, declare scrollable cursors as SENSITIVE DYNAMIC.
- To ensure maximum concurrency when you use a scrollable cursor for positioned update and delete operations, specify ISOLATION(CS) and CURRENTDATA(NO) when you bind packages and plans that contain updatable scrollable cursors. See Chapter 31, “Improving concurrency,” on page 809 for more details.
- Use the FETCH FIRST *n* ROWS ONLY clause with scrollable cursors when it is appropriate.

  In a distributed environment, when you need to retrieve a limited number of rows, FETCH FIRST *n* ROWS ONLY can improve your performance for distributed queries that use DRDA access by eliminating unneeded network traffic. See Part 4 of *DB2 Application Programming and SQL Guide* for more information.

In a local environment, if you need to scroll through a limited subset of rows in a table, you can use FETCH FIRST *n* ROWS ONLY to make the result table smaller.

- In a distributed environment, if you do not need to use your scrollable cursors to modify data, do your cursor processing in a stored procedure.

  Using stored procedures can decrease the amount of network traffic that your application requires.

- In a TEMP database, create table spaces that are large enough for processing your scrollable cursors.

  DB2 uses declared temporary tables for processing the following types of scrollable cursors:

  – SENSITIVE STATIC SCROLL
  – INSENSITIVE SCROLL
  – ASENSITIVE SCROLL, if the cursor sensitivity is INSENSITIVE. A cursor that meets the criteria for a read-only cursor has an effective sensitivity of INSENSITIVE.

  See the DECLARE CURSOR statement in *DB2 SQL Reference*for more information about cursor sensitivity. See *DB2 Installation Guide* for more information about calculating the appropriate size for declared temporary tables for cursors.

- Remember to commit changes often for the following reasons:

  – You frequently need to leave scrollable cursors open longer than non-scrollable cursors.
  – There is an increased chance of deadlocks with scrollable cursors because scrollable cursors allow rows to be accessed and updated in any order. Frequent commits can decrease the chances of deadlocks.

  To prevent cursors from closing after commit operations, declare your scrollable cursors WITH HOLD.

- While sensitive static sensitive scrollable cursors are open against a table, DB2 will disallow reuse of space in that table space to prevent the scrollable cursor from fetching newly inserted rows that were not in the original result set. Although this is normal, it can result in a seemingly false out-of-space indication. The problem can be more noticeable in a data sharing environment with transactions that access LOBs. Consider the following preventive measures:

  – Check applications such that they commit frequently
  – Close sensitive scrollable cursors when no longer needed
  – Remove WITH HOLD parm for the sensitive scrollable cursor, if possible
  – Isolate LOB table spaces in a dedicated bufferpool in the data sharing environment

Writing efficient queries on tables with data-partitioned secondary indexes

The number of partitions that DB2 accesses to evaluate a query predicate can affect the performance of the query. A query that provides data retrieval through a data-partitioned secondary index (DPSI) might access some or all partitions of the DPSI. For a query that is based only on a DPSI key value or range, DB2 must examine all partitions. If the query also has predicates on the leading columns of the partitioning key, DB2 does not need to examine all partitions. The removal from consideration of inapplicable partitions is known as *page range screening* or *limited partition scan*. A limited partition scan can be determined at bind time or at run time. For example, a limited partition scan can be determined at bind time for

a predicate in which a column is compared to a constant. A limited partition scan occurs at run time if the column is compared to a host variable, parameter marker, or special register.

The following example demonstrates how you can use a partitioning index to enable a limited partition scan on a set of partitions that DB2 needs to examine to satisfy a query predicate.

Suppose that you create table Q1, with partitioning index DATE_IX and DPSI STATE_IX:

```
CREATE TABLESPACE TS1 NUMPARTS 3;

CREATE TABLE Q1 (DATE DATE,
 CUSTNO CHAR(5),
 STATE CHAR(2),
 PURCH_AMT DECIMAL(9,2))
  IN TS1
 PARTITION BY (DATE)
  (PARTITION 1 ENDING AT ('2002-1-31'),
   PARTITION 2 ENDING AT ('2002-2-28'),
   PARTITION 3 ENDING AT ('2002-3-31'));

CREATE INDEX DATE_IX ON Q1 (DATE) PARTITIONED CLUSTER;

CREATE INDEX STATE_IX ON Q1 (STATE) PARTITIONED;
```

Now suppose that you want to execute the following query against table Q1:

```
SELECT CUSTNO, PURCH_AMT
 FROM Q1
 WHERE STATE = 'CA';
```

Because the predicate is based only on values of a DPSI key (STATE), DB2 must examine all partitions to find the matching rows.

Now suppose that you modify the query in the following way:

```
SELECT CUSTNO, PURCH_AMT
 FROM Q1
 WHERE DATE BETWEEN '2002-01-01' AND '2002-01-31' AND
 STATE = 'CA';
```

Because the predicate is now based on values of a partitioning index key (DATE) and on values of a DPSI key (STATE), DB2 can eliminate the scanning of data partitions 2 and 3, which do not satisfy the query for the partitioning key. This can be determined at bind time because the columns of the predicate are compared to constants.

Now suppose that you use host variables instead of constants in the same query:

```
SELECT CUSTNO, PURCH_AMT
 FROM Q1
 WHERE DATE BETWEEN :hv1 AND :hv2 AND
  STATE = :hv3;
```

DB2 can use the predicate on the partitioning column to eliminate the scanning of unneeded partitions at run time.

Writing queries to take advantage of limited partition scan is especially useful when a correlation exists between columns that are in a partitioning index and columns that are in a DPSI.

For example, suppose that you create table Q2, with partitioning index DATE_IX and DPSI ORDERNO_IX:

```
CREATE TABLESPACE TS2 NUMPARTS 3;

CREATE TABLE Q2 (DATE DATE,
 ORDERNO CHAR(8),
 STATE CHAR(2),
 PURCH_AMT DECIMAL(9,2))
  IN TS2
 PARTITION BY (DATE)
  (PARTITION 1 ENDING AT ('2000-12-31'),
   PARTITION 2 ENDING AT ('2001-12-31'),
   PARTITION 3 ENDING AT ('2002-12-31'));

CREATE INDEX DATE_IX ON Q2 (DATE) PARTITIONED CLUSTER;

CREATE INDEX ORDERNO_IX ON Q2 (ORDERNO) PARTITIONED;
```

Also suppose that the first 4 bytes of each ORDERNO column value represent the four-digit year in which the order is placed. This means that the DATE column and the ORDERNO column are correlated.

To take advantage of limited partition scan, when you write a query that has the ORDERNO column in the predicate, also include the DATE column in the predicate. The partitioning index on DATE lets DB2 eliminate the scanning of partitions that are not needed to satisfy the query. For example:

```
SELECT ORDERNO, PURCH_AMT
 FROM Q2
 WHERE ORDERNO BETWEEN '2002AAAA' AND '2002ZZZZ' AND
 DATE BETWEEN '2002-01-01' AND '2002-12-31';
```

Special techniques to influence access path selection

Important

This section describes tactics for rewriting queries and modifying catalog statistics to influence how DB2 selects access paths. The access path selection "tricks"that are described in the section might cause significant performance degradation if they are not carefully implemented and monitored.

Example: The selection method might change in a later release of DB2, causing your changes to degrade performance.

Save the old catalog statistics or SQL before you consider making any changes to control the choice of access path. Before and after you make any changes, take performance measurements. When you migrate to a new release, evaluate the performance again. Be prepared to back out any changes that have degraded performance.

This section contains the following information about determining and changing access paths:

- Obtaining information about access paths
- “Minimizing overhead for retrieving few rows: OPTIMIZE FOR *n* ROWS” on page 790
- “Fetching a limited number of rows: FETCH FIRST *n* ROWS ONLY” on page 789

- "Using the CARDINALITY clause to improve the performance of queries with user-defined table function references" on page 793
- "Reducing the number of matching columns" on page 794
- "Rearranging the order of tables in a FROM clause" on page 798
- "Updating catalog statistics" on page 799
- "Using a subsystem parameter" on page 800
- "Giving optimization hints to DB2" on page 801

Obtaining information about access paths

You can obtain information about DB2 access paths by using the following methods:

- Use Visual Explain.

  The DB2 Visual Explain tool, which is invoked from a workstation client, can be used to display and analyze information on access paths chosen by DB2. Visual Explain displays the access path information in both graphic and text formats. The tool provides you with an easy-to-use interface to the PLAN_TABLE output and allows you to invoke EXPLAIN for dynamic SQL statements. You can also access the catalog statistics for certain referenced objects of an access path. In addition, the tool allows you to archive EXPLAIN output from previous SQL statements to analyze changes in your SQL environment. See *DB2 Visual Explain online help* for more information.
- Run OMEGAMON DB2 Performance Monitor accounting reports.

  Another way to track performance is with the OMEGAMON DB2 Performance Monitor accounting reports. The accounting report, short layout, ordered by PLANNAME, lists the primary performance figures. Check the plans that contain SQL statements whose access paths you tried to influence. If the elapsed time, TCB time, or number of getpage requests increases sharply without a corresponding increase in the SQL activity, then there could be a problem. You can use OMEGAMON Online Monitor to track events after your changes have been implemented, providing immediate feedback on the effects of your changes.
- Specify the bind option EXPLAIN.

  You can also use the EXPLAIN option when you bind or rebind a plan or package. Compare the new plan or package for the statement to the old one. If the new one has a table space scan or a nonmatching index space scan, but the old one did not, the problem is probably the statement. Investigate any changes in access path in the new plan or package; they could represent performance improvements or degradations. If neither the accounting report ordered by PLANNAME or PACKAGE nor the EXPLAIN statement suggest corrective action, use the OMEGAMON SQL activity reports for additional information. For more information on using EXPLAIN, see "Obtaining PLAN_TABLE information from EXPLAIN" on page 928.

Fetching a limited number of rows: FETCH FIRST *n* ROWS ONLY

In some applications, you execute queries that can return a large number of rows, but you need only a small subset of those rows. Retrieving the entire result table from the query can be inefficient. You can specify the FETCH FIRST *n* ROWS ONLY clause in a SELECT statement to limit the number of rows in the result table of a query to *n* rows. In addition, for a distributed query that uses DRDA access, FETCH FIRST *n* ROWS ONLY, DB2 prefetches only *n* rows.

Example: Suppose that you write an application that requires information on only the 20 employees with the highest salaries. To return only the rows of the employee table for those 20 employees, you can write a query like this:

```
SELECT LASTNAME, FIRSTNAME, EMPNO, SALARY
  FROM EMP
  ORDER BY SALARY DESC
  FETCH FIRST 20 ROWS ONLY;
```

Interaction between OPTIMIZE FOR n ROWS and FETCH FIRST n ROWS ONLY: In general, if you specify FETCH FIRST *n* ROWS ONLY but not OPTIMIZE FOR *n* ROWS in a SELECT statement, DB2 optimizes the query as if you had specified OPTIMIZE FOR *n* ROWS.

When both the FETCH FIRST *n* ROWS ONLY clause and the OPTIMIZE FOR *n* ROWS clause are specified, the value for the OPTIMIZE FOR *n* ROWS clause is used for access path selection.

Example: Suppose that you submit the following SELECT statement:

```
SELECT * FROM EMP
FETCH FIRST 5 ROWS ONLY
OPTIMIZE FOR 20 ROWS;
```

The OPTIMIZE FOR value of 20 rows is used for access path selection.

Minimizing overhead for retrieving few rows: OPTIMIZE FOR *n* ROWS

When an application executes a SELECT statement, DB2 assumes that the application will retrieve all the qualifying rows. This assumption is most appropriate for batch environments. However, for interactive SQL applications, such as SPUFI, it is common for a query to define a very large potential result set but retrieve only the first few rows. The access path that DB2 chooses might not be optimal for those interactive applications.

This section discusses the use of OPTIMIZE FOR *n* ROWS to affect the performance of interactive SQL applications. Unless otherwise noted, this information pertains to local applications. For more information on using OPTIMIZE FOR *n* ROWS in distributed applications, see Part 4 of *DB2 Application Programming and SQL Guide*.

What OPTIMIZE FOR n ROWS does: The OPTIMIZE FOR *n* ROWS clause lets an application declare its intent to do either of these things:

- Retrieve only a subset of the result set
- Give priority to the retrieval of the first few rows

DB2 uses the OPTIMIZE FOR *n* ROWS clause to choose access paths that minimize the response time for retrieving the first few rows. For distributed queries, the value of *n* determines the number of rows that DB2 sends to the client on each DRDA network transmission. See Part 4 of *DB2 Application Programming and SQL Guide* for more information on using OPTIMIZE FOR *n* ROWS in the distributed environment.

Use OPTIMIZE FOR 1 ROW to avoid sorts: You can influence the access path most by using OPTIMIZE FOR 1 ROW. OPTIMIZE FOR 1 ROW tells DB2 to select an access path that returns the first qualifying row quickly. This means that

whenever possible, DB2 avoids any access path that involves a sort. If you specify a value for *n* that is anything but 1, DB2 chooses an access path based on cost, and you won't necessarily avoid sorts.

How to specify OPTIMIZE FOR n ROWS for a CLI application: For a Call Level Interface (CLI) application, you can specify that DB2 uses OPTIMIZE FOR *n* ROWS for all queries. To do that, specify the keyword OPTIMIZEFORNROWS in the initialization file. For more information, see Chapter 3 of *DB2 ODBC Guide and Reference*.

How many rows you can retrieve with OPTIMIZE FOR n ROWS: The OPTIMIZE FOR *n* ROWS clause does not prevent you from retrieving all the qualifying rows. However, if you use OPTIMIZE FOR *n* ROWS, the total elapsed time to retrieve all the qualifying rows might be significantly greater than if DB2 had optimized for the entire result set.

When OPTIMIZE FOR n ROWS is effective: OPTIMIZE FOR *n* ROWS is effective only on queries that can be performed incrementally. If the query causes DB2 to gather the whole result set before returning the first row, DB2 ignores the OPTIMIZE FOR *n* ROWS clause, as in the following situations:

- The query uses SELECT DISTINCT or a set function distinct, such as COUNT(DISTINCT C1).
- Either GROUP BY or ORDER BY is used, and no index can give the necessary ordering.
- A aggregate function and no GROUP BY clause is used.
- The query uses UNION.

Example: Suppose that you query the employee table regularly to determine the employees with the highest salaries. You might use a query like this:

```
SELECT LASTNAME, FIRSTNAME, EMPNO, SALARY
  FROM EMP
  ORDER BY SALARY DESC;
```

An index is defined on column EMPNO, so employee records are ordered by EMPNO. If you have also defined a descending index on column SALARY, that index is likely to be very poorly clustered. To avoid many random, synchronous I/O operations, DB2 would most likely use a table space scan, then sort the rows on SALARY. This technique can cause a delay before the first qualifying rows can be returned to the application.

If you add the OPTIMIZE FOR *n* ROWS clause to the statement, DB2 will probably use the SALARY index directly because you have indicated that you expect to retrieve the salaries of only the 20 most highly paid employees.

Example: The following statement uses that strategy to avoid a costly sort operation:

```
SELECT LASTNAME,FIRSTNAME,EMPNO,SALARY
  FROM EMP
  ORDER BY SALARY DESC
  OPTIMIZE FOR 20 ROWS;
```

Effects of using OPTIMIZE FOR n ROWS:

- The join method could change. Nested loop join is the most likely choice, because it has low overhead cost and appears to be more efficient if you want to retrieve only one row.

- An index that matches the ORDER BY clause is more likely to be picked. This is because no sort would be needed for the ORDER BY.
- List prefetch is less likely to be picked.
- Sequential prefetch is less likely to be requested by DB2 because it infers that you only want to see a small number of rows.
- In a join query, the table with the columns in the ORDER BY clause is likely to be picked as the outer table if there is an index on that outer table that gives the ordering needed for the ORDER BY clause.

Recommendation: For a local query, specify OPTIMIZE FOR *n* ROWS only in applications that frequently fetch only a small percentage of the total rows in a query result set. For example, an application might read only enough rows to fill the end user's terminal screen. In cases like this, the application might read the remaining part of the query result set only rarely. For an application like this, OPTIMIZE FOR *n* ROWS can result in better performance by causing DB2 to favor SQL access paths that deliver the first *n* rows as fast as possible.

When you specify OPTIMIZE FOR *n* ROWS for a remote query, a small value of *n* can help limit the number of rows that flow across the network on any given transmission.

You can improve the performance for receiving a large result set through a remote query by specifying a large value of *n* in OPTIMIZE FOR *n* ROWS. When you specify a large value, DB2 attempts to send the *n* rows in multiple transmissions. For better performance when retrieving a large result set, in addition to specifying OPTIMIZE FOR *n* ROWS with a large value of *n* in your query, do not execute other SQL statements until the entire result set for the query is processed. If retrieval of data for several queries overlaps, DB2 might need to buffer result set data in the DDF address space. See “Block fetching result sets” on page 1005 for more information.

For local or remote queries, to influence the access path most, specify OPTIMIZE for 1 ROW. This value does not have a detrimental effect on distributed queries.

Favoring index access

One common database design involves tables that contain groups of rows that logically belong together. Within each group, the rows should be accessed in the same sequence every time. The sequence is determined by the primary key on the table. Lock contention can occur when DB2 chooses different access paths for different applications that operate on a table with this design.

To minimize contention among applications that access tables with this design, specify the VOLATILE keyword when you create or alter the tables. A table that is defined with the VOLATILE keyword is known as a *volatile table*. When DB2 executes queries that include volatile tables, DB2 uses index access whenever possible. One exception is for DELETE statements on a VOLATILE table in a segmented table space when no WHERE clause is specified. In this case, a table space scan is used. As well as minimizing contention, using index access preserves the access sequence that the primary key provides.

Defining a table as volatile has a similar effect on a query to setting the NPGTHRSH subsystem parameter to favor matching index access for all qualified tables. (See “Using a subsystem parameter” on page 800 for information on the settings for NPGTHRSH.) However, the effect of NPGTHRSH is subsystem-wide,

and index access might not be appropriate for many queries. Defining tables as volatile lets you limit the set of queries that favor index access to queries that involve the volatile tables.

Using a subsystem parameter to control outer join processing

Subsystem parameter OJPERFEH can improve outer join processing. In particular, when the value of OJPERFEH is YES, DB2 takes the following actions, which can improve outer join processing in most cases:

- Does not merge table expressions or views if the parent query block of a table expression or view contains an outer join, and the merge would cause a column in a predicate to become an expression.
- Does not attempt to reduce work file usage for outer joins.
- Uses transitive closure for the ON predicates in outer joins.

However, these actions might not improve performance for some outer joins.

Recommendation: If the performance of queries that contain outer joins is not adequate, set OJPERFEH to NO, restart DB2, and rerun those queries.

Using the CARDINALITY clause to improve the performance of queries with user-defined table function references

The cardinality of a user-defined table function is the number of rows that are returned when the function is invoked. DB2 uses this number to estimate the cost of executing a query that invokes a user-defined table function. The cost of executing a query is one of the factors that DB2 uses when it calculates the access path. Therefore, if you give DB2 an accurate estimate of a user-defined table function's cardinality, DB2 can better calculate the best access path.

You can specify a cardinality value for a user-defined table function by using the CARDINALITY clause of the SQL CREATE FUNCTION or ALTER FUNCTION statement. However, this value applies to all invocations of the function, whereas a user-defined table function might return different numbers of rows, depending on the query in which it is referenced.

To give DB2 a better estimate of the cardinality of a user-defined table function for a particular query, you can use the CARDINALITY or CARDINALITY MULTIPLIER clause in that query. DB2 uses those clauses at bind time when it calculates the access cost of the user-defined table function. Using this clause is recommended only for programs that run on DB2 UDB for z/OS because the clause is not supported on earlier versions of DB2.

Example of using the CARDINALITY clause to specify the cardinality of a user-defined table function invocation: Suppose that when you created user-defined table function TUDF1, you set a cardinality value of 5, but in the following query, you expect TUDF1 to return 30 rows:

```
SELECT *
 FROM TABLE(TUDF1(3)) AS X;
```

Add the CARDINALITY 30 clause to tell DB2 that, for this query, TUDF1 should return 30 rows:

```
SELECT *
 FROM TABLE(TUDF1(3) CARDINALITY 30) AS X;
```

Example of using the CARDINALITY MULTIPLIER clause to specify the cardinality of a user-defined table function invocation: Suppose that when you created user-defined table function TUDF2, you set a cardinality value of 5, but in the following query, you expect TUDF2 to return 30 times that many rows:

```
SELECT *
 FROM TABLE(TUDF2(10)) AS X;
```

Add the CARDINALITY MULTIPLIER 30 clause to tell DB2 that, for this query, TUDF1 should return 5*30, or 150, rows:

```
SELECT *
 FROM TABLE(TUDF2(10) CARDINALITY MULTIPLIER 30) AS X;
```

Reducing the number of matching columns

Discourage the use of a poorer performing index by reducing the index's matching predicate on its leading column. Consider the example in Figure 77 on page 795, where the index that DB2 picks is less than optimal.

```
CREATE TABLE PART_HISTORY (
    PART_TYPE  CHAR(2),       IDENTIFIES THE PART TYPE
    PART_SUFFIX CHAR(10),     IDENTIFIES THE PART
    W_NOW      INTEGER,       TELLS WHERE THE PART IS
    W_FROM     INTEGER,       TELLS WHERE THE PART CAME FROM
    DEVIATIONS INTEGER,       TELLS IF ANYTHING SPECIAL WITH THIS PART
    COMMENTS        CHAR(254),
    DESCRIPTION     CHAR(254),
    DATE1           DATE,
    DATE2           DATE,
    DATE3           DATE);

  CREATE UNIQUE INDEX IX1 ON PART_HISTORY
    (PART_TYPE,PART_SUFFIX,W_FROM,W_NOW);
  CREATE UNIQUE INDEX IX2 ON PART_HISTORY
    (W_FROM,W_NOW,DATE1);

  +-------------------------------------------------------------------------+
  |  Table statistics              |        Index statistics   IX1         IX2  |
  |--------------------------------+----------------------------------------|
  | CARDF            100,000       |      FIRSTKEYCARDF       1000          50  |
  | NPAGES            10,000       |      FULLKEYCARDF     100,000     100,000 |
  |                                |      CLUSTERRATIO         99%         99%  |
  |                                |      NLEAF               3000        2000  |
  |                                |      NLEVELS                3           3  |
  |-------------------------------------------------------------------------|
  |                 column       cardinality    HIGH2KEY     LOW2KEY          |
  |                 --------     -----------    --------     -------          |
  |                 Part_type    1000           'ZZ'        'AA'              |
  |                 w_now          50            1000         1               |
  |                 w_from         50            1000         1               |
  +-------------------------------------------------------------------------+

  Q1:
  SELECT * FROM PART_HISTORY   --  SELECT ALL PARTS
  WHERE PART_TYPE = 'BB'    P1 --  THAT ARE 'BB' TYPES
    AND W_FROM = 3          P2 --  THAT WERE MADE IN CENTER 3
    AND W_NOW = 3           P3 --  AND ARE STILL IN CENTER 3

  +-------------------------------------------------------------------------+
  |   Filter factor of these predicates.                                    |
  |    P1 = 1/1000= .001                                                    |
  |    P2 = 1/50  = .02                                                     |
  |    P3 = 1/50  = .02                                                     |
  |-------------------------------------------------------------------------|
  |  ESTIMATED VALUES                 |    WHAT REALLY HAPPENS              |
  |                   filter    data  |                      filter   data  |
  |  index  matchcols  factor    rows |    index  matchcols   factor   rows |
  |   ix2     2       .02*.02      40 |     ix2     2       .02*.50    1000 |
  |   ix1     1       .001        100 |     ix1     1       .001        100 |
  +-------------------------------------------------------------------------+
```

Figure 77. Reducing the number of MATCHCOLS

DB2 picks IX2 to access the data, but IX1 would be roughly 10 times quicker. The problem is that 50% of all parts from center number 3 are still in Center 3; they have not moved. Assume that there are no statistics on the correlated columns in catalog table SYSCOLDIST. Therefore, DB2 assumes that the parts from center number 3 are evenly distributed among the 50 centers.

You can get the desired access path by changing the query. To discourage the use of IX2 for this particular query, you can change the third predicate to be nonindexable.

```
SELECT * FROM PART_HISTORY
  WHERE PART_TYPE = 'BB'
    AND W_FROM = 3
    AND (W_NOW = 3 + 0)      <-- PREDICATE IS MADE NONINDEXABLE
```

Now index I2 is not picked, because it has only one match column. The preferred index, I1, is picked. The third predicate is a nonindexable predicate, so an index is not used for the compound predicate.

You can make a predicate nonindexable in many ways. The recommended way is to add 0 to a predicate that evaluates to a numeric value or to concatenate an empty string to a predicate that evaluates to a character value.

Indexable	Nonindexable
`T1.C3=T2.C4`	`(T1.C3=T2.C4 CONCAT '')`
`T1.C1=5`	`T1.C1=5+0`

These techniques do not affect the result of the query and cause only a small amount of overhead.

The preferred technique for improving the access path when a table has correlated columns is to generate catalog statistics on the correlated columns. You can do that either by running RUNSTATS or by updating catalog table SYSCOLDIST manually.

Creating indexes for efficient star-join processing

A *star schema* is a database design that, in its simplest form, consists of a large table called a *fact table,* and two or more smaller tables, called *dimension tables.* More complex star schemas can be created by breaking one or more of the dimension tables into multiple tables.

To access the data in a star schema design, you often write SELECT statements that include join operations between the fact table and the dimension tables, but no join operations between dimension tables. These types of queries are known as *star-join queries.*

For a star-join query, DB2 uses a special join type called a *star join* if the following conditions are true:

- The tables meet the conditions that are specified in "Star join (JOIN_TYPE='S')" on page 963.
- The STARJOIN system parameter is set to ENABLE, and the number of tables in the query block is greater than or equal to the minimum number that is specified in the SJTABLES system parameter.

  See "Star join (JOIN_TYPE='S')" on page 963 for detailed discussions of these system parameters.

This section gives suggestions for choosing indexes might improve star-join query performance.

Recommendations for creating indexes for star-join queries

Follow these recommendations to improve performance of star-join queries:

- Define a multi-column index on all key columns of the fact table. Key columns are fact table columns that have corresponding dimension tables.
- If you do not have information about the way that your data is used, first try a multi-column index on the fact table that is based on the correlation of the data. Put less highly correlated columns later in the index key than more highly correlated columns. See "Determining the order of columns in an index for a star schema design" on page 797 for information on deriving an index that follows this recommendation.

- As the correlation of columns in the fact table changes, reevaluate the index to determine if columns in the index should be reordered.
- Define indexes on dimension tables to improve access to those tables.
- When you have executed a number of queries and have more information about the way that the data is used, follow these recommendations:
  - Put more selective columns at the beginning of the index.
  - If a number of queries do not reference a dimension, put the column that corresponds to that dimension at the end of the index.

    When a fact table has more than one multi-column index and none of those indexes contains all key columns, DB2 evaluates all of the indexes and uses the index that best exploits star join.

Determining the order of columns in an index for a star schema design

You can use the following method to determine the order of columns in a multi-column index. The description of the method uses the following terminology:

F A fact table.

D1...D*n*
Dimension tables.

C1...C*n*
Key columns in the fact table. C1 is joined to dimension D1, C2 is joined to dimension D2, and so on.

cardD1...cardD*n*
Cardinality of columns C1...C*n* in dimension tables D1...D*n*.

cardC1...cardC*n*
Cardinality of key columns C1...C*n* in fact table F.

cardC*ij*
Cardinality of pairs of column values from key columns C*i* and C*j* in fact table F.

cardC*ijk*
Cardinality of triplets of column values from key columns C*i*, C*j*, and C*k* in fact table F.

Density
A measure of the correlation of key columns in the fact table. The density is calculated as follows:

For a single column
cardC*i*/cardD*i*

For pairs of columns
cardC*ij*/(cardD*i**cardD*j*)

For triplets of columns
cardC*ijk*/(cardD*i**cardD*j**cardD*k*)

S The current set of columns whose order in the index is not yet determined.

S-{C*m*}
The current set of columns, excluding column C*m*

Follow these steps to derive a fact table index for a star-join query that joins *n* columns of fact table F to *n* dimension tables D1 through D*n*:

1. Define the set of columns whose index key order is to be determined as the *n* columns of fact table F that correspond to dimension tables. That is, S={C1,...C*n*} and L=*n*.
2. Calculate the density of all sets of L-1 columns in S.
3. Find the lowest density. Determine which column is not in the set of columns with the lowest density. That is, find column C*m* in S, such that for every C*i* in S, density(S-{C*m*})<density(S-{C*i*}).
4. Make C*m* the Lth column of the index.
5. Remove C*m* from S.
6. Decrement L by 1.
7. Repeat steps 2 through 6 *n*-2 times. The remaining column after iteration *n*-2 is the first column of the index.

Example of determining column order for a fact table index: Suppose that a star schema has three dimension tables with the following cardinalities:

```
cardD1=2000
cardD2=500
cardD3=100
```

Now suppose that the cardinalities of single columns and pairs of columns in the fact table are:

```
cardC1=2000
cardC2=433
cardC3=100
cardC12=625000
cardC13=196000
cardC23=994
```

Determine the best multi-column index for this star schema.

Step 1: Calculate the density of all pairs of columns in the fact table:

```
density(C1,C2)=625000/(2000*500)=0.625
density(C1,C3)=196000/(2000*100)=0.98
density(C2,C3)=994/(500*100)=0.01988
```

Step 2: Find the pair of columns with the lowest density. That pair is (C2,C3). Determine which column of the fact table is not in that pair. That column is C1.

Step 3: Make column C1 the third column of the index.

Step 4: Repeat steps 1 through 3 to determine the second and first columns of the index key:

```
density(C2)=433/500=0.866
density(C3)=100/100=1.0
```

The column with the lowest density is C2. Therefore, C3 is the second column of the index. The remaining column, C2, is the first column of the index. That is, the best order for the multi-column index is C2, C3, C1.

Rearranging the order of tables in a FROM clause

The order of tables or views in the FROM CLAUSE can affect the access path. If your query performs poorly, it could be because the join sequence is inefficient. You can determine the join sequence within a query block from the PLANNO column in the PLAN_TABLE. For information on using the PLAN_TABLE, see Chapter 34, “Using EXPLAIN to improve SQL performance,” on page 927. If you

think that the join sequence is inefficient, try rearranging the order of the tables and views in the FROM clause to match a join sequence that might perform better.

Updating catalog statistics

If you have the proper authority, you can influence access path selection by using an SQL UPDATE or INSERT statement to change statistical values in the DB2 catalog. However, this is not generally recommended except as a last resort. Although updating catalog statistics can help a certain query, other queries can be affected adversely. Also, the UPDATE statements must be repeated after RUNSTATS resets the catalog values. You should be very careful if you attempt to update statistics. For a list of catalog statistics that you can update, see Table 162 on page 899.

If you update catalog statistics for a table space or index manually, and you are using dynamic statement caching, you need to invalidate statements in the cache that involve those table spaces or indexes. To invalidate statements in the dynamic statement cache without updating catalog statistics or generating reports, you can run the RUNSTATS utility with the REPORT NO and UPDATE NONE options on the table space or the index that the query is dependent on.

The example shown in Figure 77 on page 795, involves this query:

```
SELECT * FROM PART_HISTORY   --  SELECT ALL PARTS
WHERE PART_TYPE = 'BB'    P1 --  THAT ARE 'BB' TYPES
  AND W_FROM = 3          P2 --  THAT WERE MADE IN CENTER 3
  AND W_NOW = 3           P3 --  AND ARE STILL IN CENTER 3
```

This query has a problem with data correlation. DB2 does not know that 50% of the parts that were made in Center 3 are still in Center 3. The problem was circumvented by making a predicate nonindexable. But suppose that hundreds of users are writing queries similar to that query. Having all users change their queries would be impossible. In this type of situation, the best solution is to change the catalog statistics.

For the query in Figure 77 on page 795, you can update the catalog statistics in one of two ways:

- Run the RUNSTATS utility, and request statistics on the correlated columns W_FROM and W_NOW. This is the preferred method. See "Gathering monitor statistics and update statistics" on page 912 and Part 2 of *DB2 Utility Guide and Reference*for more information.
- Update the catalog statistics manually.

Updating the catalog to adjust for correlated columns: One catalog table that you can update is SYSIBM.SYSCOLDIST, which gives information about a column or set of columns in a table. Assume that because columns W_NOW and W_FROM are correlated, only 100 distinct values exist for the combination of the two columns, rather than 2500 (50 for W_FROM * 50 for W_NOW). Insert a row like this to indicate the new cardinality:

```
INSERT INTO SYSIBM.SYSCOLDIST
       (FREQUENCY, FREQUENCYF, IBMREQD,
        TBOWNER, TBNAME, NAME, COLVALUE,
        TYPE, CARDF, COLGROUPCOLNO, NUMCOLUMNS)
 VALUES(0, -1, 'N',
        'USRT001','PART_HISTORY','W_FROM',' ',
        'C',100,X'00040003',2);
```

You can also use the RUNSTATS utility to put this information in SYSCOLDIST. See *DB2 Utility Guide and Reference* for more information.

You tell DB2 about the frequency of a certain combination of column values by updating SYSIBM.SYSCOLDIST. For example, you can indicate that 1% of the rows in PART_HISTORY contain the values 3 for W_FROM and 3 for W_NOW by inserting this row into SYSCOLDIST:

```
INSERT INTO SYSIBM.SYSCOLDIST
       (FREQUENCY, FREQUENCYF, STATSTIME, IBMREQD,
        TBOWNER, TBNAME, NAME, COLVALUE,
        TYPE, CARDF, COLGROUPCOLNO, NUMCOLUMNS)
 VALUES(0, .0100, '1996-12-01-12.00.00.000000','N',
        'USRT001','PART_HISTORY','W_FROM',X'00800000030080000003',
        'F',-1,X'00040003',2);
```

Updating the catalog for joins with table functions: Updating catalog statistics **might cause extreme performance problems** if the statistics are not updated correctly. Monitor performance, and be prepared to reset the statistics to their original values if performance problems arise.

Using a subsystem parameter

This section describes subsystem parameters that influence access path selection. To set subsystem parameters, modify and run installation job DSNTIJUZ. See Part 2 of *DB2 Installation Guide* for information about how to set subsystem parameters. This section contains the following topics:

- "Using a subsystem parameter to favor matching index access"
- "Using a subsystem parameter to optimize queries with IN-list predicates" on page 801

Using a subsystem parameter to favor matching index access

DB2 often does a table space scan or nonmatching index scan when the data access statistics indicate that a table is small, even though matching index access is possible. This is a problem if the table is small or empty when statistics are collected, but the table is large when it is queried. In that case, the statistics are not accurate and can lead DB2 to pick an inefficient access path.

The best solution to the problem is to run RUNSTATS again after the table is populated. However, if you cannot do that, you can use subsystem parameter NPGTHRSH to cause DB2 to favor matching index access over a table space scan and over nonmatching index access.

The value of NPGTHRSH is an integer that indicates the tables for which DB2 favors matching index access. Values of NPGTHRSH and their meanings are:

–1 DB2 favors matching index access for all tables.

0 DB2 selects the access path based on cost, and no tables qualify for special handling. This is the default.

*n***>=1** If data access statistics have been collected for all tables, DB2 favors matching index access for tables for which the total number of pages on which rows of the table appear (NPAGES) is less than *n*.

Tables with default statistics for NPAGES (NPAGES =-1) are presumed to have 501 pages. For such tables, DB2 will favor matching index access only when NPGTHRSH is set above 501.

Recommendation: Before you use NPGTHRSH, be aware that in some cases, matching index access can be more costly than a table space scan or nonmatching index access. Specify a small value for NPGTHRSH (10 or less), which limits the number of tables for which DB2 favors matching index access. If you need to use matching index access only for specific tables, create or alter those tables with the VOLATILE parameter, rather than using the system-wide NPGTHRSH parameter. See "Favoring index access" on page 792.

Using a subsystem parameter to optimize queries with IN-list predicates

You can use the INLISTP parameter to control IN-list predicate optimization. If you set the INLISTP parameter to a number n that is between 1 and 5000, DB2 optimizes for an IN-list predicate with up to n values. If you set the INLISTP predicate to zero, the optimization is disabled. The default value for the INLISTP parameter is 50.

When you enable the INLISTP parameter, you enable two primary means of optimizing some queries that contain IN-list predicates:

- The IN-list predicate is pushed down from the parent query block into the materialized table expression.
- A correlated IN-list predicate in a subquery that is generated by transitive closure is moved up to the parent query block.

Giving optimization hints to DB2

This section describes how experienced programmer analysts can tell DB2 how to process a query. You do this by giving DB2 *hints*. The process of giving hints to DB2 is relatively simple but determining what those hints should be is not. Always test and analyze the results of any query that uses optimization hints.

Giving optimization hints to DB2 is useful in the following situations:

- You want to ensure consistency of response times across rebinds and across release migrations. When a plan or package is rebound, the access path is reformulated. If the database or application has changed, or if DB2 has new function that causes it to choose a different access path, it is handy to have the ability to use an old access path if the new one does not perform as well.

  For this reason, it is a good idea to save the access paths of your critical queries before migrating to a new release of DB2.
- You want to temporarily bypass the access path chosen by DB2.

This section describes the following topics:

- "Planning to use optimization hints"
- "Enabling optimization hints for the subsystem" on page 802
- "Scenario: Preventing a change at rebind" on page 802
- "Scenario: Modifying an existing access path" on page 803
- "Reasons to use the QUERYNO clause" on page 805
- "How DB2 validates the hint" on page 805
- "Limitations on optimization hints" on page 807

Planning to use optimization hints

Before you can give hints to DB2, make sure your PLAN_TABLE is of the correct format. The steps are:

1. Migrate your existing authid.PLAN_TABLE to the 49-column format described in Figure 102 on page 931.

2. For best performance, create an ascending index on the following columns of PLAN_TABLE:
   - QUERYNO
   - APPLNAME
   - PROGNAME
   - VERSION
   - COLLID
   - OPTHINT

   The DB2 sample library, in member DSNTESC, contains an appropriate CREATE INDEX statement that you can modify.

Enabling optimization hints for the subsystem

On the subsystem where the application is bound or where dynamic queries are prepared, specify YES in the OPTIMIZATION HINTS field of installation panel DSNTIP4. If you specify NO, DB2 ignores any hints.

Scenario: Preventing a change at rebind

The following scenario assumes that DB2 is using an access path that you like for a particular query and that there is currently a row in the PLAN_TABLE for that desired access path. By making that access path a hint, DB2 will use your hint when you rebind or migrate to a new release.

1. Determine the query number that currently exists for that query in the PLAN_TABLE. To ensure that the query number is always correlated with that query in the application, modify the statement in the application to use the QUERYNO clause. (If you want to use some kind of numbering convention for queries that use access path hints, you can change the query number in PLAN_TABLE. The important thing is to have the query in the application have a query number that is unique for that application and that matches the QUERYNO value in the PLAN_TABLE.)

   Here is an example of the QUERYNO clause:

   ```
   SELECT * FROM T1
     WHERE C1 = 10 AND
     C2 BETWEEN 10 AND 20 AND
     C3 NOT LIKE 'A%'
     QUERYNO 100;
   ```

   For more information about reasons to use the QUERYNO clause, see “Reasons to use the QUERYNO clause” on page 805.
2. Make the PLAN_TABLE rows for that query (QUERYNO=100) into a hint by updating the OPTHINT column with the name you want to call the hint. In this case, the name is OLDPATH:

   ```
   UPDATE PLAN_TABLE
     SET OPTHINT = 'OLDPATH' WHERE
     QUERYNO = 100 AND
      APPLNAME = ' ' AND
      PROGNAME = 'DSNTEP2' AND
      VERSION = '' AND
      COLLID = 'DSNTEP2';
   ```
3. Tell DB2 to use the hint, and indicate in the PLAN_TABLE that DB2 used the hint.
   - For dynamic SQL statements in the program, follow these steps:
     a. Execute the SET CURRENT OPTIMIZATION HINT statement in the program to tell DB2 to use OLDPATH. For example:

        ```
        SET CURRENT OPTIMIZATION HINT = 'OLDPATH';
        ```

If you do not explicitly set the CURRENT OPTIMIZATION HINT special register, the value that you specify for the bind option OPTHINT is used.

If you execute the SET CURRENT OPTIMIZATION HINT statement statically, rebind the plan or package to pick up the SET CURRENT OPTIMIZATION HINT statement.

b. Execute the EXPLAIN statement on the SQL statements for which you have instructed DB2 to use OLDPATH. This step adds rows to the PLAN_TABLE for those statements. The rows contain a value of OLDPATH in the HINT_USED column.

If DB2 uses all of the hints that you provided, it returns SQLCODE +394 from the PREPARE of the EXPLAIN statement and from the PREPARE of SQL statements that use the hints. If any of your hints are invalid, or if any duplicate hints were found, DB2 issues SQLCODE +395. If DB2 does not find an optimization hint, DB2 returns another SQLCODE. Usually, this SQLCODE is 0.

| If the dynamic statement cache is enabled, DB2 does not use the hint to optimize a dynamic SQL statement. If the statement executes successfully in this case, DB2 usually returns SQLCODE 0.

- For static SQL statements in the program, rebind the plan or package that contains the statements. Specify bind options EXPLAIN(YES) and OPTHINT('OLDPATH') to add rows for those statements in the PLAN_TABLE that contain a value of OLDPATH in the HINT_USED column.

  If DB2 uses the hint you provided, it returns SQLCODE +394 from the rebind. If your hints are invalid, DB2 issues SQLCODE +395. If DB2 does not find an optimization hint, it returns another SQLCODE. Usually, this SQLCODE is 0.

4. Select from PLAN_TABLE to see what was used:

```
SELECT *
  FROM PLAN_TABLE
   WHERE QUERYNO = 100
  ORDER BY TIMESTAMP, QUERYNO, QBLOCKNO, PLANNO, MIXOPSEQ;
```

The PLAN_TABLE in Table 132 shows the OLDPATH hint, indicated by a value in OPTHINT and it also shows that DB2 used that hint, indicated by OLDPATH in the HINT_USED column.

Table 132. PLAN_TABLE that shows that the OLDPATH optimization hint is used.

QUERYNO	METHOD	TNAME	OPTHINT	HINT_USED
100	0	EMP	OLDPATH	
100	4	EMPPROJACT	OLDPATH	
100	3		OLDPATH	
100	0	EMP		OLDPATH
100	4	EMPPROJECT		OLDPATH
100	3			OLDPATH

Scenario: Modifying an existing access path

The following scenario assumes that DB2 is using a hybrid join where you know it can perform better with a sort merge join. The example assumes that the query is dynamic.

1. Put the old access path in the PLAN_TABLE and associate it with a query number.

```
EXPLAIN ALL SET QUERYNO=200 FOR
SELECT X.ACTNO, X.PROJNO, X.EMPNO, Y.JOB, Y.EDLEVEL
  FROM DSN8610.EMPPROJACT X, DSN8610.EMP Y
 WHERE X.EMPNO = Y.EMPNO
   AND X.EMPTIME > 0.5
   AND (Y.JOB = 'DESIGNER' OR Y.EDLEVEL >= 12)
 ORDER BY X.ACTNO, X.PROJNO;
```

2. Make the PLAN_TABLE rows into a hint by updating the OPTHINT column with the name you want to call the hint. In this case, the name is NOHYB:

```
UPDATE PLAN_TABLE
  SET OPTHINT = 'NOHYB' WHERE
  QUERYNO = 200 AND
   APPLNAME = ' ' AND
   PROGNAME = 'DSNTEP2' AND
   VERSION = '' AND
   COLLID = 'DSNTEP2';
```

3. Change the access path so that merge scan join is used rather than hybrid join:

```
UPDATE PLAN_TABLE
  SET METHOD = 2 WHERE
  QUERYNO = 200 AND
  APPLNAME = ' ' AND
  PROGNAME = 'DSNTEP2' AND
  VERSION = '' AND
  COLLID = 'DSNTEP2' AND
  OPTHINT = 'NOHYB' AND
  METHOD = 4;
```

4. Tell DB2 to look for the NOHYB hint for this query:

```
SET CURRENT OPTIMIZATION HINT = 'NOHYB';
```

5. Explain the query again to check for the results:

```
EXPLAIN ALL SET QUERYNO=200 FOR
SELECT X.ACTNO, X.PROJNO, X.EMPNO, Y.JOB, Y.EDLEVEL
  FROM DSN8610.EMPPROJACT X, DSN8610.EMP Y
 WHERE X.EMPNO = Y.EMPNO
   AND X.EMPTIME > 0.5
   AND (Y.JOB = 'DESIGNER' OR Y.EDLEVEL >= 12)
 ORDER BY X.ACTNO, X.PROJNO;
```

6. Select from the PLAN_TABLE to verify the results:

```
SELECT *
  FROM PLAN_TABLE
   WHERE QUERYNO = 200
  ORDER BY TIMESTAMP, QUERYNO, QBLOCKNO, PLANNO, MIXOPSEQ;
```

The PLAN_TABLE in Table 133 shows the NOHYB hint, indicated by a value in OPTHINT and it also shows that DB2 used that hint, indicated by NOHYB in the HINT_USED column.

Table 133. PLAN_TABLE that shows that the NOHYB optimization hint is used.

QUERYNO	METHOD	TNAME	OPTHINT	HINT_USED
200	0	EMP	NOHYB	
200	2	EMPPROJACT	NOHYB	
200	3		NOHYB	
200	0	EMP		NOHYB
200	2	EMPPROJECT		NOHYB
200	3			NOHYB

7. Analyze the performance of the statement to see if it is acceptable.

Reasons to use the QUERYNO clause

You do not need to assign a query number to use optimization hints. If you don't assign a query number, DB2 uses the statement number. However, assigning a query number is especially useful in the following cases:

- For dynamic statements

  The query number for dynamic applications is the statement number in the application *where the prepare occurs*. For some applications, such as DSNTEP2, the same statement in the application prepares each dynamic statement, resulting in the same query number for each dynamic statement. Assigning a query number to each statement that uses optimization hints eliminates ambiguity as to which rows in the PLAN_TABLE are associated with each query.

- For static statements

  If you change an application that has static statements, the statement number might change, causing rows in the PLAN_TABLE to be out of sync with the modified application. Statements that use the QUERYNO clause are not dependent on the statement number. You can move those statements around without affecting the relationship between rows in the PLAN_TABLE and the statements that use those rows in the application.

How DB2 locates the PLAN_TABLE rows for a hint

DB2 uses the QUERYNO, APPLNAME, PROGNAME, VERSION, COLLID, and OPTHINT columns of the PLAN_TABLE to determine the rows to use for a hint. For a PLAN_TABLE row, the QUERYNO, APPLNAME, PROGNAME, VERSION, and COLLID values must match the corresponding values for an SQL statement before the SQL statement can use that row. In addition, the OPTHINT value for that row must match the value in the CURRENT OPTIMIZATION HINT special register if the SQL statement is executed dynamically. If the SQL statement is executed statically, the OPTHINT value for the row must match the value of bind option OPTHINT for the package or plan that contains the SQL statement. If no PLAN_TABLE rows meet these conditions, DB2 determines the access path for the SQL statement without using hints.

How DB2 validates the hint

When you specify an optimization hint by using the OPTHINT bind option for static statements, or the CURRENT OPTIMIZATION HINT special register for dynamic statements, DB2 ensures that you choose a valid access path. To ensure a valid access path, DB2 validates the PLAN_TABLE rows in which the value in the OPTHINT column matches the specified hint. If the access path that you specify has major problems, DB2 invalidates all hints for that query block. In that event, DB2 determines the access path as it normally does.

DB2 validates the information in **only** the PLAN_TABLE columns in Table 134.

Table 134. PLAN_TABLE columns that DB2 validates

Column	Correct values or other explanation
METHOD	Must be 0, 1, 2, 3, or 4. Any other value invalidates the hints. See "Interpreting access to two or more tables (join)" on page 955 for more information about join methods.
CREATOR and TNAME	Must be specified and must name a table, materialized view, materialized nested table expression. Blank if method is 3. If a table is named that does not exist or is not involved in the query, then the hints are invalid.

Table 134. PLAN_TABLE columns that DB2 validates (continued)

Column	Correct values or other explanation
TABNO	Required only if CREATOR, TNAME, and CORRELATION_NAME do not uniquely identify the table. This situation might occur when the same table is used in multiple views (with the same CORRELATION_NAME). This field is ignored if it is not needed.
ACCESSTYPE	Must contain I, I1, N, M, R, RW, T, or V. Any other value invalidates the hints. Values of I, I1, and N all mean single index access. DB2 determines which of the three values to use based on the index specified in ACCESSNAME. M indicates multiple index access. DB2 uses only the first row in the authid.PLAN_TABLE for multiple index access (MIXOPSEQ=0). The choice of indexes, and the AND and OR operations involved, is determined by DB2. If multiple index access isn't possible, then the hints are invalidated. See "Is access through an index? (ACCESSTYPE is I, I1, N or MX)" on page 940 and "Is access through more than one index? (ACCESSTYPE=M)" on page 940 for more information.
ACCESSCREATOR and ACCESSNAME	Ignored if ACCESSTYPE is R or M. If ACCESSTYPE is I, I1, or N, then these fields must identify an index on the specified table. If the index doesn't exist, or if the index is defined on a different table, then the hints are invalid. Also, if the specified index can't be used, then the hints are invalid.
SORTN_JOIN and SORTC_JOIN	Must be Y, N or blank. Any other value invalidates the hints. This value determines if DB2 should sort the new (SORTN_JOIN) or composite (SORTC_JOIN) table. This value is ignored if the specified join method, join sequence, access type and access name dictate whether a sort of the new or composite tables is required. See "Are sorts performed?" on page 945 for more information.
PREFETCH	Must be S, L or blank. Any other value invalidates the hints. This value determines whether DB2 should use sequential prefetch (S), list prefetch (L), or no prefetch (blank). (A blank does not prevent sequential detection at run time.) This value is ignored if the specified access type and access name dictates the type of prefetch required.
PAGE_RANGE	Must be Y, N or blank. Any other value invalidates the hints. See "Was a scan limited to certain partitions? (PAGE_RANGE=Y)" on page 944 for more information.

Table 134. PLAN_TABLE columns that DB2 validates (continued)

Column	Correct values or other explanation
PARALLELISM_MODE	This value is used only if it is possible to run the query in parallel; that is, the SET CURRENT DEGREE special register contains ANY, or the plan or package was bound with DEGREE(ANY). If parallelism is possible, this value must be I, C, X or null. All of the restrictions involving parallelism still apply when using access path hints. If the specified mode cannot be performed, the hints are either invalidated or the mode is modified by the optimizer, possibly resulting in the query being run sequentially. If the value is null then the optimizer determines the mode. See Chapter 35, "Parallel operations and query performance," on page 987 for more information.
ACCESS_DEGREE or JOIN_DEGREE	If PARALLELISM_MODE is specified, use this field to specify the degree of parallelism. If you specify a degree of parallelism, this must a number greater than zero, and DB2 might adjust the parallel degree from what you set here. If you want DB2 to determine the degree, do not enter a value in this field. If you specify a value for ACCESS_DEGREE or JOIN_DEGREE, you must also specify a corresponding ACCESS_PGROUP_ID and JOIN_PGROUP_ID.
WHEN_OPTIMIZE	Must be R, B, or blank. Any other value invalidates the hints. When a statement in a plan that is bound with REOPT(ALWAYS) qualifies for reoptimization at run time, and you have provided optimization hints for that statement, the value of WHEN_OPTIMIZE determines whether DB2 reoptimizes the statement at run time. If the value of WHEN_OPTIMIZE is blank or B, DB2 uses only the access path that is provided by the optimization hints at bind time. If the value of WHEN_OPTIMIZE is R, DB2 determines the access path at bind time using the optimization hints. At run time, DB2 searches the PLAN_TABLE for hints again, and if hints for the statement are still in the PLAN_TABLE and are still valid, DB2 optimizes the access path using those hints again.
PRIMARY_ACCESSTYPE	Must be D or blank. Any other value invalidates the hints.

Limitations on optimization hints

Optimization hints cannot force or undo query transformations, such as subquery transformation to join or materialization or merge of a view or table expression.

A query that is not transformed in one release of DB2 might be transformed in a later release of DB2. Therefore, if you include a hint in one release of DB2 for a query that is not transformed in that release, but in a later release of DB2, the query is transformed, DB2 does not use the hint in the later release.

Chapter 31. Improving concurrency

This chapter begins with an overview of concurrency and locks in the following sections:

- "Definitions of concurrency and locks" on page 810
- "Effects of DB2 locks" on page 810
- "Basic recommendations to promote concurrency" on page 812

After the basic recommendations, the chapter covers some of the major techniques that DB2 uses to control concurrency:

- **Transaction locks** mainly control access by SQL statements. Those locks are the ones over which you have the most control.
  - "Aspects of transaction locks" on page 817 describes the various types of transaction locks that DB2 uses and how they interact.
  - "Options for tuning locks" on page 832 describes what you can change to control locking. Your choices include:
    - "IRLM startup procedure options" on page 832
    - "Installation options for wait times" on page 833
    - "Other options that affect locking" on page 837
    - "Bind options" on page 843
    - "Isolation overriding with SQL statements" on page 857
    - "The LOCK TABLE statement" on page 859

  Under those headings, *lock* (with no qualifier) refers to *transaction lock*.
- **LOB locks** are described separately from transaction locks because the purpose of LOB locks is different than that of regular transition locks. See "LOB locks" on page 860.
- **Latches** are conceptually similar to locks in that they control serialization. They can improve concurrency because they are usually held for shorter duration than locks and they cannot "deadlatch". However, latches can wait, and this wait time is reported in accounting trace class 3. Because latches are not under your control, they are not described in any detail.
- **Claims and drains** provide another mechanism to control serialization. SQL applications and some utilities make claims for objects when they first access them. Operations that drain can take control of an object by quiescing the existing claimers and preventing new claims. After a drainer completes its operations, claimers can resume theirs. DB2 utilities, commands, and some SQL statements can act as drainers.

  "Claims and drains for concurrency control" on page 864 describes claims and drains in more detail and explains how to plan utility jobs and other activities to maximize efficiency.

The chapter also describes how you can monitor DB2's use of locks and how you can analyze a sample problem in "Monitoring of DB2 locking" on page 869.

Finally, "Deadlock detection scenarios" on page 877 shows how to detect deadlock situations.

DB2 extends its concurrency controls to multiple subsystems for data sharing. For information about that, see *DB2 Data Sharing: Planning and Administration*.

© Copyright IBM Corp. 1982, 2008

Definitions of concurrency and locks

Definition: *Concurrency* is the ability of more than one application process to access the same data at essentially the same time.

Example: An application for order entry is used by many transactions simultaneously. Each transaction makes inserts in tables of invoices and invoice items, reads a table of data about customers, and reads and updates data about items on hand. Two operations on the same data, by two simultaneous transactions, might be separated only by microseconds. To the users, the operations appear concurrent.

Conceptual background: Concurrency must be controlled to prevent lost updates and such possibly undesirable effects as unrepeatable reads and access to uncommitted data.

Lost updates. Without concurrency control, two processes, A and B, might both read the same row from the database, and both calculate new values for one of its columns, based on what they read. If A updates the row with its new value, and then B updates the same row, A's update is lost.

Access to uncommitted data. Also without concurrency control, process A might update a value in the database, and process B might read that value before it was committed. Then, if A's value is not later committed, but backed out, B's calculations are based on uncommitted (and presumably incorrect) data.

Unrepeatable reads. Some processes require the following sequence of events: A reads a row from the database and then goes on to process other SQL requests. Later, A reads the first row again and must find the same values it read the first time. Without control, process B could have changed the row between the two read operations.

To prevent those situations from occurring unless they are specifically allowed, DB2 might use *locks* to control concurrency.

What do locks do? A lock associates a DB2 resource with an application process in a way that affects how other processes can access the same resource. The process associated with the resource is said to "hold" or "own" the lock. DB2 uses locks to ensure that no process accesses data that has been changed, but not yet committed, by another process.

What do you do about locks? To preserve data integrity, your application process acquires locks implicitly, that is, under DB2 control. It is not necessary for a process to request a lock explicitly to conceal uncommitted data. Therefore, sometimes you need not do anything about DB2 locks. Nevertheless processes acquire, or avoid acquiring, locks based on certain general parameters. You can make better use of your resources and improve concurrency by understanding the effects of those parameters.

Effects of DB2 locks

The effects of locks that you want to minimize are *suspension*, *timeout*, and *deadlock*.

Suspension

Definition: An application process is *suspended* when it requests a lock that is already held by another application process and cannot be shared. The suspended process temporarily stops running.

Order of precedence for lock requests: Incoming lock requests are queued. Requests for lock promotion, and requests for a lock by an application process that already holds a lock on the same object, precede requests for locks by new applications. Within those groups, the request order is "first in, first out".

Example: Using an application for inventory control, two users attempt to reduce the quantity on hand of the same item at the same time. The two lock requests are queued. The second request in the queue is suspended and waits until the first request releases its lock.

Effects: The suspended process resumes running when:

- All processes that hold the conflicting lock release it.
- The requesting process times out or deadlocks and the process resumes to deal with an error condition.

Timeout

Definition: An application process is said to *time out* when it is terminated because it has been suspended for longer than a preset interval.

Example: An application process attempts to update a large table space that is being reorganized by the utility REORG TABLESPACE with SHRLEVEL NONE. It is likely that the utility job will not release control of the table space before the application process times out.

Effects: DB2 terminates the process, issues two messages to the console, and returns SQLCODE -911 or -913 to the process (SQLSTATEs '40001' or '57033'). Reason code 00C9008E is returned in the SQLERRD(3) field of the SQLCA. Alternatively, you can use the GET DIAGNOSTICS statement to check the reason code. If statistics trace class 3 is active, DB2 writes a trace record with IFCID 0196.

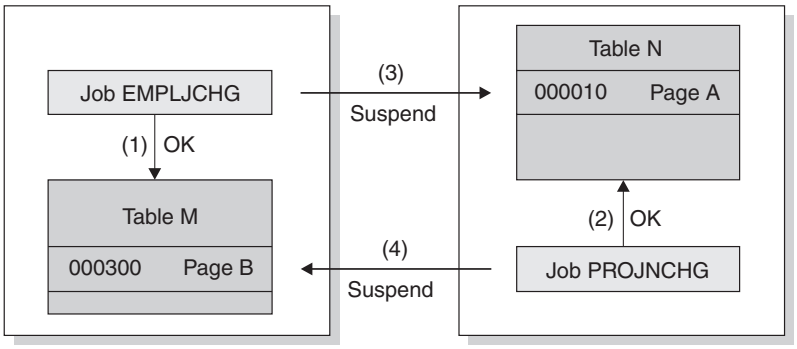
COMMIT and ROLLBACK operations do not time out. The command STOP DATABASE, however, may time out and send messages to the console, but it will retry up to 15 times.

For more information about setting the timeout interval, see "Installation options for wait times" on page 833.

Deadlock

Definition: A *deadlock* occurs when two or more application processes each hold locks on resources that the others need and without which they cannot proceed.

Example: Figure 78 on page 812 illustrates a deadlock between two transactions.



Notes:

1. Jobs EMPLJCHG and PROJNCHG are two transactions. Job EMPLJCHG accesses table M, and acquires an exclusive lock for page B, which contains record 000300.
2. Job PROJNCHG accesses table N, and acquires an exclusive lock for page A, which contains record 000010.
3. Job EMPLJCHG requests a lock for page A of table N while still holding the lock on page B of table M. The job is suspended, because job PROJNCHG is holding an exclusive lock on page A.
4. Job PROJNCHG requests a lock for page B of table M while still holding the lock on page A of table N. The job is suspended, because job EMPLJCHG is holding an exclusive lock on page B. The situation is a deadlock.

Figure 78. A deadlock example

Effects: After a preset time interval (the value of DEADLOCK TIME), DB2 can roll back the current unit of work for one of the processes or request a process to terminate. That frees the locks and allows the remaining processes to continue. If statistics trace class 3 is active, DB2 writes a trace record with IFCID 0172. Reason code 00C90088 is returned in the SQLERRD(3) field of the SQLCA. Alternatively, you can use the GET DIAGNOSTICS statement to check the reason code. (The codes that describe DB2's exact response depend on the operating environment; for details, see Part 5 of *DB2 Application Programming and SQL Guide*.)

It is possible for two processes to be running on distributed DB2 subsystems, each trying to access a resource at the other location. In that case, neither subsystem can detect that the two processes are in deadlock; the situation resolves only when one process times out.

Basic recommendations to promote concurrency

Recommendations are grouped by their scope:
- "Recommendations for system and subsystem options"
- "Recommendations for database design" on page 813
- "Recommendations for application design" on page 814

Recommendations for system and subsystem options

Optimize overall performance: Some performance problems can appear to be locking problems although they are really problems somewhere else in the system. For example, a table space scan of a large table can result in timeouts. Resolve overall system, subsystem, and application performance problems to ensure that you not only eliminate locking symptoms but also correct other underlying performance problems.

Eliminate swapping: If a task is waiting or is swapped out and the unit of work has not been committed, then it still holds locks. When a system is heavily loaded, contention for processing, I/O, and storage can cause waiting. Consider reducing the number of threads or initiators, increasing the priority for the DB2 tasks, and providing more processing, I/O, and real memory.

Recommendations for database design

Keep like things together: Put tables relevant to the same application into the same database. Give each application process that creates private tables a private database. Put tables together in a segmented table space if they are similar in size and can be recovered together.

Keep unlike things apart: Use an adequate number of databases, schema or authorization-ID qualifiers, and table spaces. Concurrency and performance is improved for SQL data definition statements, GRANT statements, REVOKE statements, and utilities. For example, a general guideline is a maximum of 50 tables per database. For more information about lock contention on the catalog, see "Locks on the DB2 catalog" on page 824 and "Utility locks on the catalog and directory" on page 866.

Plan for batch inserts: If your application does sequential batch insertions, excessive contention on the space map pages for the table space can occur. This problem is especially apparent in data sharing, where contention on the space map means the added overhead of page P-lock negotiation. For these types of applications, consider using the MEMBER CLUSTER option of CREATE TABLESPACE. This option causes DB2 to disregard the clustering index (or implicit clustering index) when assigning space for the SQL INSERT statement. For more information about using this option in data sharing, see Chapter 6 of *DB2 Data Sharing: Planning and Administration*. For the syntax, see Chapter 5 of *DB2 SQL Reference*.

Use LOCKSIZE ANY until you have reason not to: LOCKSIZE ANY is the default for CREATE TABLESPACE. It allows DB2 to choose the lock size, and DB2 usually chooses LOCKSIZE PAGE and LOCKMAX SYSTEM for non-LOB table spaces. For LOB table spaces, it chooses LOCKSIZE LOB and LOCKMAX SYSTEM. You should use LOCKSIZE TABLESPACE or LOCKSIZE TABLE only for read-only table spaces or tables, or when concurrent access to the object is not needed. Before you choose LOCKSIZE ROW, you should estimate whether there will be an increase in overhead for locking and weigh that against the increase in concurrency.

Examine small tables: For small tables with high concurrency requirements, estimate the number of pages in the data and in the index. If the index entries are short or they have many duplicates, then the entire index can be one root page and a few leaf pages. In this case, spread out your data to improve concurrency, or consider it a reason to use row locks.

Partition the data: Large tables can be partitioned to take advantage of parallelism for online queries, batch jobs, and utilities. When batch jobs are run in parallel and each job goes after different partitions, lock contention is reduced. In addition, in a data sharing environment, data sharing overhead is reduced when applications that are running on different members go after different partitions.

Partition secondary indexes: The use of data-partitioned secondary indexes (DPSIs) promotes partition independence and, therefore, can reduce lock contention and

improve index availability, especially for utility processing, partition-level operations (such as dropping or rotating partitions), and recovery of indexes.

However, the use of data-partitioned secondary indexes does not always improve the performance of queries. For example, for a query with a predicate that references only the columns of a data-partitioned secondary index, DB2 must probe each partition of the index for values that satisfy the predicate if index access is chosen as the access path. Therefore, take into account data access patterns and maintenance practices when deciding to use a data-partitioned secondary index. Replace a nonpartitioned index with a partitioned index only if there are perceivable benefits such as improved data or index availability, easier data or index maintenance, or improved performance.

For examples of how query performance can be improved with data-partitioned secondary indexes, see "Writing efficient queries on tables with data-partitioned secondary indexes" on page 786.

Fewer rows of data per page: By using the MAXROWS clause of CREATE or ALTER TABLESPACE, you can specify the maximum number of rows that can be on a page. For example, if you use MAXROWS 1, each row occupies a whole page, and you confine a page lock to a single row. Consider this option if you have a reason to avoid using row locking, such as in a data sharing environment where row locking overhead can be greater.

Consider volatile tables to ensure index access: If multiple applications access the same table, consider defining the table as VOLATILE. DB2 uses index access whenever possible for volatile tables, even if index access does not appear to be the most efficient access method because of volatile statistics. Because each application generally accesses the rows in the table in the same order, lock contention can be reduced.

Recommendations for application design

Access data in a consistent order: When different applications access the same data, try to make them do so in the same sequence. For example, make both access rows 1,2,3,5 in that order. In that case, the first application to access the data delays the second, but the two applications cannot deadlock. For the same reason, try to make different applications access the same tables in the same order.

Commit work as soon as is practical: To avoid unnecessary lock contention, issue a COMMIT statement as soon as possible after reaching a point of consistency, even in read-only applications. To prevent unsuccessful SQL statements (such as PREPARE) from holding locks, issue a ROLLBACK statement after a failure. Statements issued through SPUFI can be committed immediately by the SPUFI autocommit feature.

Taking commit points frequently in a long running unit of recovery (UR) has the following benefits at the possible cost of more CPU usage and log write I/Os:

- Reduces lock contention, especially in a data sharing environment
- Improves the effectiveness of lock avoidance, especially in a data sharing environment
- Reduces the elapsed time for DB2 system restart following a system failure
- Reduces the elapsed time for a unit of recovery to rollback following an application failure or an explicit rollback request by the application
- Provides more opportunity for utilities, such as online REORG, to break in

Consider using the UR CHECK FREQ field or the UR LOG WRITE CHECK field of installation panel DSNTIPN to help you identify those applications that are not committing frequently. UR CHECK FREQ, which identifies when too many checkpoints have occurred without a UR issuing a commit, is helpful in monitoring overall system activity. UR LOG WRITE CHECK enables you to detect applications that might write too many log records between commit points, potentially creating a lengthy recovery situation for critical tables.

Even though an application might conform to the commit frequency standards of the installation under normal operational conditions, variation can occur based on system workload fluctuations. For example, a low-priority application might issue a commit frequently on a system that is lightly loaded. However, under a heavy system load, the use of the CPU by the application may be pre-empted, and, as a result, the application may violate the rule set by the UR CHECK FREQ parameter. For this reason, add logic to your application to commit based on time elapsed since last commit, and not solely based on the amount of SQL processing performed. In addition, take frequent commit points in a long running unit of work that is read-only to reduce lock contention and to provide opportunities for utilities, such as online REORG, to access the data.

Retry an application after deadlock or timeout: Include logic in a batch program so that it retries an operation after a deadlock or timeout. Such a method could help you recover from the situation without assistance from operations personnel. Field SQLERRD(3) in the SQLCA returns a reason code that indicates whether a deadlock or timeout occurred. Alternatively, you can use the GET DIAGNOSTICS statement to check the reason code.

Close cursors: If you define a cursor using the WITH HOLD option, the locks it needs can be held past a commit point. Use the CLOSE CURSOR statement as soon as possible in your program to cause those locks to be released and the resources they hold to be freed at the first commit point that follows the CLOSE CURSOR statement. Whether page or row locks are held for WITH HOLD cursors is controlled by the RELEASE LOCKS parameter on installation panel DSNTIP4. Closing cursors is particularly important in a distributed environment.

Free locators: If you have executed the HOLD LOCATOR statement, the LOB locator holds locks on LOBs past commit points. Use the FREE LOCATOR statement to release these locks.

Bind plans with ACQUIRE(USE): ACQUIRE(USE), which indicates that DB2 will acquire table and table space locks when the objects are first used and not when the plan is allocated, is the best choice for concurrency. Packages are always bound with ACQUIRE(USE), by default. ACQUIRE(ALLOCATE) can provide better protection against timeouts. Consider ACQUIRE(ALLOCATE) for applications that need gross locks instead of intent locks or that run with other applications that may request gross locks instead of intent locks. Acquiring the locks at plan allocation also prevents any one transaction in the application from incurring the cost of acquiring the table and table space locks. If you need ACQUIRE(ALLOCATE), you might want to bind all DBRMs directly to the plan.

For information about intent and gross locks, see “The mode of a lock” on page 821.

Bind with ISOLATION(CS) and CURRENTDATA(NO) typically: ISOLATION(CS) lets DB2 release acquired row and page locks as soon as possible.

CURRENTDATA(NO) lets DB2 avoid acquiring row and page locks as often as possible. After that, in order of decreasing preference for concurrency, use these bind options:

1. ISOLATION(CS) with CURRENTDATA(YES), when data returned to the application must not be changed before your next FETCH operation.
2. ISOLATION(RS), when data returned to the application must not be changed before your application commits or rolls back. However, you do not care if other application processes insert additional rows.
3. ISOLATION(RR), when data evaluated as the result of a query must not be changed before your application commits or rolls back. New rows cannot be inserted into the answer set.

For more information about the ISOLATION option, see "The ISOLATION option" on page 846.

For updatable static scrollable cursors, ISOLATION(CS) provides the additional advantage of letting DB2 use *optimistic concurrency control* to further reduce the amount of time that locks are held. With optimistic concurrency control, DB2 releases the row or page locks on the base table after it materializes the result table in a temporary global table. DB2 also releases the row lock after each FETCH, taking a new lock on a row only for a positioned update or delete to ensure data integrity. For more information about optimistic concurrency control, see "Advantages and disadvantages of the isolation values" on page 848.

For updatable dynamic scrollable cursors and ISOLATION(CS), DB2 holds row or page locks on the base table (DB2 does not use a temporary global table). The most recently fetched row or page from the base table remains locked to maintain data integrity for a positioned update or delete.

Use ISOLATION(UR) cautiously: UR isolation acquires almost no locks on rows or pages. It is fast and causes little contention, but it reads uncommitted data. Do not use it unless you are sure that your applications and end users can accept the logical inconsistencies that can occur.

Use sequence objects to generate unique, sequential numbers: Using an identity column is one way to generate unique sequential numbers. However, as a column of a table, an identity column is associated with and tied to the table, and a table can have only one identity column. Your applications might need to use one sequence of unique numbers for many tables or several sequences for each table. As a user-defined object, sequences provide a way for applications to have DB2 generate unique numeric key values and to coordinate the keys across multiple rows and tables.

The use of sequences can avoid the lock contention problems that can result when applications implement their own sequences, such as in a one-row table that contains a sequence number that each transaction must increment. With DB2 sequences, many users can access and increment the sequence concurrently without waiting. DB2 does not wait for a transaction that has incremented a sequence to commit before allowing another transaction to increment the sequence again.

Examine multi-row operations: In an application, multi-row inserts, positioned updates, and positioned deletes have the potential of expanding the unit of work. This can affect the concurrency of other users accessing the data. Minimize

contention by adjusting the size of the host-variable-array, committing between inserts, updates, and preventing lock escalation.

Use global transactions: The Resource Recovery Services attachment facility (RRSAF) relies on an z/OS component called Resource Recovery Services (RRS). RRS provides system-wide services for coordinating two-phase commit operations across z/OS products. For RRSAF applications and IMS transactions that run under RRS, you can group together a number of DB2 agents into a single global transaction. A global transaction allows multiple DB2 agents to participate in a single global transaction and thus share the same locks and access the same data. When two agents that are in a global transaction access the same DB2 object within a unit of work, those agents will not deadlock with each other. The following restrictions apply:

- There is no Parallel Sysplex support for global transactions.
- Because each of the "branches" of a global transaction are sharing locks, uncommitted updates issued by one branch of the transaction are visible to other branches of the transaction.
- Claim/drain processing is not supported across the branches of a global transaction, which means that attempts to issue CREATE, DROP, ALTER, GRANT, or REVOKE may deadlock or timeout if they are requested from different branches of the same global transaction.
- LOCK TABLE may deadlock or timeout across the branches of a global transaction.

For information on how to make an agent part of a global transaction for RRSAF applications, see Section 7 of *DB2 Application Programming and SQL Guide*.

Aspects of transaction locks

Transaction locks have the following four basic aspects:

- "The size of a lock"
- "The duration of a lock" on page 820
- "The mode of a lock" on page 821
- "The object of a lock" on page 823

Knowing the aspects helps you understand why a process suspends or times out or why two processes deadlock. To change the situation, you also need to know:

- "DB2 choices of lock types" on page 826

The size of a lock

Definition

The *size* (sometimes *scope* or *level*) of a lock on data in a table describes the amount of data controlled. The possible sizes of locks are table space, table, partition, page, and row. This section contains information about locking for non-LOB data. See "LOB locks" on page 860 for information on locking for LOBs.

Hierarchy of lock sizes

The same piece of data can be controlled by locks of different sizes. A table space lock (the largest size) controls the most data, all the data in an entire table space. A page or row lock controls only the data in a single page or row.

As Figure 79 suggests, row locks and page locks occupy an equal place in the hierarchy of lock sizes.

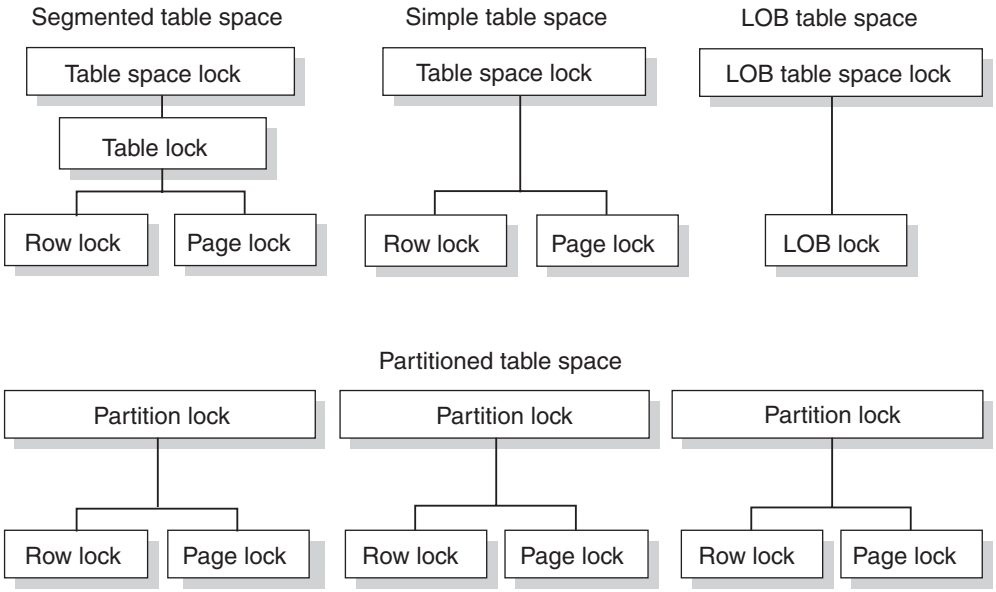


Figure 79. Sizes of objects locked

General effects of size

Locking larger or smaller amounts of data allows you to trade performance for concurrency. Using page or row locks instead of table or table space locks has the following effects:

- Concurrency usually improves, meaning better response times and higher throughput rates for many users.
- Processing time and use of storage increases. That is especially evident in batch processes that scan or update a large number of rows.

Using only table or table space locks has the following effects:

- Processing time and storage usage is reduced.
- Concurrency can be reduced, meaning longer response times for some users but better throughput for one user.

Effects of table spaces of different types

- In a **partitioned table space**, locks are obtained at the partition level. Individual partitions are locked as they are accessed. This locking behavior enables greater concurrency because gross locks (S, U, or X) can be obtained on individual partitions instead of on the entire partitioned table space. (Partition locks are always acquired, even if the table space was defined in a version of DB2 prior to Version 8 with the LOCKPART NO clause. For LOCKPART NO table spaces, DB2 no longer locks the entire table space with one lock when any partition of the table space is accessed.)

  Restrictions: If any of the following conditions are true, DB2 must lock *all* partitions:
  - The plan is bound with ACQUIRE(ALLOCATE).
  - The table space is defined with LOCKSIZE TABLESPACE.
  - The LOCK TABLE statement is used without the PART option.
- A **simple table space** can contain more than one table. A lock on the table space locks all the data in every table. A single page of the table space can contain

rows from every table. A lock on a page locks every row in the page, no matter what tables the data belongs to. Thus, a lock needed to access data from one table can make data from other tables temporarily unavailable. That effect can be partly undone by using row locks instead of page locks.

- In a **segmented table space**, rows from different tables are contained in different pages. Locking a page does not lock data from more than one table. Also, DB2 can acquire a table lock, which locks only the data from one specific table. Because a single row, of course, contains data from only one table, the effect of a row lock is the same as for a simple or partitioned table space: it locks one row of data from one table.
- In a **LOB table space**, pages are not locked. Because there is no concept of a row in a LOB table space, rows are not locked. Instead, LOBs are locked. See "LOB locks" on page 860 for more information.

Differences between simple and segmented table spaces

Figure 80 on page 820 illustrates the difference between the effects of page locks on simple and segmented table spaces. Suppose that tables T1 and T2 reside in table space TS1. In a *simple* table space, a single page can contain rows from both T1 and T2. If User 1 and User 2 acquire incompatible locks on different pages, such as exclusive locks for updating data, neither can access all the rows in T1 and T2 until one of the locks is released. (User 1 and User 2 can both hold a page lock on the same page when the mode of the locks are compatible, such as locks for reading data.)

As the figure also shows, in a *segmented* table space, a table lock applies only to segments assigned to a single table. Thus, User 1 can lock all pages assigned to the segments of T1 while User 2 locks all pages assigned to segments of T2. Similarly, User 1 can lock a page of T1 without locking any data in T2.

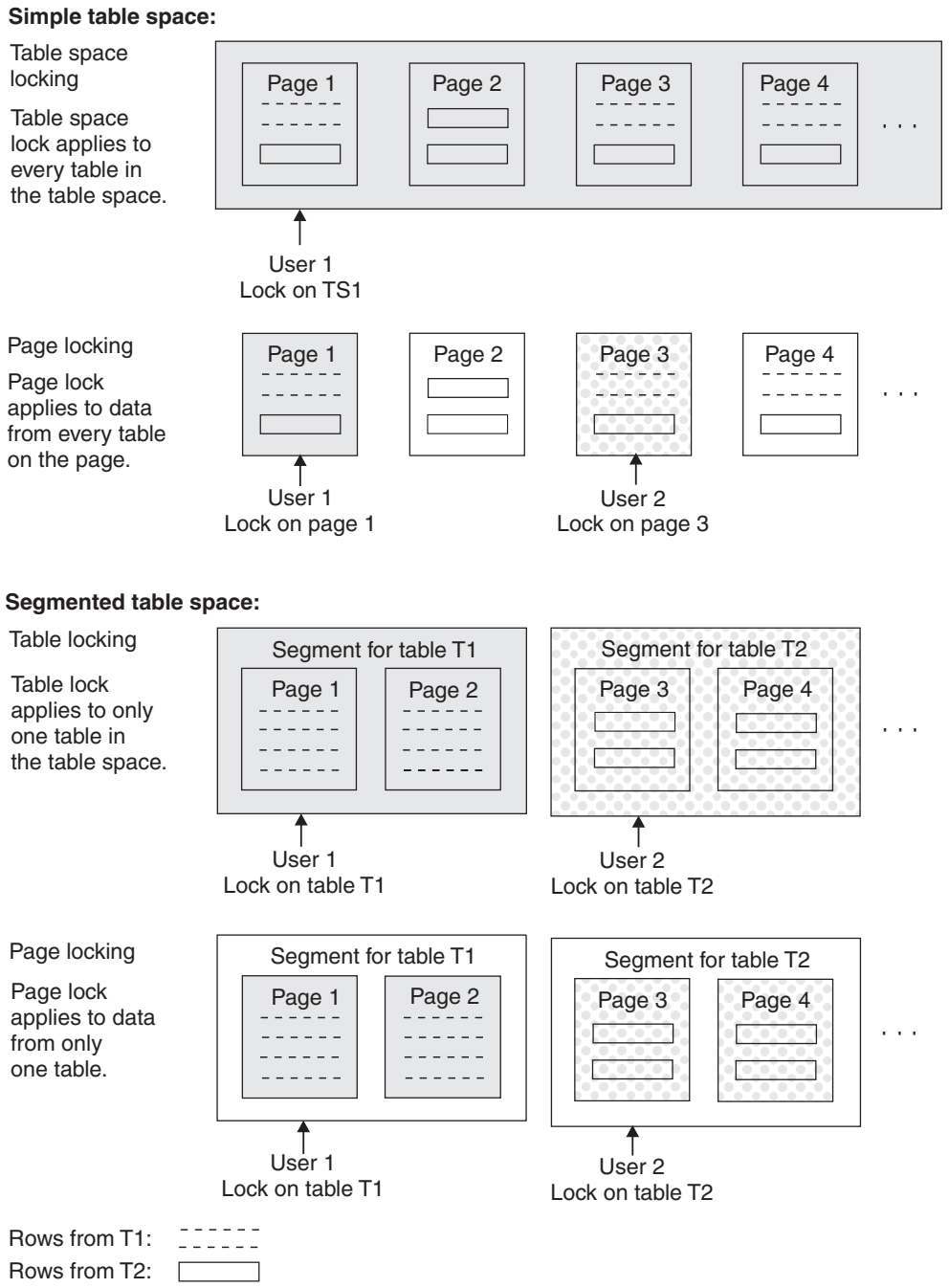


Figure 80. Page locking for simple and segmented table spaces

For information about controlling the size of locks, see:

- "LOCKSIZE clause of CREATE and ALTER TABLESPACE" on page 838
- "The LOCK TABLE statement" on page 859

The duration of a lock

Definition: The *duration* of a lock is the length of time the lock is held. It varies according to when the lock is acquired and when it is released.

Effects

For maximum concurrency, locks on a small amount of data held for a short duration are better than locks on a large amount of data held for a long duration.

However, acquiring a lock requires processor time, and holding a lock requires storage; thus, acquiring and holding one table space lock is more economical than acquiring and holding many page locks. Consider that trade-off to meet your performance and concurrency objectives.

Duration of partition, table, and table space locks: Partition, table, and table space locks can be acquired when a plan is first allocated, or you can delay acquiring them until the resource they lock is first used. They can be released at the next commit point or be held until the program terminates.

On the other hand, LOB table space locks are always acquired when needed and released at a commit or held until the program terminates. See "LOB locks" on page 860 for information about locking LOBs and LOB table spaces.

Duration of page and row locks: If a page or row is locked, DB2 acquires the lock only when it is needed. When the lock is released depends on many factors, but it is rarely held beyond the next commit point.

For information about controlling the duration of locks, see "Bind options" on page 843 for information about the ACQUIRE and RELEASE, ISOLATION, and CURRENTDATA bind options.

The mode of a lock

Definition: The *mode* (sometimes *state*) of a lock tells what access to the locked object is permitted to the lock owner and to any concurrent processes.

The possible modes for page and row locks and the modes for partition, table, and table space locks are listed in "Modes of page and row locks" and "Modes of table, partition, and table space locks" on page 822. See "LOB locks" on page 860 for more information about modes for LOB locks and locks on LOB table spaces.

When a page or row is locked, the table, partition, or table space containing it is also locked. In that case, the table, partition, or table space lock has one of the *intent* modes: IS, IX, or SIX. The modes S, U, and X of table, partition, and table space locks are sometimes called *gross* modes. In the context of reading, SIX is a gross mode lock because you don't get page or row locks; in this sense, it is like an S lock.

Example: An SQL statement locates John Smith in a table of customer data and changes his address. The statement locks the entire table space in mode IX and the specific row that it changes in mode X.

Modes of page and row locks

Modes and their effects are listed in the order of increasing control over resources.

S (SHARE) The lock owner and any concurrent processes can read, but not change, the locked page or row. Concurrent processes can acquire S or U locks on the page or row or might read data without acquiring a page or row lock.

U (UPDATE) The lock owner can read, but not change, the locked page or row. Concurrent processes can acquire S locks or might read data without acquiring a page or row lock, but no concurrent process can acquire a U lock.

U locks reduce the chance of deadlocks when the lock owner is reading a page or row to determine whether to change it, because the owner can start with the U lock and then promote the lock to an X lock to change the page or row.

X (EXCLUSIVE)
The lock owner can read or change the locked page or row. A concurrent process cannot acquire S, U, or X locks on the page or row. However, a concurrent process, such as those bound with the CURRENTDATA(NO) or ISO(UR) options or running with YES specified for the EVALUNC subsystem parameter, can read the data without acquiring a page or row lock.

Modes of table, partition, and table space locks

Modes and their effects are listed in the order of increasing control over resources.

IS (INTENT SHARE)
The lock owner can read data in the table, partition, or table space, but not change it. Concurrent processes can both read and change the data. The lock owner might acquire a page or row lock on any data it reads.

IX (INTENT EXCLUSIVE)
The lock owner and concurrent processes can read and change data in the table, partition, or table space. The lock owner might acquire a page or row lock on any data it reads; it must acquire one on any data it changes.

S (SHARE)
The lock owner and any concurrent processes can read, but not change, data in the table, partition, or table space. The lock owner does not need page or row locks on data it reads.

U (UPDATE)
The lock owner can read, but not change, the locked data; however, the owner can promote the lock to an X lock and then can change the data. Processes concurrent with the U lock can acquire S locks and read the data, but no concurrent process can acquire a U lock. The lock owner does not need page or row locks.

U locks reduce the chance of deadlocks when the lock owner is reading data to determine whether to change it. U locks are acquired on a table space when locksize is TABLESPACE and the statement is a SELECT with a FOR UPDATE clause. Similarly, U locks are acquired on a table when lock size is TABLE and the statement is a SELECT with a FOR UPDATE clause.

SIX (SHARE with INTENT EXCLUSIVE)
The lock owner can read and change data in the table, partition, or table space. Concurrent processes can read data in the table, partition, or table space, but not change it. Only when the lock owner changes data does it acquire page or row locks.

X (EXCLUSIVE)
The lock owner can read or change data in the table, partition, or table space. A concurrent

process can access the data if the process runs with UR isolation or if data in a partitioned table space is running with CS isolation and CURRENTDATA((NO). The lock owner does not need page or row locks.

Lock mode compatibility

The major effect of the lock mode is to determine whether one lock is compatible with another.

Definition: Locks of some modes do not shut out all other users. Assume that application process A holds a lock on a table space that process B also wants to access. DB2 requests, on behalf of B, a lock of some particular mode. If the mode of A's lock permits B's request, the two locks (or modes) are said to be *compatible*.

Effects of incompatibility: If the two locks are not compatible, B cannot proceed. It must wait until A releases its lock. (And, in fact, it must wait until all existing incompatible locks are released.)

Compatible lock modes: Compatibility for page and row locks is easy to define. Table 135 shows whether page locks of any two modes, or row locks of any two modes, are compatible (Yes) or not (No). No question of compatibility of a page lock with a row lock can arise, because a table space cannot use both page and row locks.

Table 135. Compatibility of page lock and row lock modes

Lock Mode	S	U	X
S	Yes	Yes	No
U	Yes	No	No
X	No	No	No

Compatibility for table space locks is slightly more complex. Table 136 shows whether or not table space locks of any two modes are compatible.

Table 136. Compatibility of table and table space (or partition) lock modes

Lock Mode	IS	IX	S	U	SIX	X
IS	Yes	Yes	Yes	Yes	Yes	No
IX	Yes	Yes	No	No	No	No
S	Yes	No	Yes	Yes	No	No
U	Yes	No	Yes	No	No	No
SIX	Yes	No	No	No	No	No
X	No	No	No	No	No	No

The object of a lock

Definition and examples

The *object* of a lock is the resource being locked.

You might have to consider locks on any of the following objects:

- **User data in target tables**. A *target table* is a table that is accessed specifically in an SQL statement, either by name or through a view. Locks on those tables are the most common concern, and the ones over which you have most control.

- **User data in related tables**. Operations subject to referential constraints can require locks on related tables. For example, if you delete from a parent table, DB2 might delete rows from the dependent table as well. In that case, DB2 locks data in the dependent table as well as in the parent table.

  Similarly, operations on rows that contain LOB values might require locks on the LOB table space and possibly on LOB values within that table space. See "LOB locks" on page 860 for more information.

  If your application uses triggers, any triggered SQL statements can cause additional locks to be acquired.
- **DB2 internal objects**. Most of these you are never aware of, but you might notice the following locks on internal objects:
  - Portions of the **DB2 catalog**. For more information, see "Locks on the DB2 catalog."
  - The **skeleton cursor table** (SKCT) representing an application plan.
  - The **skeleton package table** (SKPT) representing a package. For more information on skeleton tables, see "Locks on the skeleton tables (SKCT and SKPT)" on page 825.
  - The **database descriptor** (DBD) representing a DB2 database. For more information, see "Locks on the database descriptors (DBDs)" on page 825.

Indexes and data-only locking

No index page locks are acquired during processing. Instead, DB2 uses a technique called *data-only locking* to serialize changes. Index page latches are acquired to serialize changes within a page and guarantee that the page is physically consistent. Acquiring page latches ensures that transactions accessing the same index page concurrently do not see the page in a partially changed state.

The underlying data page or row locks are acquired to serialize the reading and updating of index entries to ensure the data is logically consistent, meaning that the data is committed and not subject to rollback or abort. The data locks can be held for a long duration such as until commit. However, the page latches are only held for a short duration while the transaction is accessing the page. Because the index pages are not locked, hot spot insert scenarios (which involve several transactions trying to insert different entries into the same index page at the same time) do not cause contention problems in the index.

A query that uses index-only access might lock the data page or row, and that lock can contend with other processes that lock the data. However, using lock avoidance techniques can reduce the contention. See "Lock avoidance" on page 855 for more information about lock avoidance.

Locks on the DB2 catalog

SQL data definition statements, GRANT statements, and REVOKE statements require locks on the DB2 catalog. If different application processes are issuing these types of statements, catalog contention can occur. You can take action to avoid contention.

Contention within table space SYSDBASE: SQL statements that update the catalog table space SYSDBASE contend with each other when those statements are on the same table space. Those statements are:

- CREATE TABLESPACE, TABLE, and INDEX
- ALTER TABLESPACE, TABLE, INDEX
- DROP TABLESPACE, TABLE, and INDEX

- CREATE VIEW, SYNONYM, and ALIAS
- DROP VIEW and SYNONYM, and ALIAS
- COMMENT ON and LABEL ON
- GRANT and REVOKE of table privileges
- RENAME TABLE
- ALTER VIEW

Recommendation: Reduce the concurrent use of statements that update SYSDBASE for the same table space. When you alter a table or table space, quiesce other work on that object.

Contention independent of databases: The following limitations on concurrency are independent of the referenced database:

- CREATE and DROP statements for a table space or index that uses a storage group contend significantly with other such statements.
- CREATE, ALTER, and DROP DATABASE, and GRANT and REVOKE database privileges all contend with each other and with any other function that requires a database privilege.
- CREATE, ALTER, and DROP STOGROUP contend with any SQL statements that refer to a storage group and with extensions to table spaces and indexes that use a storage group.
- GRANT and REVOKE for plan, package, system, or use privileges contend with other GRANT and REVOKE statements for the same type of privilege and with data definition statements that require the same type of privilege.

Locks on the skeleton tables (SKCT and SKPT)

The SKCT of a plan, or the SKPT of a package, is locked while the plan or package is running. The following operations require incompatible locks on the SKCT or SKPT, whichever is applicable, and cannot run concurrently:

- Binding, rebinding, or freeing the plan or package
- Dropping a resource or revoking a privilege that the plan or package depends on
- In some cases, altering a resource that the plan or package depends on

Locks on the database descriptors (DBDs)

Whether a process locks a target DBD depends largely on whether the DBD is already in the EDM DBD cache.

If the DBD is not in the EDM DBD cache, most processes acquire locks on the database descriptor table space (DBD01). That has the effect of locking the DBD and can cause conflict with other processes.

If the DBD is in the EDM DBD cache, the lock on the DBD depends on the type of process, as shown in Table 137.

Table 137. Contention for locks on a DBD in the EDM DBD cache

Process Type	Process	Lock acquired	Conflicts with process type
1	Static DML statements (SELECT, DELETE, INSERT, UPDATE)[1]	None	None
2	Dynamic DML statements[2]	S	3
3	Data definition statements (ALTER, CREATE, DROP)	X	2,3,4

Table 137. Contention for locks on a DBD in the EDM DBD cache (continued)

Process Type	Process	Lock acquired	Conflicts with process type
4	Utilities	S	3

Notes:

1. Static DML statements can conflict with other processes because of locks on data.
2. If caching of dynamic SQL is turned on, no lock is taken on the DBD when a statement is prepared for insertion in the cache or for a statement in the cache.

DB2 choices of lock types

Overview: For the locks acquired on target tables by different types of SQL data manipulation statements, see:

- "Modes of locks acquired for SQL statements"

The lock acquired because of an SQL statement is not always a constant throughout the time of execution. For two situations in which DB2 can change acquired locks during execution, see:

- "Lock promotion" on page 829
- "Lock escalation" on page 829

For a summary of the locks acquired by other operations, see:

- "Modes of transaction locks for various processes" on page 831

Modes of locks acquired for SQL statements

Table 138 on page 827 shows the modes of locks that a process acquires. The mode depends on:

- The type of processing being done
- The value of LOCKSIZE for the target table
- The isolation level of the plan, package, or statement
- The method of access to data

For details about:

- LOCKSIZE, see "LOCKSIZE clause of CREATE and ALTER TABLESPACE" on page 838
- ISOLATION, see "The ISOLATION option" on page 846
- Access methods, see Chapter 34, "Using EXPLAIN to improve SQL performance," on page 927

Reading the table: The following SQL statements and sample steps provide a way to understand the table that shows the modes of locks.

```
EXEC SQL DELETE FROM DSN8810.EMP WHERE CURRENT OF C1;
```

Use the following sample steps to understand the table:

1. Find the portion of the table that describes DELETE operations using a cursor.
2. Find the row for the appropriate values of LOCKSIZE and ISOLATION. Table space DSN8810 is defined with LOCKSIZE ANY. If the value of ISOLATION was not specifically chosen, it is RR by default.
3. Find the subrow for the expected access method. The operation probably uses the index on employee number. Because the operation deletes a row, it must update the index. Hence, you can read the locks acquired in the subrow for "Index, updated":

- An IX lock on the table space
- An IX lock on the table (but see the step that follows)
- An X lock on the page containing the row that is deleted

4. Check the notes to the entries you use, at the end of the table. For this sample operation, see:
   - Note 2, on the column heading for "Table". If the table is not segmented, there is no separate lock on the table.
   - Note 3, on the column heading for "Data Page or Row". Because LOCKSIZE for the table space is ANY, DB2 can choose whether to use page locks, table locks, or table space locks. Typically it chooses page locks.

Table 138. Modes of locks acquired for SQL statements

			Lock mode		
LOCKSIZE	**ISOLATION**	**Access method[1]**	**Table space[9]**	**Table[2]**	**Data page or row[3]**
Processing statement: SELECT with read-only or ambiguous cursor, or with no cursor. UR isolation is allowed and requires none of these locks.					
TABLESPACE	CS RS RR	Any	S	n/a	n/a
TABLE[2]	CS RS RR	Any	IS	S	n/a
PAGE, ROW, or ANY	CS	Index, any use	IS[4, 10]	IS[4]	S[5]
		Table space scan	IS[4, 10]	IS[4]	S[5]
PAGE, ROW, or ANY	RS	Index, any use	IS[4, 10]	IS[4]	S[5], U[11], or X[11]
		Table space scan	IS[4, 10]	IS[4]	S[5], U[11], or X[11]
PAGE, ROW, or ANY	RR	Index/data probe	IS[4]	IS[4]	S[5], U[11], or X[11]
		Index scan[6]	IS[4] or S	S, IS[4], or n/a	S[5], U[11], X[11], or n/a
		Table space scan[6]	IS[2] or S	S or n/a	n/a
Processing statement: INSERT ... VALUES(...) or INSERT ... fullselect[7]					
TABLESPACE	CS RS RR	Any	X	n/a	n/a
TABLE[2]	CS RS RR	Any	IX	X	n/a
PAGE, ROW, or ANY[12]	CS RS RR	Any	IX	IX	X
Processing statement: UPDATE or DELETE, without cursor. Data page and row locks apply only to selected data.					
TABLESPACE	CS RS RR	Any	X	n/a	n/a
TABLE[2]	CS RS RR	Any	IX	X	n/a
PAGE, ROW, or ANY	CS	Index selection	IX	IX	• For delete: X • For update: U→X
		Index/data selection	IX	IX	U→X
		Table space scan	IX	IX	U→X
PAGE, ROW, or ANY	RS	Index selection	IX	IX	• For update: S or U[8] →X • For delete: [S→X] or X
		Index/data selection	IX	IX	S or U[8] →X
		Table space scan	IX	IX	S or U[8] →X

Table 138. Modes of locks acquired for SQL statements (continued)

			Lock mode		
LOCKSIZE	**ISOLATION**	**Access method[1]**	**Table space[9]**	**Table[2]**	**Data page or row[3]**
PAGE, ROW, or ANY	RR	Index selection	IX	IX	• For update: [S or U[8] →X] or X • For delete: [S→X] or X
		Index/data selection	IX	IX	S or U[8] →X
		Table space scan	IX[2] or X	X or n/a	n/a
Processing statement: SELECT with FOR UPDATE OF. Data page and row locks apply only to selected data.					
TABLESPACE	CS RS RR	Any	U	n/a	n/a
TABLE[2]	CS RS RR	Any	IS or IX	U	n/a
PAGE, ROW, or ANY	CS	Index, any use	IX	IX	U
		Table space scan	IX	IX	U
PAGE, ROW, or ANY	RS	Index, any use	IX	IX	S, U, or X[9]
		Table space scan	IX	IX	S, U, or X[8]
PAGE, ROW, or ANY	RR	Index/data probe	IX	IX	S, U, or X[8]
		Index scan[6]	IX or X	X, IX, or n/a	S, U, X[9], or n/a
		Table space scan[6]	IX[2] or X	X or n/a	S, U, X[8], or n/a
Processing Statement: UPDATE or DELETE with cursor					
TABLESPACE	Any	Any	X	n/a	n/a
TABLE[2]	Any	Any	IX	X	n/a
PAGE, ROW, or ANY	CS, RS, or RR	Index, updated	IX	IX	X
		Index not updated	IX	IX	X

Notes for Table 138 on page 827

1. All access methods are either scan-based or probe-based. Scan-based means the index or table space is scanned for successive entries or rows. Probe-based means the index is searched for an entry as opposed to a range of entries, which a scan does. ROWIDs provide data probes to look for a single data row directly. The type of lock used depends on the backup access method. Access methods may be index-only, data-only, or index-to-data.

   Index-only The index alone identifies qualifying rows and the return data.

   Data-only: The data alone identifies qualifying rows and the return data, such as a table space scan or the use of ROWID for a probe.

   Index-to-data The index is used or the index plus data are used to evaluate the predicate:
   - **Index selection**: index is used to evaluate predicate and data is used to return values.
   - **Index/data selection**: index and data are used to evaluate predicate and data is used to return values.

2. Used for segmented table spaces only.
3. These locks are taken on pages if LOCKSIZE is PAGE or on rows if LOCKSIZE is ROW. When the maximum number of locks per table space (LOCKMAX) is reached, locks escalate to a table lock for tables in a segmented table space, or to a table space lock for tables in a non-segmented table space. Using LOCKMAX 0 in CREATE or ALTER TABLESPACE disables lock escalation.
4. If the table or table space is started for read-only access, DB2 attempts to acquire an S lock. If an incompatible lock already exists, DB2 acquires the IS lock.
5. SELECT statements that do not use a cursor, or that use read-only or ambiguous cursors and are bound with CURRENTDATA(NO), might not require any lock if DB2 can determine that the data to be read is committed. This is known as *lock avoidance*.
6. Even if LOCKMAX is 0, the bind process can promote the lock size to TABLE or TABLESPACE. If that occurs, SQLCODE +806 is issued.
7. The locks listed are acquired on the object into which the insert is made. A subselect acquires additional locks on the objects it reads, as if for SELECT with read-only cursor or ambiguous cursor, or with no cursor.
8. An installation option determines whether the lock is S, U, or X. For a full description, see "The option U LOCK FOR RR/RS" on page 840. If you use the WITH clause to specify the isolation as RR or RS, you can use the USE AND KEEP UPDATE LOCKS option to obtain and hold a U lock instead of an S lock, or you can use the USE AND KEEP EXCLUSIVE LOCKS option to obtain and hold an X lock instead of an S lock.
9. Includes partition locks. Does not include LOB table space locks. See "LOB locks" on page 860 for information about locking LOB table spaces.
10. If the table space is partitioned, locks can be avoided on the partitions.
11. If you use the WITH clause to specify the isolation as RR or RS, you can use the USE AND KEEP UPDATE LOCKS option to obtain and hold a U lock instead of an S lock, or you can use the USE AND KEEP EXCLUSIVE LOCKS option to obtain and hold an X lock instead of an S lock.
12. When a unique index exists on the table that receives the INSERT and the value of a pseudo deleted key is the same as the value of the key to be inserted, an S lock is requested on the data that is referenced by the pseudo deleted key entry to determine whether the deletion is committed.

Lock promotion

Definition: *Lock promotion* is the action of exchanging one lock on a resource for a more restrictive lock on the same resource, held by the same application process.

Example: An application reads data, which requires an IS lock on a table space. Based on further calculation, the application updates the same data, which requires an IX lock on the table space. The application is said to *promote* the table space lock from mode IS to mode IX.

Effects: When promoting the lock, DB2 first waits until any incompatible locks held by other processes are released. When locks are promoted, they are promoted in the direction of increasing control over resources: from IS to IX, S, or X; from IX to SIX or X; from S to X; from U to X; and from SIX to X.

Lock escalation

Definition: *Lock escalation* is the act of releasing a large number of page, row or LOB locks, held by an application process on a single table or table space, to

acquire a table or table space lock, or a set of partition locks, of mode S or X instead. When it occurs, DB2 issues message DSNI031I, which identifies the table space for which lock escalation occurred, and some information to help you identify what plan or package was running when the escalation occurred.

Lock counts are always kept on a table or table space level. For an application process that is accessing LOBs, the LOB lock count on the LOB table space is maintained separately from the base table space, and lock escalation occurs separately from the base table space.

When escalation occurs for a partitioned table space, only partitions that are currently locked are escalated. Unlocked partitions remain unlocked. After lock escalation occurs, any unlocked partitions that are subsequently accessed are locked with a gross lock.

For an application process that is using Sysplex query parallelism, the lock count is maintained on a member basis, not globally across the group for the process. Thus, escalation on a table space or table by one member does not cause escalation on other members.

Example: Assume that a segmented table space is defined with LOCKSIZE ANY and LOCKMAX 2000. DB2 can use page locks for a process that accesses a table in the table space and can escalate those locks. If the process attempts to lock more than 2000 pages in the table at one time, DB2 promotes its intent locks on the table to mode S or X and then releases its page locks.

If the process is using Sysplex query parallelism and a table space that it accesses has a LOCKMAX value of 2000, lock escalation occurs for a member only if more than 2000 locks are acquired for that member.

When it occurs: Lock escalation balances concurrency with performance by using page or row locks while a process accesses relatively few pages or rows, and then changing to table space, table, or partition locks when the process accesses many. When it occurs, lock escalation varies by table space, depending on the values of LOCKSIZE and LOCKMAX, as described in

- "LOCKSIZE clause of CREATE and ALTER TABLESPACE" on page 838
- "LOCKMAX clause of CREATE and ALTER TABLESPACE" on page 839

Lock escalation is suspended during the execution of SQL statements for ALTER, CREATE, DROP, GRANT, and REVOKE.

See "Controlling LOB lock escalation" on page 863 for information about lock escalation for LOBs.

Recommendations: The DB2 statistics and performance traces can tell you how often lock escalation has occurred and whether it has caused timeouts or deadlocks. As a rough estimate, if one quarter of your lock escalations cause timeouts or deadlocks, then escalation is not effective for you. You might alter the table to increase LOCKMAX and thus decrease the number of escalations.

Alternatively, if lock escalation is a problem, use LOCKMAX 0 to disable lock escalation.

Example: Assume that a table space is used by transactions that require high concurrency and that a batch job updates almost every page in the table space. For

high concurrency, you should probably create the table space with LOCKSIZE PAGE and make the batch job commit every few seconds.

LOCKSIZE ANY is a possible choice, if you take other steps to avoid lock escalation. If you use LOCKSIZE ANY, specify a LOCKMAX value large enough so that locks held by transactions are not normally escalated. Also, LOCKS PER USER must be large enough so that transactions do not reach that limit.

If the batch job is:

- *Concurrent* with transactions, then it must use page or row locks and commit frequently: for example, every 100 updates. Review LOCKS PER USER to avoid exceeding the limit. The page or row locking uses significant processing time. Binding with ISOLATION(CS) may discourage lock escalation to an X table space lock for those applications that read a lot and update occasionally. However, this may not prevent lock escalation for those applications that are update intensive.
- *Non-concurrent* with transactions, then it need not use page or row locks. The application could explicitly lock the table in exclusive mode, described under "The LOCK TABLE statement" on page 859.

Modes of transaction locks for various processes

The rows in Table 139 show a sample of several types of DB2 processes. The columns show the most restrictive mode of locks used for different objects and the possible conflicts between application processes.

Table 139. Modes of DB2 transaction locks

Process	Catalog table spaces	Skeleton tables (SKCT and SKPT)	Database descriptor (DBD) (1)	Target table space (2)
Transaction with static SQL	IS (3)	S	n/a (4)	Any (5)
Query with dynamic SQL	IS (6)	S	S	Any (5)
BIND process	IX	X	S	n/a
SQL CREATE TABLE statement	IX	n/a	X	n/a
SQL ALTER TABLE statement	IX	X (7)	X	n/a
SQL ALTER TABLESPACE statement	IX	X (9)	X	n/a
SQL DROP TABLESPACE statement	IX	X (8)	X	n/a
SQL GRANT statement	IX	n/a	n/a	n/a
SQL REVOKE statement	IX	X (8)	n/a	n/a

Notes for Table 139:

1. In a lock trace, these locks usually appear as locks on the DBD.
2. The target table space is one of the following table spaces:
   - Accessed and locked by an application process
   - Processed by a utility
   - Designated in the data definition statement
3. The lock is held briefly to check EXECUTE authority.
4. If the required DBD is not already in the EDM DBD cache, locks are acquired on table space DBD01, which effectively locks the DBD.
5. For details, see Table 138 on page 827.

6. Except while checking EXECUTE authority, IS locks on catalog tables are held until a commit point.
7. The plan or package using the SKCT or SKPT is marked invalid if a referential constraint (such as a new primary key or foreign key) is added or changed, or the AUDIT attribute is added or changed for a table.
8. The plan or package using the SKCT or SKPT is marked invalid as a result of this operation.
9. These locks are not held when ALTER TABLESPACE is changing the following options: PRIQTY, SECQTY, PCTFREE, FREEPAGE, CLOSE, and ERASE.

Options for tuning locks

The following sections describes the options that affect transaction locks:
- "IRLM startup procedure options"
- "Installation options for wait times" on page 833
- "Other options that affect locking" on page 837
- "Bind options" on page 843
- "Isolation overriding with SQL statements" on page 857
- "The LOCK TABLE statement" on page 859

IRLM startup procedure options

When you issue the z/OS command START *irlmproc*, the values of the options are passed to the startup procedure for the DB2 internal resource lock manager (IRLM). (If an option is not explicitly specified on the command, the value of its corresponding installation parameter is used.)

The options relevant to DB2 locking are:

SCOPE Whether IRLM is used for data sharing (GLOBAL) or not (LOCAL). Use LOCAL unless you are using data sharing. If you use data sharing, specify GLOBAL.

DEADLOK The two values of this option specify:
1. The number of seconds between two successive scans for a local deadlock
2. The number of local scans that occur before a scan for global deadlock starts

PC Ignored by IRLM. However, PC is positional and must be maintained in the IRLM for compatibility.

MAXCSA Ignored by IRLM. However, MAXCSA is positional and must be maintained in the IRLM for compatibility.

The maximum amount of storage available for IRLM locks is limited to 90% of the total space given to the IRLM private address space during the startup procedure. The other 10% is reserved for IRLM system services, z/OS system services, and "must complete" processes to prevent the IRLM address space from abending, which would bring down your DB2 system. When the storage limit is reached, lock requests are rejected with an out-of-storage reason code.

You can use the F irlmproc,STATUS,STOR command to monitor the amount of storage that is available for locks and the MODIFY irlmproc,SET command to dynamically change the maximum amount of IRLM private storage to use for locks.

Estimating the storage needed for locks

For a conservative figure, assume:

- 540 bytes of storage for each lock.
- All concurrent threads hold the maximum number of row or page locks (LOCKS PER USER on installation panel DSNTIPJ). The number of table and table space locks is negligible.
- The maximum number of concurrent threads are active.

Then calculate: `Storage = 540 × (LOCKS PER USER) × (MAX USERS)`.

That value is calculated when DB2 is installed.

Installation options for wait times

These options determine how long it takes DB2 to identify that a process must be timed out or is deadlocked. They affect locking in your entire DB2 subsystem.

The following fields of the installation panels are relevant to transaction locks:

- "DEADLOCK TIME on installation panel DSNTIPJ"
- "RESOURCE TIMEOUT on installation panel DSNTIPI"
- "IDLE THREAD TIMEOUT on installation panel DSNTIPR" on page 835

The following field is relevant to drain locks:

- "UTILITY TIMEOUT on installation panel DSNTIPI" on page 836

DEADLOCK TIME on installation panel DSNTIPJ

Effect: DB2 scans for deadlocked processes at regular intervals. This field sets the length of the interval, in seconds.

Default: 5 seconds.

Recommendation: Determining the best value for deadlock detection is dependent on your workload. Deadlock detection can cause latch suspensions; therefore, for systems in which deadlocking is not a problem, have deadlock detection run less frequently for the best performance and concurrency (but don't choose a number greater than 5). However, if your system is prone to deadlocks, you want those detected as quickly as possible. In that case, choose 1.

RESOURCE TIMEOUT on installation panel DSNTIPI

Effect: Specifies a minimum number of seconds before a timeout can occur. A small value can cause a large number of timeouts. With a larger value, suspended processes more often resume normally, but they remain inactive for longer periods.

Default: 60 seconds.

Recommendation: If you can allow a suspended process to remain inactive for 60 seconds, use the defaults for both RESOURCE TIMEOUT and DEADLOCK TIME. To specify a different inactive period, you must consider how DB2 times out a process that is waiting for a transaction lock, as described in "Wait time for transaction locks."

Wait time for transaction locks

In a scanning schedule to determine whether a process waiting for a transaction lock has timed out, DB2 uses the following two factors:

- A timeout period
- An operation multiplier

The timeout period: From the value of RESOURCE TIMEOUT and DEADLOCK TIME, DB2 calculates a timeout period. Assume that DEADLOCK TIME is 5 and RESOURCE TIMEOUT is 18.

1. Divide RESOURCE TIMEOUT by DEADLOCK TIME (18/5 = 3.6). IRLM limits the result of this division to 255.
2. Round the result to the next largest integer (Round up 3.6 to 4).
3. Multiply the DEADLOCK TIME by that integer (4 * 5 = 20).

The result, the timeout period (20 seconds), is always at least as large as the value of RESOURCE TIMEOUT (18 seconds), except when the RESOURCE TIMEOUT divided by DEADLOCK TIME exceeds 255.

The timeout multiplier: Requests from different types of processes wait for different multiples of the timeout period. In a data sharing environment, you can add another multiplier to those processes to wait for retained locks.

In some cases, you can modify the multiplier value. Table 140 indicates the multiplier by type of process, and whether you can change it.

Table 140. Timeout multiplier by type

Type	Multiplier	Modifiable?
IMS MPP, IMS Fast Path Message Processing, CICS, QMF, CAF, TSO batch and online, RRSAF, global transactions	1	No
IMS BMPs	4	Yes
IMS DL/I batch	6	Yes
IMS Fast Path Non-message processing	6	No
BIND subcommand processing	3	No
STOP DATABASE command processing	10	No
Utilities	6	Yes
Retained locks for all types	0	Yes

See "UTILITY TIMEOUT on installation panel DSNTIPI" on page 836 for information about modifying the utility timeout multiplier.

Changing the multiplier for IMS BMP and DL/I batch: You can modify the multipliers for IMS BMP and DL/I batch by modifying the following subsystem parameters on installation panel DSNTIPI:

IMS BMP TIMEOUT
The timeout multiplier for IMS BMP connections. A value from 1 to 254 is acceptable. The default is 4.

DL/I BATCH TIMEOUT
The timeout multiplier for IMS DL/I batch connections. A value from 1 to 254 is acceptable. The default is 6.

Additional multiplier for retained locks: For data sharing, you can specify an additional timeout multiplier to be applied to the connection's normal timeout multiplier. This multiplier is used when the connection is waiting for a retained lock, which is a lock held by a failed member of a data sharing group. A zero means don't wait for retained locks. See *DB2 Data Sharing: Planning and Administration* for more information about retained locks.

The scanning schedule: Figure 81 illustrates the following example of scanning to detect a timeout:

- DEADLOCK TIME has the default value of 5 seconds.
- RESOURCE TIMEOUT was chosen to be 18 seconds. Therefore, the timeout period is 20 seconds.
- A bind operation starts 4 seconds before the next scan. The operation multiplier for a bind operation is 3.

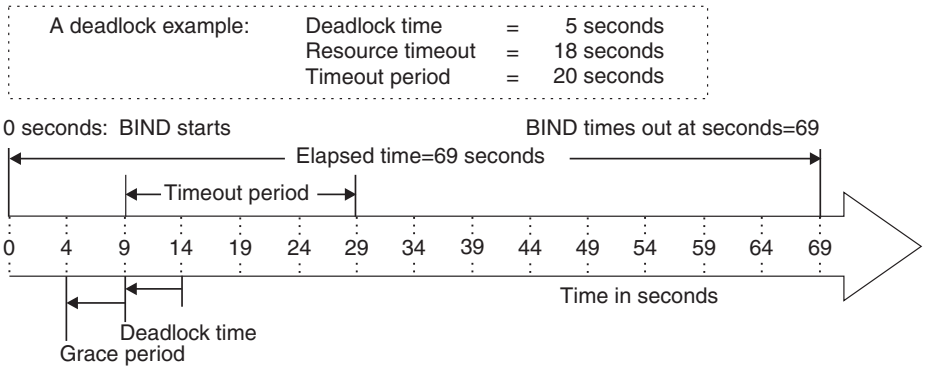


Figure 81. An example of scanning for timeout

The scans proceed through the following steps:

1. A scan starts 4 seconds after the bind operation requests a lock. As determined by the DEADLOCK TIME, scans occur every 5 seconds. The first scan in the example detects that the operation is inactive.
2. IRLM allows at least one full interval of DEADLOCK TIME as a "grace period" for an inactive process. After that, its lock request is judged to be waiting. At 9 seconds, the second scan detects that the bind operation is waiting.
3. The bind operation continues to wait for a multiple of the timeout period. In the example, the multiplier is 3 and the timeout period is 20 seconds. The bind operation continues to wait for 60 seconds longer.
4. The scan that starts 69 seconds after the bind operation detects that the process has timed out.

Effects: An operation can remain inactive for longer than the value of RESOURCE TIMEOUT.

If you are in a data sharing environment, the deadlock and timeout detection process is longer than that for non-data-sharing systems. See *DB2 Data Sharing: Planning and Administration* for more information about global detection processing and elongation of the timeout period.

Recommendation: Consider the length of inaction time when choosing your own values of DEADLOCK TIME and RESOURCE TIMEOUT.

IDLE THREAD TIMEOUT on installation panel DSNTIPR

Effect: Specifies a period for which an active distributed thread can hold locks without doing any processing. After that period, a regular scan (at 3-minute intervals) detects that the thread has been idle for the specified period, and DB2 cancels the thread.

The cancellation applies only to active threads. If your installation permits distributed threads to be inactive and hold no resources, those threads are allowed to remain idle indefinitely.

Default: 0. That value disables the scan to time out idle threads. The threads can then remain idle indefinitely.

Recommendation: If you have experienced distributed users leaving an application idle while it holds locks, pick an appropriate value other than 0 for this period. Because the scan occurs only at 3-minute intervals, your idle threads will generally remain idle for somewhat longer than the value you specify.

UTILITY TIMEOUT on installation panel DSNTIPI

Effect: Specifies an operation multiplier for utilities waiting for a drain lock, for a transaction lock, or for claims to be released.

Default: 6.

Recommendation: With the default value, a utility generally waits longer for a resource than does an SQL application. To specify a different inactive period, you must consider how DB2 times out a process that is waiting for a drain, as described in "Wait time for drains."

Wait time for drains

A process that requests a drain might wait for two events:

1. Acquiring the drain lock. If another user holds the needed drain lock in an incompatible lock mode, then the drainer waits.
2. Releasing all claims on the object. Even after the drain lock is acquired, the drainer waits until all claims are released before beginning to process.

If the process drains more than one claim class, it must wait for those events to occur for *each claim class* it drains.

Hence, to calculate the maximum amount of wait time:

1. Start with the wait time for a drain lock.
2. Add the wait time for claim release.
3. Multiply the result by the number of claim classes drained.

Wait times for drain lock and claim release: Both wait times are based on the timeout period that is calculated in "RESOURCE TIMEOUT on installation panel DSNTIPI" on page 833. For the REORG utility with the SHRLEVEL REFERENCE or SHRLEVEL CHANGE option, you can use utility parameters to specify the wait time for a drain lock and to indicate if additional attempts should be made to acquire the drain lock. For more information, see *DB2 Utility Guide and Reference*.

Drainer	**Each wait time is:**
Utility	(timeout period) * (value of UTILITY TIMEOUT)
Other process	timeout period

Maximum wait time: Because the maximum wait time for a drain lock is the same as the maximum wait time for releasing claims, you can calculate the total maximum wait time as follows:

For **utilities**:

```
2 * (timeout period) * (UTILITY TIMEOUT) * (number of claim classes)
```

For **other processes**:

```
2 * (timeout period) * (operation multiplier) * (number of claim classes)
```

Example: Suppose that LOAD must drain 3 claim classes, that the timeout period is 20 seconds, and that the value of UTILITY TIMEOUT is 6. Use the following calculation to determine how long the LOAD might utility be suspended before being timed out:

```
Maximum wait time = 2 * 20 * 6 * 3 = 720 seconds
```

Wait times less than maximum: The maximum drain wait time is the longest possible time a drainer can wait for a drain, **not** the length of time it always waits.

Example: Table 141 lists the steps LOAD takes to drain the table space and the maximum amount of wait time for each step. A timeout can occur at any step. At step 1, the utility can wait 120 seconds for the repeatable read drain lock. If that lock is not available by then, the utility times out after 120 seconds. It does not wait 720 seconds.

Table 141. Maximum drain wait times: LOAD utility

Step	Maximum Wait Time (seconds)
1. Get repeatable read drain lock	120
2. Wait for all RR claims to be released	120
3. Get cursor stability read drain lock	120
4. Wait for all CS claims to be released	120
5. Get write drain lock	120
6. Wait for all write claims to be released	120
Total	**720**

Other options that affect locking

You can use various options to control such things as how many locks are used and which mode is used for certain locks. This section describes the following installation options and SQL statement clauses:

- "LOCKS PER USER field of installation panel DSNTIPJ"
- "LOCKSIZE clause of CREATE and ALTER TABLESPACE" on page 838
- "LOCKMAX clause of CREATE and ALTER TABLESPACE" on page 839
- "LOCKS PER TABLE(SPACE) field of installation panel DSNTIPJ" on page 840
- "The option U LOCK FOR RR/RS" on page 840
- "Option to release locks for cursors defined WITH HOLD" on page 841
- "Option XLOCK for searched updates/deletes" on page 841
- "Option to avoid locks during predicate evaluation" on page 842
- "Option to disregard uncommitted inserts" on page 842

See "Effects of table spaces of different types" on page 818 for information about the LOCKPART clause of CREATE and ALTER TABLESPACE.

LOCKS PER USER field of installation panel DSNTIPJ

Effect: Specifies the maximum number of page, row, or LOB locks that can be held by a single process at any one time. It includes locks for both the DB2 catalog and directory and for user data.

When a request for a page, row, or LOB lock exceeds the specified limit, it receives SQLCODE -904: "resource unavailable" (SQLSTATE '57011'). The requested lock cannot be acquired until some of the existing locks are released.

Default: 10 000

Recommendation: The default should be adequate for 90 percent of the work load when using page locks. If you use row locks on very large tables, you might want a higher value. If you use LOBs, you might need a higher value.

Review application processes that require higher values to see if they can use table space locks rather than page, row, or LOB locks. The accounting trace shows the maximum number of page, row, or LOB locks a process held while running.

Remember that the value specified is for a single application. Each concurrent application can potentially hold up to the maximum number of locks specified. Do not specify zero or a very large number unless it is required to run your applications.

LOCKSIZE clause of CREATE and ALTER TABLESPACE

The information under this heading, up to "LOCKMAX clause of CREATE and ALTER TABLESPACE" on page 839, is General-use Programming Interface and Associated Guidance Information, as defined in "Notices" on page 1301.

Effect: Specifies the size for locks held on a table or table space by any application process that accesses it. In addition to using the ALTER TABLESPACE statement to change the lock size for user data, you can also change the lock size of any DB2 catalog table space that is neither a LOB table space nor a table space that contains links. The options are:

LOCKSIZE TABLESPACE
A process acquires no table, page, row, or LOB locks within the table space. That improves performance by reducing the number of locks maintained, but greatly inhibits concurrency.

LOCKSIZE TABLE
A process acquires table locks on tables in a segmented table space. If the table space contains more than one table, this option can provide acceptable concurrency with little extra cost in processor resources.

LOCKSIZE PAGE
A process acquires page locks, plus table, partition, or table space locks of modes that permit page locks (IS, IX, or SIX). The effect is not absolute: a process can still acquire a table, partition, or table space lock of mode S or X, without page locks, if that is needed. In that case, the bind process issues a message warning that the lock size has been promoted as described under "Lock promotion" on page 829.

LOCKSIZE ROW
A process acquires row locks, plus table, partition, or table space locks of modes that permit row locks (IS, IX, or SIX). The effect is not absolute: a process can still acquire a table or table space lock of mode S or X, without row locks, if that is needed. In that case, the bind process issues a message warning that the lock size has been promoted as described under "Lock promotion" on page 829.

LOCKSIZE ANY
DB2 chooses the size of the lock, usually LOCKSIZE PAGE.

LOCKSIZE LOB
If a LOB must be accessed, a process acquires LOB locks and the necessary LOB table space locks (IS or IX). This option is valid only for LOB table spaces. See "LOB locks" on page 860 for more information about LOB locking.

DB2 attempts to acquire an S lock on table spaces that are started with read-only access. If the LOCKSIZE is PAGE or ROW, and DB2 cannot get the S lock, it requests an IS lock. If a partition is started with read-only access, DB2 attempts to get an S lock on the partition that is started RO. For a complete description of how the LOCKSIZE clause affects lock attributes, see "DB2 choices of lock types" on page 826.

Default: LOCKSIZE ANY

Catalog record: Column LOCKRULE of table SYSIBM.SYSTABLESPACE.

Recommendation: If you do not use the default, base your choice upon the results of monitoring applications that use the table space. When considering changing the lock size for a DB2 catalog table space, be aware that, in addition to user queries, DB2 internal processes such as bind and authorization checking and utility processing can access the DB2 catalog.

Row locks or page locks? The question of whether to use row or page locks depends on your data and your applications. If you are experiencing contention on data pages of a table space now defined with LOCKSIZE PAGE, consider LOCKSIZE ROW. But consider also the trade-offs.

The resource required to acquire, maintain, and release a row lock is about the same as that required for a page lock. If your data has 10 rows per page, a table space scan or an index scan can require nearly 10 times as much resource for row locks as for page locks. But locking only a row at a time, rather than a page, might reduce the chance of contention with some other process by 90%, especially if access is random. (Row locking is not recommended for sequential processing.)

Lock avoidance is very important when row locking is used. Therefore, use ISOLATION(CS) CURRENTDATA(NO) or ISOLATION(UR) whenever possible. In many cases, DB2 can avoid acquiring a lock when reading data that is known to be committed. Thus, if only 2 of 10 rows on a page contain uncommitted data, DB2 must lock the entire page when using page locks, but might ask for locks on only the 2 rows when using row locks. Then, the resource required for row locks would be only twice as much, not 10 times as much, as that required for page locks.

On the other hand, if two applications update the same rows of a page, and not in the same sequence, then row locking might even increase contention. With page locks, the second application to access the page must wait for the first to finish and might time out. With row locks, the two applications can access the same page simultaneously, and might deadlock while trying to access the same set of rows.

In short, no single answer fits all cases.

LOCKMAX clause of CREATE and ALTER TABLESPACE

The information under this heading, up to "LOCKS PER TABLE(SPACE) field of installation panel DSNTIPJ" on page 840, is General-use Programming Interface and Associated Guidance Information, as defined in "Notices" on page 1301.

Effect: You can specify these values not only for tables of user data but also, by using ALTER TABLESPACE, for tables in the DB2 catalog.

LOCKMAX *n*

Specifies the maximum number of page or row locks that a single application process can hold on the table space before those locks are escalated as described in "Lock escalation" on page 829. For LOB table spaces, this value

specifies the number of LOB locks that the application process can hold before escalating. For an application that uses Sysplex query parallelism, a lock count is maintained on each member.

LOCKMAX SYSTEM
Specifies that *n* is effectively equal to the system default set by the field LOCKS PER TABLE(SPACE) of installation panel DSNTIPJ.

LOCKMAX 0
Disables lock escalation entirely.

Default: The default depends on the value of LOCKSIZE, as shown in Table 142.

Table 142. How the default for LOCKMAX is determined

LOCKSIZE	Default for LOCKMAX
ANY	SYSTEM
other	0

Catalog record: Column LOCKMAX of table SYSIBM.SYSTABLESPACE.

Recommendations: If you do not use the default, base your choice upon the results of monitoring applications that use the table space.

Aim to set the value of LOCKMAX high enough that, when lock escalation occurs, one application already holds so many locks that it significantly interferes with others. For example, if an application holds half a million locks on a table with a million rows, it probably already locks out most other applications. Yet lock escalation can prevent it from potentially acquiring another half million locks.

If you alter a table space from LOCKSIZE PAGE or LOCKSIZE ANY to LOCKSIZE ROW, consider increasing LOCKMAX to allow for the increased number of locks that applications might require.

LOCKS PER TABLE(SPACE) field of installation panel DSNTIPJ

Effect: The value becomes the default value (SYSTEM) for the LOCKMAX clause of the SQL statements CREATE TABLESPACE and ALTER TABLESPACE.

Default: 1000

Recommendation: Use the default or, if you are migrating from a previous release of DB2, continue to use the existing value. The value should be less than the value for LOCKS PER USER, unless the value or LOCKS PER USER is 0. When you create or alter a table space, especially when you alter one to use row locks, use the LOCKMAX clause explicitly for that table space.

The option U LOCK FOR RR/RS

This option, on installation panel DSNTIPI, determines the mode of the lock first acquired on a row or page, table, partition, or table space for certain statements bound with RR or RS isolation. Those statements include:

- SELECT with FOR UPDATE OF

  Table 143 on page 841 shows which mode of lock is held on rows or pages when you specify the SELECT using the WITH RS or WITH RR isolation clause.

Table 143. Which mode of lock is held on rows or pages when you specify the SELECT using the WITH RS or WITH RR isolation clause

Option Value	Lock Mode
USE AND KEEP EXCLUSIVE LOCKS	X
USE AND KEEP UPDATE LOCKS	U
USE AND KEEP SHARE LOCKS	S

- UPDATE and DELETE, without a cursor

  Table 144 shows which mode of lock is held on rows or pages when you specify an update or a delete without a cursor.

Table 144. Which mode of lock is held on rows or pages when you specify an update or a delete without a cursor

Option Value	Lock Mode
NO (default)	S
YES	U or X

The YES option can avoid deadlocks but reduces concurrency.

Option to release locks for cursors defined WITH HOLD

Effect: The RELEASE LOCKS field of installation panel DSNTIP4 lets you indicate that you want DB2 to release a data page or row lock after a COMMIT is issued for cursors defined WITH HOLD. This lock is not necessary for maintaining cursor position.

Default: YES

Recommendation: The default, YES, causes DB2, at commit time, to release the data page or row lock for the row on which the cursor is positioned. This lock is unnecessary for maintaining cursor position. To improve concurrency, specify YES. Specify NO only for those cases in which existing applications rely on that particular data lock.

Option XLOCK for searched updates/deletes

Effect: A subsystem parameter XLKUPDLT lets you disable update lock (ULOCK) on searched UPDATEs and DELETEs so that you do not have to issue a second lock request to upgrade the lock from U to X (exclusive lock) for each updated row.

Default: NO

Recommendation: This feature is primarily beneficial in a data sharing environment. It should be used when most or all searched UPDATEs/DELETEs use an index or can be evaluated by stage 1 processing. When NO is specified, DB2 uses an S lock or a U lock when scanning for qualifying rows. The lock on any qualifying row or page is then upgraded to an X lock before performing the update or delete. For non-qualifying rows or pages, the lock is released if ISOLATION(CS) is used. For ISOLATION(RS) or ISOLATION(RR), an S lock is retained on the row or page until the next commit point. This option is best for achieving the highest rates of concurrency.

When YES is specified, DB2 uses an X lock on rows or pages that qualify during stage 1 processing. With ISOLATION(CS), the lock is released if the row or page is not updated or deleted because it is rejected by stage 2 processing. With

ISOLATION(RR) or ISOLATION(RS), DB2 acquires an X lock on all rows that fall within the range of the selection expression. Thus, a lock upgrade request is not needed for qualifying rows though the lock duration is changed from manual to commit. The lock duration change is not as costly as a lock upgrade.

Option to avoid locks during predicate evaluation

Effect: The EVALUATE UNCOMMITTED field of installation panel DSNTIP4 indicates if predicate evaluation can occur on uncommitted data of other transactions. The option applies only to stage 1 predicate processing that uses table access (table space scan, index-to-data access, and RID list processing) for queries with isolation level RS or CS.

Although this option influences whether predicate evaluation can occur on uncommitted data, it does not influence whether uncommitted data is returned to an application. Queries with isolation level RS or CS will return only committed data. They will never return the uncommitted data of other transactions, even if predicate evaluation occurs on such. If data satisfies the predicate during evaluation, the data is locked as needed, and the predicate is re-evaluated as needed before the data is returned to the application.

A value of NO specifies that predicate evaluation occurs only on committed data (or on the application's own uncommitted changes). NO ensures that all qualifying data is always included in the answer set.

A value of YES specifies that predicate evaluation can occur on uncommitted data of other transactions. With YES, data might be excluded from the answer set. Data that does not satisfy the predicate during evaluation but then, because of undo processing (ROLLBACK or statement failure), reverts to a state that does satisfy the predicate is missing from the answer set. A value of YES enables DB2 to take fewer locks during query processing. The number of locks avoided depends on:

- The query's access path
- The number of evaluated rows that do not satisfy the predicate
- The number of those rows that are on overflow pages

Default: NO

Recommendation: Specify YES to improve concurrency if your applications can tolerate returned data to falsely exclude any data that would be included as the result of undo processing (ROLLBACK or statement failure).

Option to disregard uncommitted inserts

Effect: The SKIP UNCOMM INSERTS field on installation panel DSNTIP8 controls whether uncommitted inserts are ignored. DB2 can handle uncommitted inserts in the following ways:

- DB2 can wait until the INSERT transaction completes (commits or rolls back) and return data accordingly. This is the default option, NO.
- DB2 can ignore uncommitted inserts, which in many cases can improve concurrency. This behavior must be specified as YES.

Default: NO

Recommendation: In general, YES produces greater concurrency and is preferred for most applications. However, the following examples indicate some instances when the default option, NO, is preferred.

Example: Suppose that an initial transaction produces a second transaction. The initial transaction passes information to the second transaction by inserting data into a table that the second transaction reads. In this case, NO should be used.

Example: Suppose that you frequently modify data by deleting the data and inserting the new image of the data. In such cases that avoid UPDATE statements, the default should be used.

Bind options

The information under this heading, up to "Isolation overriding with SQL statements" on page 857, is General-use Programming Interface and Associated Guidance Information, as defined in "Notices" on page 1301.

These options determine when an application process acquires and releases its locks and to what extent it isolates its actions from possible effects of other processes acting concurrently.

These options of bind operations are relevant to transaction locks:
- "The ACQUIRE and RELEASE options"
- "The ISOLATION option" on page 846
- "The CURRENTDATA option" on page 853

The ACQUIRE and RELEASE options

Effects: The ACQUIRE and RELEASE options of bind determine when DB2 locks an object (table, partition, or table space) your application uses and when it releases the lock. (The ACQUIRE and RELEASE options do not affect page, row, or LOB locks.) The options apply to static SQL statements, which are bound before your program executes. If your program executes dynamic SQL statements, the objects they lock are locked when first accessed and released at the next commit point though some locks acquired for dynamic SQL may be held past commit points. See 844.

ACQUIRE(ALLOCATE)
Acquires the lock when the object is allocated. This option is not allowed for BIND or REBIND PACKAGE.

ACQUIRE(USE)
Acquires the lock when the object is first accessed.

RELEASE(DEALLOCATE)
Releases the lock when the object is deallocated (the application ends). The value has no effect on dynamic SQL statements, which always use RELEASE(COMMIT), unless you are using dynamic statement caching. For information about the RELEASE option with dynamic statement caching, see 844. The value also has no effect on packages that are executed on a DB2 server through a DRDA connection with the client system.

RELEASE(COMMIT)
Releases the lock at the next commit point, unless there are held cursors or held locators. If the application accesses the object again, it must acquire the lock again.

Example: An application selects employee names and telephone numbers from a table, according to different criteria. Employees can update their own telephone numbers. They can perform several searches in succession. The application is bound with the options ACQUIRE(USE) and RELEASE(DEALLOCATE), for these reasons:

- The alternative to ACQUIRE(USE), ACQUIRE(ALLOCATE), gets a lock of mode IX on the table space as soon as the application starts, because that is needed if an update occurs. But most uses of the application do not update the table and so need only the less restrictive IS lock. ACQUIRE(USE) gets the IS lock when the table is first accessed, and DB2 promotes the lock to mode IX if that is needed later.
- Most uses of this application do not update and do not commit. For those uses, there is little difference between RELEASE(COMMIT) and RELEASE(DEALLOCATE). But administrators might update several phone numbers in one session with the application, and the application commits after each update. In that case, RELEASE(COMMIT) releases a lock that DB2 must acquire again immediately. RELEASE(DEALLOCATE) holds the lock until the application ends, avoiding the processing needed to release and acquire the lock several times.

Partition locks: Partition locks follow the same rules as table space locks, and **all** partitions are held for the same duration. Thus, if one package is using RELEASE(COMMIT) and another is using RELEASE(DEALLOCATE), all partitions use RELEASE(DEALLOCATE).

The RELEASE option and dynamic statement caching: Generally, the RELEASE option has no effect on dynamic SQL statements with one exception. When you use the bind options RELEASE(DEALLOCATE) and KEEPDYNAMIC(YES), and your subsystem is installed with YES for field CACHE DYNAMIC SQL on installation panel DSNTIP4, DB2 retains prepared SELECT, INSERT, UPDATE, and DELETE statements in memory past commit points. For this reason, DB2 can honor the RELEASE(DEALLOCATE) option for these dynamic statements. The locks are held until deallocation, or until the commit after the prepared statement is freed from memory, in the following situations:

- The application issues a PREPARE statement with the same statement identifier.
- The statement is removed from memory because it has not been used.
- An object that the statement is dependent on is dropped or altered, or a privilege needed by the statement is revoked.
- RUNSTATS is run against an object that the statement is dependent on.

If a lock is to be held past commit and it is an S, SIX, or X lock on a table space or a table in a segmented table space, DB2 sometimes demotes that lock to an intent lock (IX or IS) at commit. DB2 demotes a gross lock if it was acquired for one of the following reasons:

- DB2 acquired the gross lock because of lock escalation.
- The application issued a LOCK TABLE.
- The application issued a mass delete (DELETE FROM ... without a WHERE clause).

For partitioned table spaces, lock demotion occurs for each partition for which there is a lock.

Defaults: The defaults differ for different types of bind operations, as shown in Table 145.

Table 145. Default ACQUIRE and RELEASE values for different bind options

Operation	Default values
BIND PLAN	ACQUIRE(USE) and RELEASE(COMMIT).

Table 145. Default ACQUIRE and RELEASE values for different bind options (continued)

Operation	Default values
BIND PACKAGE	There is no option for ACQUIRE; ACQUIRE(USE) is always used. At the local server the default for RELEASE is the value used by the plan that includes the package in its package list. At a remote server the default is COMMIT.
REBIND PLAN or PACKAGE	The existing values for the plan or package that is being rebound.

Recommendation: Choose a combination of values for ACQUIRE and RELEASE based on the characteristics of the particular application.

The RELEASE option and DDL operations for remote requesters: When you perform DDL operations on behalf of remote requesters and RELEASE(DEALLOCATE) is in effect, be aware of the following condition. When a package that is bound with RELEASE(DEALLOCATE) accesses data at a server, it might prevent other remote requesters from performing CREATE, ALTER, DROP, GRANT, or REVOKE operations at the server.

To allow those operations to complete, you can use the command STOP DDF MODE(SUSPEND). The command suspends server threads and terminates their locks so that DDL operations from remote requesters can complete. When these operations complete, you can use the command START DDF to resume the suspended server threads. However, even after the command STOP DDF MODE(SUSPEND) completes successfully, database resources might be held if DB2 is performing any activity other than inbound DB2 processing. You might have to use the command CANCEL THREAD to terminate other processing and thereby free the database resources.

Advantages and disadvantages of the combinations

Different combinations of bind options have advantages and disadvantages for certain situations.

ACQUIRE(ALLOCATE) / RELEASE(DEALLOCATE): In some cases, this combination can avoid deadlocks by locking all needed resources as soon as the program starts to run. This combination is most useful for a long-running application that runs for hours and accesses various tables, because it prevents an untimely deadlock from wasting that processing.

- All tables or table spaces used in DBRMs bound directly to the plan are locked when the plan is allocated.
- All tables or table spaces are unlocked only when the plan terminates.
- The locks used are the most restrictive needed to execute all SQL statements in the plan regardless of whether the statements are actually executed.
- Restrictive states are not checked until the page set is accessed. Locking when the plan is allocated insures that the job is compatible with other SQL jobs. Waiting until the first access to check restrictive states provides greater availability; however, it is possible that an SQL transaction could:
  - Hold a lock on a table space or partition that is stopped
  - Acquire a lock on a table space or partition that is started for DB2 utility access only (ACCESS(UT))
  - Acquire an exclusive lock (IX, X) on a table space or partition that is started for read access only (ACCESS(RO)), thus prohibiting access by readers

Disadvantages: This combination reduces concurrency. It can lock resources in high demand for longer than needed. Also, the option ACQUIRE(ALLOCATE) turns off selective partition locking; if you are accessing a partitioned table space, all partitions are locked.

Restriction: This combination is not allowed for BIND PACKAGE. Use this combination if processing efficiency is more important than concurrency. It is a good choice for batch jobs that would release table and table space locks only to reacquire them almost immediately. It might even improve concurrency, by allowing batch jobs to finish sooner. Generally, do not use this combination if your application contains many SQL statements that are often not executed.

ACQUIRE(USE) / RELEASE(DEALLOCATE): This combination results in the most efficient use of processing time in most cases.

- A table, partition, or table space used by the plan or package is locked only if it is needed while running.
- All tables or table spaces are unlocked only when the plan terminates.
- The least restrictive lock needed to execute each SQL statement is used, with the exception that if a more restrictive lock remains from a previous statement, that lock is used without change.

Disadvantages: This combination can increase the frequency of deadlocks. Because all locks are acquired in a sequence that is predictable only in an actual run, more concurrent access delays might occur.

ACQUIRE(USE) / RELEASE(COMMIT): This combination is the default combination and provides the greatest concurrency, but it requires more processing time if the application commits frequently.

- A table or table space is locked only when needed. That locking is important if the process contains many SQL statements that are rarely used or statements that are intended to access data only in certain circumstances.
- Table, partition, or table space locks are released at the next commit point unless the cursor is defined WITH HOLD. See "The effect of WITH HOLD for a cursor" on page 857 for more information.
- The least restrictive lock needed to execute each SQL statement is used except when a more restrictive lock remains from a previous statement. In that case, that lock is used without change.

Disadvantages: This combination can increase the frequency of deadlocks. Because all locks are acquired in a sequence that is predictable only in an actual run, more concurrent access delays might occur.

ACQUIRE(ALLOCATE) / RELEASE(COMMIT): This combination is not allowed; it results in an error message from BIND.

The ISOLATION option

Effects: The ISOLATION option Specifies the degree to which operations are isolated from the possible effects of other operations acting concurrently. Based on this information, DB2 releases S and U locks on rows or pages as soon as possible. You can use the following ISOLATION options:

ISOLATION(CS)

Cursor stability: A row or page lock is held only long enough to allow the cursor to move to another row or page. For data that satisfies the search

condition of the application, the lock is held until the application locks the next row or page. For data that does not satisfy the search condition, the lock is immediately released.

The data returned to an application that uses ISOLATION(CS) is committed, but if the application process returns to the same page, another application might have since updated or deleted the data, or might have inserted additional qualifying rows. This is especially true if DB2 returns data from a result table in a work file.

For example, if DB2 has to put an answer set in a result table (such as for a sort), DB2 releases the lock immediately after it puts the row or page in the result table in the work file. Using cursor stability, the base table can change while your application is processing the result of the sort output.

In some cases, DB2 can avoid taking the lock altogether, depending on the value of the CURRENTDATA bind option or the value of the EVALUATE UNCOMMITTED field on installation panel DSNTIP4.

- *Lock avoidance on committed data*: If DB2 can determine that the data it is reading has already been committed, it can avoid taking the lock altogether. For rows that do not satisfy the search condition, this lock avoidance is possible with CURRENTDATA(YES) or CURRENTDATA(NO). For a row that satisfies the search condition on a singleton SELECT, lock avoidance is possible with CURRENTDATA(YES) or CURRENTDATA(NO). For other rows that satisfy the search condition, lock avoidance is possible only when you use the option CURRENTDATA(NO). For more details, see "The CURRENTDATA option" on page 853.
- *Lock avoidance on uncommitted data*: For rows that do not satisfy the search condition, lock avoidance is possible when the value of EVALUATE UNCOMMITTED is YES. For details, see "Option to avoid locks during predicate evaluation" on page 842.

ISOLATION(UR)

Uncommitted read: The application acquires no page or row locks and can run concurrently with most other operations.[7] But the application is in danger of reading data that was changed by another operation but not yet committed. A UR application can acquire LOB locks, as described in "LOB locks" on page 860.

For restrictions on isolation UR, see "Restrictions" on page 851 for more information.

ISOLATION(RS)

Read stability: A row or page lock is held for rows or pages that are returned to an application at least until the next commit point. If a row or page is rejected during stage 2 processing, its lock is still held, even though it is not returned to the application. For multiple-row fetch, the lock is released if the stage 2 predicate fails.

If the application process returns to the same page and reads the same row again, another application cannot have changed the rows, although additional qualifying rows might have been inserted by another application process. A similar situation can also occur if a row or page that is not returned to the application is updated by another application process. If the row now satisfies the search condition, it appears.

7. The exceptions are mass delete operations and utility jobs that drain all claim classes.

When determining whether a row satisfies the search condition, DB2 can avoid taking the lock altogether if the row contains uncommitted data. If the row does not satisfy the predicate, lock avoidance is possible when the value of the EVALUATE UNCOMMITTED field of installation panel DSNTIP4 is YES. For details, see "Option to avoid locks during predicate evaluation" on page 842.

ISOLATION(RR)

Repeatable read: A row or page lock is held for all accessed rows, qualifying or not, at least until the next commit point. If the application process returns to the same page and reads the same row again, another application cannot have changed the rows nor have inserted any new qualifying rows.

The repeatability of the read is guaranteed only until the application commits. Even if a cursor is held on a specific row or page, the result set can change after a commit.

Default: The default differs for different types of bind operations, as shown in.

Table 146. The default ISOLATION values for different bind operations

Operation	Default value
BIND PLAN	ISOLATION(RR)
BIND PACKAGE	The value used by the plan that includes the package in its package list
REBIND PLAN or PACKAGE	The existing value for the plan or package being rebound

For more detailed examples, see *DB2 Application Programming and SQL Guide*.

Recommendation: Choose an ISOLATION value based on the characteristics of the particular application.

Advantages and disadvantages of the isolation values

The various isolation levels offer less or more concurrency at the cost of more or less protection from other application processes. The values you choose should be based primarily on the needs of the application. This section presents the isolation levels in order of recommendation, from the most recommended (CS) to the least recommended (RR). For ISOLATION (CS), CURRENTDATA(NO) is preferred over CURRENTDATA(YES). For more information on the CURRENTDATA option, see "The CURRENTDATA option" on page 853.

Regardless of the isolation level, uncommitted claims on DB2 objects can inhibit the execution of DB2 utilities or commands. For more information about how claims can affect concurrency see "Claims and drains for concurrency control" on page 864.

ISOLATION (CS)

Allows maximum concurrency with data integrity. However, after the process leaves a row or page, another process can change the data. With CURRENTDATA(NO), the process doesn't have to leave a row or page to allow another process to change the data. If the first process returns to read the same row or page, the data is not necessarily the same. Consider these consequences of that possibility:

- For table spaces created with LOCKSIZE ROW, PAGE, or ANY, a change can occur even while executing a single SQL statement, if the statement reads the same row more than once. In the following example:

  ```
  SELECT * FROM T1
    WHERE COL1 = (SELECT MAX(COL1) FROM T1);
  ```

  data read by the inner SELECT can be changed by another transaction before it is read by the outer SELECT. Therefore, the information returned by this query might be from a row that is no longer the one with the maximum value for COL1.
- In another case, if your process reads a row and returns later to update it, that row might no longer exist or might not exist in the state that it did when your application process originally read it. That is, another application might have deleted or updated the row. **If your application is doing non-cursor operations on a row under the cursor, make sure the application can tolerate "not found" conditions**.

  Similarly, assume another application updates a row after you read it. If your process returns later to update it based on the value you originally read, you are, in effect, erasing the update made by the other process. **If you use isolation (CS) with update, your process might need to lock out concurrent updates.** One method is to declare a cursor with the FOR UPDATE clause.

Product-sensitive Programming Interface

For packages and plans that contain updatable static scrollable cursors, ISOLATION(CS) lets DB2 use *optimistic concurrency control*. DB2 can use optimistic concurrency control to shorten the amount of time that locks are held in the following situations:

- Between consecutive fetch operations
- Between fetch operations and subsequent positioned update or delete operations

DB2 cannot use optimistic concurrency control for dynamic scrollable cursors. With dynamic scrollabe cursors, the most recently fetched row or page from the base table remains locked to maintain position for a positioned update or delete.

Figure 82 on page 850 and Figure 83 on page 850 show processing of positioned update and delete operations with static scrollable cursors without optimistic concurrency control and with optimistic concurrency control.

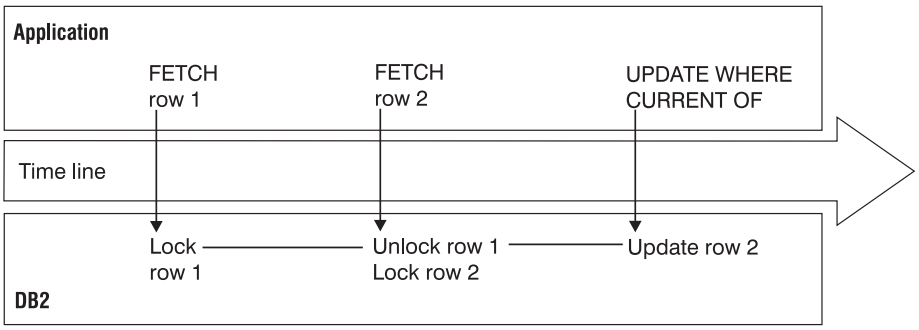


Figure 82. Positioned updates and deletes without optimistic concurrency control

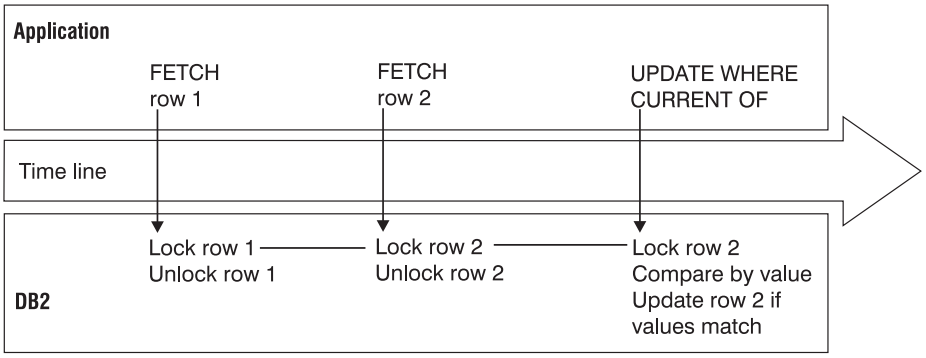


Figure 83. Positioned updates and deletes with optimistic concurrency control

Optimistic concurrency control consists of the following steps:

1. When the application requests a fetch operation to position the cursor on a row, DB2 locks that row, executes the FETCH, and releases the lock.
2. When the application requests a positioned update or delete operation on the row, DB2 performs the following steps:
   a. Locks the row.
   b. Reevaluates the predicate to ensure that the row still qualifies for the result table.
   c. For columns that are in the result table, compares current values in the row to the values of the row when step 1 was executed. Performs the positioned update or delete operation only if the values match.

End of Product-sensitive Programming Interface

ISOLATION (UR)

Allows the application to read while acquiring few locks, at the risk of reading uncommitted data. UR isolation applies only to read-only operations: SELECT, SELECT INTO, or FETCH from a read-only result table.

Reading uncommitted data introduces an element of uncertainty.

Example: An application tracks the movement of work from station to station along an assembly line. As items move from one station to another, the application subtracts from the count of items at the first station and

adds to the count of items at the second. Assume you want to query the count of items at all the stations, while the application is running concurrently.

What can happen if your query reads data that the application has changed but has not committed?

If the application subtracts an amount from one record before adding it to another, *the query could miss the amount entirely.*

If the application adds first and then subtracts, *the query could add the amount twice*.

When an application uses ISO(UR) and runs concurrently with applications
that update variable-length records such that the update creates a
double-overflow record, the ISO(UR) application might miss rows that are
being updated.

If those situations can occur and are unacceptable, do not use UR isolation.

Restrictions: You cannot use UR isolation for the following types of statements:
- INSERT, UPDATE, and DELETE
- Any cursor defined with a FOR UPDATE clause

If you bind with ISOLATION(UR) and the statement does not specify WITH RR or WITH RS, DB2 uses CS isolation for thee types of statements.

When can you use uncommitted read (UR)? You can probably use UR isolation in cases like the following ones:

- **When errors cannot occur**.

  Example: A reference table, like a table of descriptions of parts by part number. It is rarely updated, and reading an uncommitted update is probably no more damaging than reading the table 5 seconds earlier. Go ahead and read it with ISOLATION(UR).

  Example: The employee table of Spiffy Computer, our hypothetical user. For security reasons, updates can be made to the table only by members of a single department. And that department is also the only one that can query the entire table. It is easy to restrict queries to times when no updates are being made and then run with UR isolation.

- **When an error is acceptable**.

  Example: Spiffy wants to do some statistical analysis on employee data. A typical question is, "What is the average salary by sex within education level?" Because reading an occasional uncommitted record cannot affect the averages much, UR isolation can be used.

- **When the data already contains inconsistent information**.

  Example: Spiffy gets sales leads from various sources. The data is often inconsistent or wrong, and end users of the data are accustomed to dealing with that. Inconsistent access to a table of data on sales leads does not add to the problem.

Do not use uncommitted read (UR) in the following cases:
When the computations must balance
When the answer must be accurate
When you are not sure it can do no damage

ISOLATION (RS)

Allows the application to read the same pages or rows more than once without allowing qualifying rows to be updated or deleted by another process. It offers possibly greater concurrency than repeatable read,

because although other applications cannot change rows that are returned to the original application, they can insert new rows or update rows that did not satisfy the original application's search condition. Only those rows or pages that satisfy the stage 1 predicate (and all rows or pages evaluated during stage 2 processing) are locked until the application commits. Figure 84 illustrates this. In the example, the rows held by locks L2 and L4 satisfy the predicate.

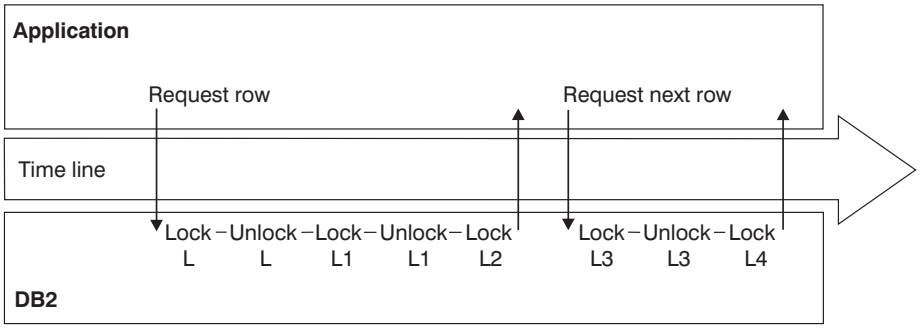


Figure 84. How an application using RS isolation acquires locks when no lock avoidance techniques are used. Locks L2 and L4 are held until the application commits. The other locks aren't held.

Applications using read stability can leave rows or pages locked for long periods, especially in a distributed environment.

If you do use read stability, plan for frequent commit points.

An installation option determines the mode of lock chosen for a cursor defined with the FOR UPDATE OF clause and bound with read stability. For details, see "The option U LOCK FOR RR/RS" on page 840.

ISOLATION (RR)

Allows the application to read the same pages or rows more than once without allowing any UPDATE, INSERT, or DELETE by another process. All accessed rows or pages are locked, even if they do not satisfy the predicate.

Figure 85 shows that all locks are held until the application commits. In the following example, the rows held by locks L2 and L4 satisfy the predicate.

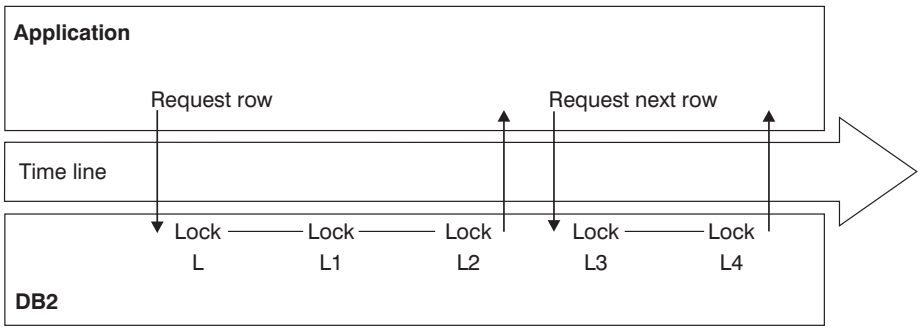


Figure 85. How an application using RR isolation acquires locks. All locks are held until the application commits.

Applications that use repeatable read can leave rows or pages locked for longer periods, especially in a distributed environment, and they can claim more logical partitions than similar applications using cursor stability.

Applications that use repeatable read and access a nonpartitioned index cannot run concurrently with utility operations that drain all claim classes of the nonpartitioned index, even if they are accessing different logical partitions. For example, an application bound with ISOLATION(RR) cannot update partition 1 while the LOAD utility loads data into partition 2. Concurrency is restricted because the utility needs to drain all the repeatable-read applications from the nonpartitioned index to protect the repeatability of the reads by the application.

Because so many locks can be taken, lock escalation might take place. Frequent commits release the locks and can help avoid lock escalation.

With repeatable read, lock promotion occurs for table space scan to prevent the insertion of rows that might qualify for the predicate. (If access is via index, DB2 locks the key range. If access is via table space scans, DB2 locks the table, partition, or table space.)

An installation option determines the mode of lock chosen for a cursor defined with the FOR UPDATE OF clause and bound with repeatable read. For details, see "The option U LOCK FOR RR/RS" on page 840.

Plans and packages that use UR isolation: Auditors and others might need to determine what plans or packages are bound with UR isolation. For queries that select that information from the catalog, see "Ensuring that concurrent users access consistent data" on page 295.

Restrictions on concurrent access: An application using UR isolation cannot run concurrently with a utility that drains all claim classes. Also, the application must acquire the following locks:

- A special *mass delete lock* acquired in S mode on the target table or table space. A "mass delete" is a DELETE statement without a WHERE clause; that operation must acquire the lock in X mode and thus cannot run concurrently.
- An IX lock on any table space used in the work file database. That lock prevents dropping the table space while the application is running.
- If LOB values are read, LOB locks and a lock on the LOB table space. If the LOB lock is not available because it is held by another application in an incompatible lock state, the UR reader skips the LOB and moves on to the next LOB that satisfies the query.

The CURRENTDATA option

The CURRENTDATA option has different effects, depending on if access is local or remote:

- For **local** access, the option tells whether the data upon which your cursor is positioned must remain identical to (or "current with") the data in the local base table. For cursors positioned on data in a work file, the CURRENTDATA option has no effect. This effect only applies to read-only or *ambiguous* cursors in plans or packages bound with CS isolation.

  A cursor is "ambiguous" if DB2 cannot tell whether it is used for update or read-only purposes. If the cursor appears to be used only for read-only, but dynamic SQL could modify data through the cursor, then the cursor is ambiguous. If you use CURRENTDATA to indicate an ambiguous cursor is

read-only when it is actually targeted by dynamic SQL for modification, you'll get an error. See "Problems with ambiguous cursors" on page 856 for more information about ambiguous cursors.

- For a request to a **remote** system, CURRENTDATA has an effect for ambiguous cursors using isolation levels RR, RS, or CS. For ambiguous cursors, it turns block fetching on or off. (Read-only cursors and UR isolation always use block fetch.) Turning on block fetch offers best performance, but it means the cursor is not current with the base table at the remote site.

Local access: Locally, **CURRENTDATA(YES)** means that the data upon which the cursor is positioned cannot change while the cursor is positioned on it. If the cursor is positioned on data in a local base table or index, then the data returned with the cursor is current with the contents of that table or index. If the cursor is positioned on data in a work file, the data returned with the cursor is current only with the contents of the work file; it is not necessarily current with the contents of the underlying table or index.

Figure 86 shows locking with CURRENTDATA(YES).

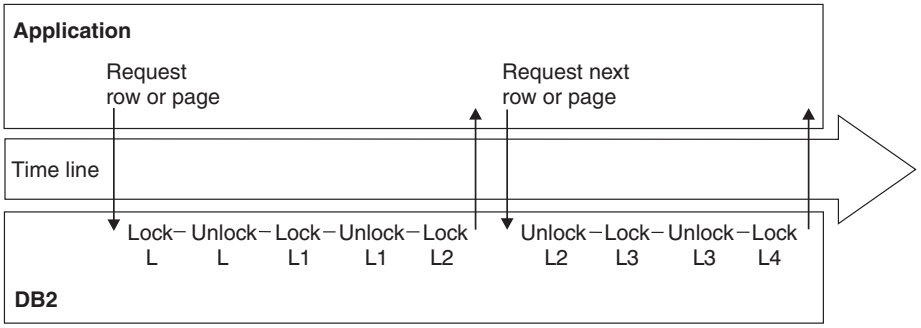


Figure 86. How an application using CS isolation with CURRENTDATA(YES) acquires locks. This figure shows access to the base table. The L2 and L4 locks are released after DB2 moves to the next row or page. When the application commits, the last lock is released.

As with work files, if a cursor uses query parallelism, data is not necessarily current with the contents of the table or index, regardless of whether a work file is used. Therefore, for work file access or for parallelism on read-only queries, the CURRENTDATA option has no effect.

If you are using parallelism but want to maintain currency with the data, you have the following options:

- Disable parallelism (Use SET DEGREE = '1' or bind with DEGREE(1)).
- Use isolation RR or RS (parallelism can still be used).
- Use the LOCK TABLE statement (parallelism can still be used).

For local access, **CURRENTDATA(NO)** is similar to CURRENTDATA(YES) except for the case where a cursor is accessing a base table rather than a result table in a work file. In those cases, although CURRENTDATA(YES) can guarantee that the cursor and the base table are current, CURRENTDATA(NO) makes no such guarantee.

Remote access: For access to a remote table or index, **CURRENTDATA(YES)** turns off block fetching for ambiguous cursors. The data returned with the cursor

is current with the contents of the remote table or index for ambiguous cursors. See "Ensuring block fetch" on page 1007 for information about the effect of CURRENTDATA on block fetch.

Lock avoidance: With CURRENTDATA(NO), you have much greater opportunity for avoiding locks. DB2 can test to see if a row or page has committed data on it. If it has, DB2 does not have to obtain a lock on the data at all. Unlocked data is returned to the application, and the data can be changed while the cursor is positioned on the row. (For SELECT statements in which no cursor is used, such as those that return a single row, a lock is not held on the row unless you specify WITH RS or WITH RR on the statement.)

To take the best advantage of this method of avoiding locks, make sure all applications that are accessing data concurrently issue COMMITs frequently.

Figure 87 shows how DB2 can avoid taking locks and Table 152 summarizes the factors that influence lock avoidance.

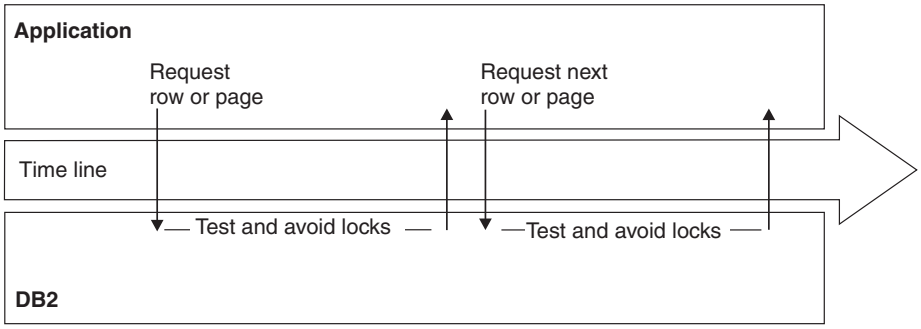


Figure 87. Best case of avoiding locks using CS isolation with CURRENTDATA(NO). This figure shows access to the base table. If DB2 must take a lock, then locks are released when DB2 moves to the next row or page, or when the application commits (the same as CURRENTDATA(YES)).

Table 152. Lock avoidance factors. "Returned data" means data that satisfies the predicate. "Rejected data" is that which does not satisfy the predicate.

Isolation	CURRENTDATA	Cursor type	Avoid locks on returned data?	Avoid locks on rejected data?
UR	N/A	Read-only	N/A	N/A
# CS	YES	Read-only	No	Yes[1]
		Updatable		
		Ambiguous		
	NO	Read-only	Yes	
		Updatable	No	
		Ambiguous	Yes	
# RS	N/A	Read-only	No	Yes[1, 2]
		Updatable		
		Ambiguous		

Table 152. Lock avoidance factors (continued). "Returned data" means data that satisfies the predicate. "Rejected data" is that which does not satisfy the predicate.

Isolation	CURRENTDATA	Cursor type	Avoid locks on returned data?	Avoid locks on rejected data?
RR	N/A	Read-only	No	No
		Updatable		
		Ambiguous		

Note:

1. Locks are avoided when the row is disqualified after stage 1 processing
2. When using ISO(RS) and multi-row fetch, DB2 releases locks that were acquired on Stage 1 qualified rows, but which subsequently failed to qualify for stage 2 predicates at the next fetch of the cursor.

Problems with ambiguous cursors: As shown in Table 152 on page 855, ambiguous cursors can sometimes prevent DB2 from using lock avoidance techniques. However, misuse of an ambiguous cursor can cause your program to receive a -510 SQLCODE:

- The plan or package is bound with CURRENTDATA(NO)
- An OPEN CURSOR statement is performed *before* a dynamic DELETE WHERE CURRENT OF statement against that cursor is prepared
- One of the following conditions is true for the open cursor:
  - Lock avoidance is successfully used on that statement.
  - Query parallelism is used.
  - The cursor is distributed, and block fetching is used.

In all cases, it is a good programming technique to eliminate the ambiguity by declaring the cursor with either the FOR FETCH ONLY or the FOR UPDATE clause.

When plan and package options differ

A plan bound with one set of options can include packages in its package list that were bound with different sets of options. In general, statements in a DBRM bound as a package use the options that the package was bound with, and statements in DBRMs bound to a plan use the options that the plan was bound with.

For example, the plan value for CURRENTDATA has no effect on the packages executing under that plan. If you do not specify a CURRENTDATA option explicitly when you bind a package, the default is CURRENTDATA(YES).

The rules are slightly different for the bind options RELEASE and ISOLATION. The values of those two options are set when the lock on the resource is acquired and usually stay in effect until the lock is released. But a conflict can occur if a statement that is bound with one pair of values requests a lock on a resource that is already locked by a statement that is bound with a different pair of values. DB2 resolves the conflict by resetting each option with the available value that causes the lock to be held for the greatest duration.

If the conflict is between RELEASE(COMMIT) and RELEASE(DEALLOCATE), then the value used is RELEASE(DEALLOCATE).

Table 153 shows how conflicts between isolation levels are resolved. The first column is the existing isolation level, and the remaining columns show what happens when another isolation level is requested by a new application process.

Table 153. Resolving isolation conflicts

	UR	**CS**	**RS**	**RR**
UR	n/a	CS	RS	RR
CS	CS	n/a	RS	RR
RS	RS	RS	n/a	RR
RR	RR	RR	RR	n/a

The effect of WITH HOLD for a cursor

For a cursor defined as WITH HOLD, the cursor position is maintained past a commit point. Hence, locks and claims needed to maintain that position are not released immediately, even if they were acquired with ISOLATION(CS) or RELEASE(COMMIT).

For locks and claims that are needed for cursor position, the following exceptions exist for special cases:

Page and row locks: If you specify NO on the RELEASE LOCKS field on installation panel DSNTIP4, described in "Option to release locks for cursors defined WITH HOLD" on page 841, a page or row lock, if the lock is not successfully avoided through lock avoidance, is held past the commit point. This page or row lock is not necessary for cursor position, but the NO option is provided for compatibility with applications that might rely on this lock. However, an X or U lock is demoted to an S lock at that time. (Because changes have been committed, exclusive control is no longer needed.) After the commit point, the lock is released at the next commit point, provided that no cursor is still positioned on that page or row.

If your installation specifies YES on the RELEASE LOCKS field on installation
panel DSNTIP4, data page or row locks are not held past commit.

Table, table space, and DBD locks: All necessary locks are held past the commit point. After that, they are released according to the RELEASE option under which they were acquired: for COMMIT, at the next commit point after the cursor is closed; for DEALLOCATE, when the application is deallocated.

Claims: All claims, for any claim class, are held past the commit point. They are released at the next commit point after all held cursors have moved off the object or have been closed.

Isolation overriding with SQL statements

The information under this heading, up to "The LOCK TABLE statement" on page 859 is General-use Programming Interface and Associated Guidance Information, as defined in "Notices" on page 1301.

Function of the WITH clause: You can override the isolation level with which a plan or package is bound by the WITH clause on certain SQL statements.

Example: This statement:

```
SELECT MAX(BONUS), MIN(BONUS), AVG(BONUS)
  INTO :MAX, :MIN, :AVG
  FROM DSN8810.EMP
     WITH UR;
```

finds the maximum, minimum, and average bonus in the sample employee table. The statement is executed with uncommitted read isolation, regardless of the value of ISOLATION with which the plan or package containing the statement is bound.

Rules for the WITH clause: The WITH clause:

- Can be used on these statements:
  - Select-statement
  - SELECT INTO
  - Searched delete
  - INSERT from fullselect
  - Searched update
- Cannot be used on subqueries.
- Can specify the isolation levels that specifically apply to its statement. (For example, because WITH UR applies only to read-only operations, you cannot use it on an INSERT statement.)
- Overrides the isolation level for the plan or package only for the statement in which it appears.

USE AND KEEP ... LOCKS options of the WITH clause: If you use the WITH RR or WITH RS clause, you can use the USE AND KEEP EXCLUSIVE LOCKS, USE AND KEEP UPDATE LOCKS, USE AND KEEP SHARE LOCKS options in SELECT and SELECT INTO statements.

Example: To use these options, specify them as shown in the following example:

```
SELECT ...
 WITH RS USE KEEP UPDATE LOCKS;
```

By using one of these options, you tell DB2 to acquire and hold a specific mode of lock on all the qualified pages or rows. Table 154 shows which mode of lock is held on rows or pages when you specify the SELECT using the WITH RS or WITH RR isolation clause.

Table 154. Which mode of lock is held on rows or pages when you specify the SELECT using the WITH RS or WITH RR isolation clause

Option Value	Lock Mode
USE AND KEEP EXCLUSIVE LOCKS	X
USE AND KEEP UPDATE LOCKS	U
USE AND KEEP SHARE LOCKS	S

With read stability (RS) isolation, a row or page that is rejected during stage 2 processing might still have a lock held on it, even though it is not returned to the application.

With repeatable read (RR) isolation, DB2 acquires locks on all pages or rows that fall within the range of the selection expression.

All locks are held until the application commits. Although this option can reduce concurrency, it can prevent some types of deadlocks and can better serialize access to data.

The LOCK TABLE statement

The information under this heading, up to "Claims and drains for concurrency control" on page 864 is General-use Programming Interface and Associated Guidance Information, as defined in "Notices" on page 1301.

For information about using LOCK TABLE on an auxiliary table, see "The LOCK TABLE statement for LOBs" on page 864.

The purpose of LOCK TABLE

Use the LOCK TABLE statement to override DB2's rules for choosing initial lock attributes. Two examples are:

```
LOCK TABLE table-name IN SHARE MODE;
LOCK TABLE table-name PART n IN EXCLUSIVE MODE;
```

Executing the statement requests a lock immediately, unless a suitable lock exists already. The bind option RELEASE determines when locks acquired by LOCK TABLE or LOCK TABLE with the PART option are released.

You can use LOCK TABLE on any table, including auxiliary tables of LOB table spaces. See "The LOCK TABLE statement for LOBs" on page 864 for information about locking auxiliary tables.

LOCK TABLE has no effect on locks acquired at a remote server.

When to use LOCK TABLE

The statement is often appropriate for a particularly high-priority application. The statement can improve performance if LOCKMAX disables lock escalation or sets a high threshold for it.

For example, suppose that you intend to execute an SQL statement to change job code 21A to code 23 in a table of employee data. The table is defined with:

- The name PERSADM1.EMPLOYEE_DATA
- LOCKSIZE ROW
- LOCKMAX 0, which disables lock escalation

Because the change affects about 15% of the employees, the statement can require many row locks of mode X. To avoid the overhead for locks, first execute:

```
LOCK TABLE PERSADM1.EMPLOYEE_DATA IN EXCLUSIVE MODE;
```

If EMPLOYEE_DATA is a partitioned table space, you could choose to lock individual partitions as you update them. An example is:

```
LOCK TABLE PERSADM1.EMPLOYEE_DATA PART 1 IN EXCLUSIVE MODE;
```

When the statement is executed, DB2 locks partition 1 with an X lock. The lock has no effect on locks that already exist on other partitions in the table space.

The effect of LOCK TABLE

Table 155 on page 860 shows the modes of locks acquired in segmented and nonsegmented table spaces for the SHARE and EXCLUSIVE modes of LOCK TABLE. Auxiliary tables of LOB table spaces are considered nonsegmented table spaces and have the same locking behavior.

Table 155. Modes of locks acquired by LOCK TABLE. LOCK TABLE on partitions behave the same as nonsegmented table spaces.

LOCK TABLE IN	Nonsegmented Table Space	Segmented Table Space	
		Table	Table Space
EXCLUSIVE MODE	X	X	IX
SHARE MODE	S or SIX	S or SIX	IS

Note: The SIX lock is acquired if the process already holds an IX lock. SHARE MODE has no effect if the process already has a lock of mode SIX, U, or X.

LOB locks

The locking activity for LOBs is described separately from transaction locks because the purpose of LOB locks is different than that of regular transaction locks.

A lock that is taken on a LOB value in a LOB table space is called a *LOB lock*.

In this section, the following topics are described:

- "Relationship between transaction locks and LOB locks"
- "Hierarchy of LOB locks" on page 861
- "LOB and LOB table space lock modes" on page 861
- "LOB lock and LOB table space lock duration" on page 862
- "Instances when LOB table space locks are not taken" on page 863
- "Control of the number of LOB locks" on page 863
- "The LOCK TABLE statement for LOBs" on page 864
- "LOCKSIZE clause for LOB table spaces" on page 864

Relationship between transaction locks and LOB locks

As described in *DB2 Application Programming and SQL Guide*, LOB column values are stored in a different table space, a LOB table space, from the values in the base table. An application that reads or updates a row in a table that contains LOB columns obtains its normal transaction locks on the base table. The locks on the base table also control concurrency for the LOB table space. When locks are not acquired on the base table, such as for ISO(UR), DB2 maintains data consistency by using locks on the LOB table space. Even when locks are acquired on the base table, DB2 still obtains locks on the LOB table space. When a conditional LOB lock cannot be acquired for some SQL statements with UR isolation, DB2 might return no rows and issue an SQL return code +100.

DB2 also obtains locks on the LOB table space and the LOB values stored in that LOB table space, but those locks have the following primary purposes:

- To determine whether space from a deleted LOB can be reused by an inserted or updated LOB

  Storage for a deleted LOB is not reused until no more readers (including held locators) are on the LOB and the delete operation has been committed.
- To prevent deallocating space for a LOB that is currently being read

  A LOB can be deleted from one application's point-of-view while a reader from another application is reading the LOB. The reader continues reading the LOB because all readers, including those readers that are using uncommitted read isolation, acquire S-locks on LOBs to prevent the storage for the LOB they are reading from being deallocated. That lock is held until commit. A held LOB locator or a held cursor cause the LOB lock and LOB table space lock to be held past commit.

In summary, the main purpose of LOB locks is for managing the space used by LOBs and to ensure that LOB readers do not read partially updated LOBs. Applications need to free held locators so that the space can be reused.

Table 156 shows the relationship between the action that is occurring on the LOB value and the associated LOB table space and LOB locks that are acquired.

Table 156. Locks that are acquired for operations on LOBs. This table does not account for gross locks that can be taken because of LOCKSIZE TABLESPACE, the LOCK TABLE statement, or lock escalation.

Action on LOB value	LOB table space lock	LOB lock	Comment
Read (including UR)	IS	S	Prevents storage from being reused while the LOB is being read or while locators are referencing the LOB
Insert	IX	X	Prevents other processes from seeing a partial LOB
Delete	IS	S	To hold space in case the delete is rolled back. (The X is on the base table row or page.) Storage is not reusable until the delete is committed and no other readers of the LOB exist.
Update	IS->IX	Two LOB locks: an S-lock for the delete and an X-lock for the insert.	Operation is a delete followed by an insert.
Update the LOB to null or zero-length	IS	S	No insert, just a delete.
Update a null or zero-length LOB to a value	IX	X	No delete, just an insert.

ISOLATION(UR) or ISOLATION(CS): When an application is reading rows using uncommitted read or lock avoidance, no page or row locks are taken on the base table. Therefore, these readers must take an S LOB lock to ensure that they are not reading a partial LOB or a LOB value that is inconsistent with the base row.

Hierarchy of LOB locks

Just as page locks (or row locks) and table space locks have a hierarchical relationship, LOB locks and locks on LOB table spaces have a hierarchical relationship. (See Figure 79 on page 818 for a picture of the hierarchical relationship.) If the LOB table space is locked with a gross lock, then LOB locks are not acquired. In a data sharing environment, the lock on the LOB table space is used to determine whether the lock on the LOB must be propagated beyond the local IRLM.

LOB and LOB table space lock modes

This section describes the modes of LOB locks and LOB table space locks.

Modes of LOB locks

The following LOB lock modes are possible:

S (SHARE)
The lock owner and any concurrent processes can read, update, or delete the locked LOB. Concurrent processes can acquire an S lock on the LOB. The purpose of the S lock is to reserve the space used by the LOB.

X (EXCLUSIVE)
The lock owner can read or change the locked LOB. Concurrent processes cannot access the LOB.

Modes of LOB table space locks

The following locks modes are possible on the LOB table space:

IS (INTENT SHARE)
The lock owner can update LOBs to null or zero-length, or read or delete LOBs in the LOB table space. Concurrent processes can both read and change LOBs in the same table space. The lock owner acquires a LOB lock on any data that it reads or deletes.

IX (INTENT EXCLUSIVE)
The lock owner and concurrent processes can read and change data in the LOB table space. The lock owner acquires a LOB lock on any data it accesses.

S (SHARE)
The lock owner and any concurrent processes can read and delete LOBs in the LOB table space. The lock owner does not need LOB locks.

SIX (SHARE with INTENT EXCLUSIVE)
The lock owner can read and change data in the LOB table space. If the lock owner is inserting (INSERT or UPDATE), the lock owner obtains a LOB lock. Concurrent processes can read or delete data in the LOB table space (or update to a null or zero-length LOB).

X (EXCLUSIVE)
The lock owner can read or change LOBs in the LOB table space. The lock owner does not need LOB locks. Concurrent processes cannot access the data.

LOB lock and LOB table space lock duration

This section describes the duration of LOB locks and LOB table space locks.

The duration of LOB locks

Locks on LOBs are taken when they are needed and are usually released at commit. However, if that LOB value is assigned to a LOB locator, the S lock remains until the application commits.

If the application uses HOLD LOCATOR, the LOB lock is not freed until the first commit operation after a FREE LOCATOR statement is issued, or until the thread is deallocated.

If a cursor is defined WITH HOLD, LOB locks are held through commit operations.

Because LOB locks are held until commit and because locks are put on each LOB column in both a source table and a target table, it is possible that a statement such as an INSERT with a fullselect that involves LOB columns can accumulate many

more locks than a similar statement that does not involve LOB columns. To prevent system problems caused by too many locks, you can:

- Ensure that you have lock escalation enabled for the LOB table spaces that are involved in the INSERT. In other words, make sure that LOCKMAX is non-zero for those LOB table spaces.
- Alter the LOB table space to change the LOCKSIZE to TABLESPACE before executing the INSERT with fullselect.
- Increase the LOCKMAX value on the table spaces involved and ensure that the user lock limit is sufficient.
- Use LOCK TABLE statements to lock the LOB table spaces. (Locking the auxiliary table that is contained in the LOB table space locks the LOB table space.)

The duration of LOB table space locks

Locks on LOB table spaces are acquired when they are needed; that is, the ACQUIRE option of BIND has no effect on when the table space lock on the LOB table space is taken. When the table space lock is released is determined by a combination of factors:

- The RELEASE option of bind
- Whether the SQL statement is static or dynamic
- Whether there are held cursors or held locators

When the release option is COMMIT, the lock is released at the next commit point, unless there are held corsors or held locators. If the release option is DEALLOCATE, the lock is released when the object is deallocated (the application ends). The BIND option has no effect on dynamic SQL statements, which always use RELEASE(COMMIT), unless you use dynamic statement caching.

Instances when LOB table space locks are not taken

A lock might not be acquired on a LOB table space at all. For example, if a row is deleted from a table and the value of the LOB column is null, the LOB table space associated with that LOB column is not locked. DB2 does not access the LOB table space if the application:

- Selects a LOB that is null or zero length
- Deletes a row where the LOB is null or zero length
- Inserts a null or zero length LOB
- Updates a null or zero-length LOB to null or zero-length

Control of the number of LOB locks

This section describes how you can control the number of LOB locks that are taken.

Controlling the number of LOB locks that are acquired for a user

LOB locks are counted toward the total number of locks allowed per user. Control this number by the value you specify on the LOCKS PER USER field of installation panel DSNTIPJ. The number of LOB locks that are acquired during a unit of work is reported in IFCID 0020.

Controlling LOB lock escalation

As with any table space, use the LOCKMAX clause of the CREATE or ALTER TABLESPACE statement to control the number of LOB locks that are acquired within a particular LOB table space before the lock is escalated. See “LOCKMAX clause of CREATE and ALTER TABLESPACE” on page 839 for more information.

When the number of LOB locks reaches the maximum you specify in the LOCKMAX clause, the LOB locks escalate to a gross lock on the LOB table space, and the LOB locks are released.

Information about LOB locks and lock escalation is reported in IFCID 0020.

The LOCK TABLE statement for LOBs

"The LOCK TABLE statement" on page 859 describes how and why you might use a LOCK TABLE statement on a table. The reasons for using LOCK TABLE on an auxiliary table are somewhat different than that for regular tables.

- You can use LOCK TABLE to control the number of locks acquired on the auxiliary table.
- You can use LOCK TABLE IN SHARE MODE to prevent other applications from inserting LOBs.

  With auxiliary tables, LOCK TABLE IN SHARE MODE does not prevent any changes to the auxiliary table. The statement does prevent LOBs from being inserted into the auxiliary table, but it does not prevent deletes. Updates are generally restricted also, except where the LOB is updated to a null value or a zero-length string.
- You can use LOCK TABLE IN EXCLUSIVE MODE to prevent other applications from accessing LOBs.

  With auxiliary tables, LOCK TABLE IN EXCLUSIVE MODE also prevents access from uncommitted readers.
- Either statement eliminates the need for lower-level LOB locks.

LOCKSIZE clause for LOB table spaces

The LOCKSIZE TABLE, PAGE, and ROW options are not valid for LOB table spaces. The other options act as follows:

LOCKSIZE TABLESPACE
A process acquires no LOB locks.

LOCKSIZE ANY
DB2 chooses the size of the lock. For a LOB table space, this is usually LOCKSIZE LOB.

LOCKSIZE LOB
If LOBs are accessed, a process acquires LOB locks and the necessary LOB table space locks (IS or IX).

Claims and drains for concurrency control

DB2 utilities, commands, and some ALTER, CREATE, and DROP statements can take over access to some objects independently of any transaction locks that are held on the object.

Objects that are subject to takeover

DB2 utilities, commands, and some ALTER, CREATE, and DROP statements can take over access to the following table and index spaces:

- Simple and segmented table spaces
- Partitions of table spaces
- LOB table spaces
- Nonpartitioned index spaces
- Partitions of index spaces

- Logical partitions of nonpartitioned indexes

The effects of those takeovers are described in the following sections:
- "Claims"
- "Drains"
- "Usage of drain locks" on page 866
- "Utility locks on the catalog and directory" on page 866
- "Compatibility of utilities" on page 867
- "Concurrency during REORG" on page 868
- "Utility operations with nonpartitioned indexes" on page 869

Claims

Definition: A *claim* is a notification to DB2 that an object is being accessed.

Example: When an application first accesses an object, within a unit of work, it makes a claim on the object. It releases the claim at the next commit point.

Effects of a claim: Claims have the following effects:
- Unlike a transaction lock, a claim normally does not persist past the commit point. To access the object in the next unit of work, the application must make a new claim. However, there is an exception. If a cursor defined with the clause WITH HOLD is positioned on the claimed object, the claim is not released at a commit point. For more about cursors defined as WITH HOLD, see "The effect of WITH HOLD for a cursor" on page 857.
- A claim indicates to DB2 that there is activity on or interest in a particular page set or partition. Claims prevent drains from occurring until the claim is released.
- Agents get a claim on data before they get a claim on an index.
- For partitioned table spaces, agents get a claim at the table space level before they claim partitions or indexes.

Three classes of claims: Table 157 shows the three classes of claims and the actions that they allow.

Table 157. Three classes of claims and the actions that they allow

Claim class	Actions allowed
Write	Reading, updating, inserting, and deleting
Repeatable read	Reading only, with repeatable read (RR) isolation
Cursor stability read	Reading only, with read stability (RS), cursor stability (CS), or uncommitted read (UR) isolation

Detecting long-running read claims: DB2 issues a warning message and generates a trace record for each time period that a task holds an uncommitted read claim. You can set the length of the period in minutes by using the LRDRTHLD subsystem parameter.

Drains

Definition: A *drain* is the action of taking over access to an object by preventing new claims and waiting for existing claims to be released.

Example: A utility can drain a partition when applications are accessing it.

Effects of a drain: The drain quiesces the applications by allowing each one to reach a commit point, but preventing any of them, or any other applications, from making a new claim. When no more claims exist, the process that drains (the drainer) controls access to the drained object. The applications that were drained can still hold transaction locks on the drained object, but they cannot make new claims until the drainer has finished.

Claim classes drained: A drainer does not always need complete control. It could drain the following combinations of claim classes:
- Only the write claim class
- Only the repeatable read claim class
- All claim classes

Example: The CHECK INDEX utility needs to drain only writers from an index space and its associated table space. RECOVER, however, must drain all claim classes from its table space. The REORG utility can drain either writers (with DRAIN WRITERS) or all claim classes (with DRAIN ALL).

Usage of drain locks

Definition: A *drain lock* prevents conflicting processes from trying to drain the same object at the same time. Processes that drain only writers can run concurrently; but a process that drains all claim classes cannot drain an object concurrently with any other process. To drain an object, a drainer first acquires one or more drain locks on the object, one for each claim class that it needs to drain. When the locks are in place, the drainer can begin after all processes with claims on the object have released their claims.

A drain lock also prevents new claimers from accessing an object while a drainer has control of it.

Types of drain locks: Three types of drain locks on an object correspond to the three claim classes:
- Write
- Repeatable read
- Cursor stability read

In general, after an initial claim has been made on an object by a user, no other user in the system needs a drain lock. When the drain lock is granted, no drains on the object are in process for the claim class needed, and the claimer can proceed.

Exception: The claimer of an object requests a drain lock in two exceptional cases:
- A drain on the object is in process for the claim class needed. In this case, the claimer waits for the drain lock.
- The claim is the first claim on an object before its data set has been physically opened. Here, acquiring the drain lock ensures that no exception states prohibit allocating the data set.

When the claimer gets the drain lock, it makes its claim and releases the lock before beginning its processing.

Utility locks on the catalog and directory

When the target of a utility is an object in the catalog or directory, such as a catalog table, the utility either drains or claims the object.

When the target is a user-defined object, the utility claims or drains it but also uses the directory and, perhaps, the catalog; for example, to check authorization. In those cases, the utility uses transaction locks on catalog and directory tables. It acquires those locks in the same way as an SQL transaction does. For information about the SQL statements that require locks on the catalog, see "Locks on the DB2 catalog" on page 824.

The UTSERIAL lock: Access to the SYSUTILX table space in the directory is controlled by a unique lock called UTSERIAL. A utility must acquire the UTSERIAL lock to read or write in SYSUTILX, whether SYSUTILX is the target of the utility or is used only incidentally.

Compatibility of utilities

Definition: Two utilities are considered *compatible* if they do not need access to the same object at the same time in incompatible modes.

Compatibility rules: The concurrent operation of two utilities is not typically controlled by either drain locks or transaction locks, but merely by a set of compatibility rules.

Before a utility starts, it is checked against all other utilities running on the same target object. The utility starts only if all the others are compatible.

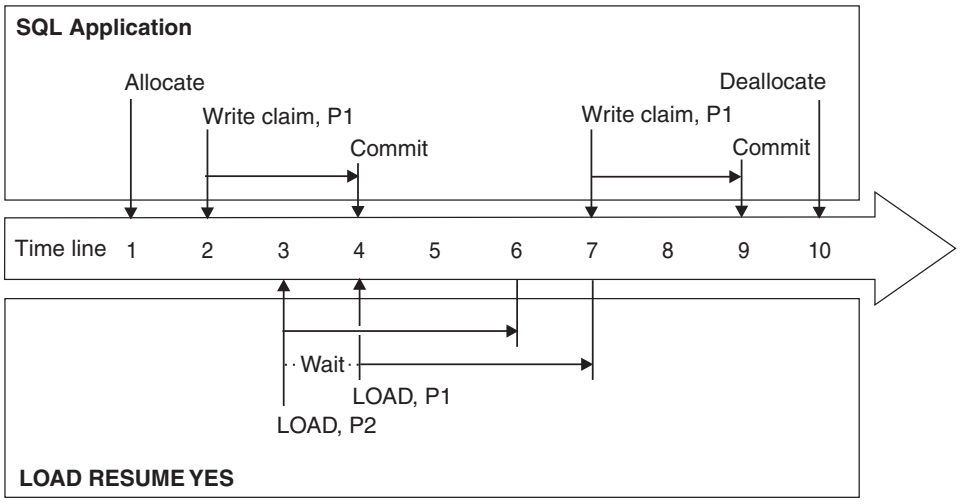
The check for compatibility obeys the following rules:

- The check is made for each target object, but only for target objects. Typical utilities access one or more table spaces or indexes, but if two utility jobs use none of the same target objects, the jobs are always compatible.

  An exception is a case in which one utility must update a catalog or directory table space that is not the direct target of the utility. For example, the LOAD utility on a user table space updates DSNDB06.SYSCOPY. Therefore, other utilities that have DSNDB06.SYSCOPY as a target might not be compatible.
- Individual data and index partitions are treated as distinct target objects. Utilities operating on different partitions in the same table or index space are compatible.
- When two utilities access the same target object, their most restrictive access modes determine whether they are compatible. For example, if utility job 1 reads a table space during one phase and writes during the next, it is considered a writer. It cannot start concurrently with utility 2, which allows only readers on the table space. (Without this restriction, utility 1 might start and run concurrently with utility 2 for one phase; but then it would fail in the second phase, because it could not become a writer concurrently with utility 2.)

For details on which utilities are compatible, refer to each utility's description in *DB2 Utility Guide and Reference*.

Figure 88 on page 868 illustrates how SQL applications and DB2 utilities can operate concurrently on separate partitions of the same table space.



Time	Event
t1	An SQL application obtains a transaction lock on every partition in the table space. The duration of the locks extends until the table space is deallocated.
t2	The SQL application makes a write claim on data partition 1 and index partition 1.
t3	The LOAD jobs begin draining all claim classes on data partitions 1 and 2 and index partitions 1 and 2. LOAD on partition 2 operates concurrently with the SQL application on partition 1. LOAD on partition 1 waits.
t4	The SQL application commits, releasing its write claims on partition 1. LOAD on partition 1 can begin.
t6	LOAD on partition 2 completes.
t7	LOAD on partition 1 completes, releasing its drain locks. The SQL application (if it has not timed out) makes another write claim on data partition 1.
t10	The SQL application deallocates the table space and releases its transaction locks.

Figure 88. SQL and utility concurrency. Two LOAD jobs execute concurrently on two partitions of a table space

Concurrency during REORG

If you are getting deadlocks when you use REORG with the SHRLEVEL CHANGE option, run the REORG utility with the DRAIN ALL option. The default is DRAIN WRITERS, which is done in the log phase. The specification of DRAIN ALL indicates that both writers and readers will be drained when the MAXRO threshold is reached. The DRAIN ALL option should be considered in environments where a lot of update activity occurs during the log phase. With this specification, there is no need for a subsequent drain in the switch phase.

When multiple REORG utility jobs with the SHRLEVEL CHANGE or REFERENCE and the FASTSWITCH options run against separate partitions of the same partitioned table space, some of the utilities might fail with reason code 00E40012. This code, which indicates the unavailability of the database descriptor block (DBD) is caused by multiple utilities arriving at the SWITCH phase simultaneously. The switch phase times out if it cannot acquire the DBD within the timeout period specified by the UTILITY TIMEOUT field on installation panel DSNTIPI. Increase the value of the installation parameter to alleviate the problem.

Utility operations with nonpartitioned indexes

In the example of Figure 88 on page 868, two LOAD jobs execute concurrently on different partitions of the same table space. When the jobs proceed to build a partitioned index, either a partitioning idex or a secondary index, they operate on different partitions of the index and can operate concurrently. Concurrent operations on different partitions are possible because the index entries in an index partition refer only to data in the corresponding data partition for the table.

However, in a nonpartitioned index, either a partitioning index or a secondary index, an entry can refer to any partition in the underlying table space. DB2 can process a set of entries of a nonpartitioned index that all refer to a single partition and achieve the same results as for a partition of a partitioned index. (Such a set of entries is called a *logical partition* of the nonpartitioned index.)

Utility processing can be more efficient with partitioned indexes because, with the correspondence of index partitions to data partitions, they promote partition-level independence. For example, the REORG utility with the PART option can run faster and with more concurrency when the indexes are partitioned. REORG rebuilds the parts for each partitioned index during the BUILD phase, which can increase parallel processing and reduce the lock contention of nonpartitioned indexes. In addition, for REORG PART with the SHRLEVEL CHANGE or REFERENCE option, the BUILD2 phase can be eliminated. Nonpartitioned indexes are corrected during BUILD2. Thus, if all the indexes on the table are partitioned, the utility skips the BUILD2 phase.

Similarly, for the LOAD PART and REBUILD INDEX PART utilities, the parts for each partitioned index can be built in parallel during the BUILD phase, which reduces lock contention and improves concurrency. The LOAD PART utility also processes partitioned indexes with append logic, instead of the insert logic that it uses to process nonpartitioned indexes, which also improves performance.

Monitoring of DB2 locking

If you have problems with suspensions, timeouts, or deadlocks, you will want to monitor DB2's use of locks.

Recommendation: As described in "Continuous performance monitoring" on page 632, always have statistics classes 1, 3, and 4 and accounting classes 1 and 3 running. The statistics reports provide counters for timeouts, deadlocks and suspensions. Statistics class 3 includes IFCID 0172 (deadlocks) and IFCID 0196 (deadlocks). If deadlocks or timeouts are a concern, print these detail records to investigate the situation during exceptional situations.

The accounting reports show the locking activity under the heading of LOCKING (as shown in Figure 90 on page 872). The other key indication of locking problems are the class 3 suspensions LOCK/LATCH(DB2+IRLM) (**C** in Figure 91 on page 874). If locking and latching are increasing the elapsed time of your transactions or batch work, you want to investigate further.

Use EXPLAIN to monitor the locks required by a particular SQL statement, or all the SQL in a particular plan or package, and see "Using EXPLAIN to tell which locks DB2 chooses" on page 870.

Use the statistics trace to monitor the system-wide use of locks, the accounting trace to monitor locks used by a particular application process, and see "Using the statistics and accounting traces to monitor locking" on page 871.

For an example of resolving a particular locking problem, see "Scenario for analyzing concurrency" on page 872.

Using EXPLAIN to tell which locks DB2 chooses

The information under this heading, up to "Using the statistics and accounting traces to monitor locking" on page 871, is Product-sensitive Programming Interface and Associated Guidance Information, as defined in "Notices" on page 1301.

Procedure:

1. Use the EXPLAIN statement, or the EXPLAIN option of the BIND and REBIND subcommands, to determine which modes of table and table space locks DB2 initially assigns for an SQL statement. Follow the instructions under "Obtaining PLAN_TABLE information from EXPLAIN" on page 928. (EXPLAIN does not return information about the locks acquired on LOB table spaces.)
2. EXPLAIN stores its results in a table called PLAN_TABLE. To review the results, query PLAN_TABLE. After running EXPLAIN, each row of PLAN_TABLE describes the processing for a single table, either one named explicitly in the SQL statement that is being explained or an intermediate table that DB2 has to create. The column TSLOCKMODE of PLAN_TABLE shows an initial lock mode for that table. The lock mode applies to the table or the table space, depending on the value of LOCKSIZE and whether the table space is segmented or nonsegmented.
3. In Table 158, find what table or table space lock is used and whether page or row locks are used also, for the particular combination of lock mode and LOCKSIZE you are interested in.

For statements executed remotely: EXPLAIN gathers information only about data access in the DBMS where the statement is run or the bind operation is carried out. To analyze the locks obtained at a remote DB2 location, you must run EXPLAIN at that location. For more information on running EXPLAIN, and a fuller description of PLAN_TABLE, see Chapter 34, "Using EXPLAIN to improve SQL performance," on page 927.

Table 158. Which locks DB2 chooses. N/A = Not applicable; Yes = Page or row locks are acquired; No = No page or row locks are acquired.

	Lock mode from EXPLAIN				
Table space structure	**IS**	**S**	**IX**	**U**	**X**
For nonsegmented table spaces:					
Table space lock acquired is:	IS	S	IX	U	X
Page or row locks acquired?	Yes	No	Yes	No	No
Note: For partitioned table spaces, the lock mode applies only to those partitions that are locked. Lock modes for LOB table spaces are not reported with EXPLAIN.					
For segmented table spaces with:					
LOCKSIZE ANY, ROW, or PAGE					
Table space lock acquired is:	IS	IS	IX	n/a	IX
Table lock acquired is:	IS	S	IX	n/a	X
Page or row locks acquired?	Yes	No	Yes	n/a	No

Table 158. Which locks DB2 chooses (continued). N/A = Not applicable; Yes = Page or row locks are acquired; No = No page or row locks are acquired.

	Lock mode from EXPLAIN				
Table space structure	**IS**	**S**	**IX**	**U**	**X**
LOCKSIZE TABLE					
Table space lock acquired is:	n/a	IS	n/a	IX	IX
Table lock acquired is:	n/a	S	n/a	U	X
Page or row locks acquired?	n/a	No	n/a	No	No
LOCKSIZE TABLESPACE					
Table space lock acquired is:	n/a	S	n/a	U	X
Table lock acquired is:	n/a	n/a	n/a	n/a	n/a
Page or row locks acquired?	n/a	No	n/a	No	No

Using the statistics and accounting traces to monitor locking

The statistics and accounting trace records contain information on locking. OMEGAMON provides one way to view the trace results. The OMEGAMON reports for the Accounting Trace and Statistics Trace each corresponds to a single DB2 trace record. (Details of those reports are subject to change without notification from DB2 and are available in the appropriate OMEGAMON documentation).

Figure 89 shows a portion of the Statistics Trace, which tells how many suspensions (**A**), timeouts (**B**), deadlocks (**C**), and lock escalations (**D**) occur in the trace record. You can also use the statistics trace to monitor each occurrence of lock escalation.

```
LOCKING ACTIVITY             QUANTITY  /SECOND  /THREAD  /COMMIT
---------------------------  --------  -------  -------  -------
SUSPENSIONS (ALL)  A                2     0.02     1.00     0.40
SUSPENSIONS (LOCK ONLY)             2     0.02     1.00     0.40
SUSPENSIONS (IRLM LATCH)            0     0.00     0.00     0.00
SUSPENSIONS (OTHER)                 0     0.00     0.00     0.00

TIMEOUTS  B                         0     0.00     0.00     0.00
DEADLOCKS  C                        1     0.01     0.50     0.20

LOCK REQUESTS                      17     0.18     8.50     3.40
UNLOCK REQUESTS                    12     0.13     6.00     2.40
QUERY REQUESTS                      0     0.00     0.00     0.00
CHANGE REQUESTS                     5     0.05     2.50     1.00
OTHER REQUESTS                      0     0.00     0.00     0.00

LOCK ESCALATION (SHARED)  D         0     0.00     0.00     0.00
LOCK ESCALATION (EXCLUSIVE)         0     0.00     0.00     0.00

DRAIN REQUESTS                      0     0.00     0.00     0.00
DRAIN REQUESTS FAILED               0     0.00     0.00     0.00
CLAIM REQUESTS                      7     0.08     3.50     1.40
CLAIM REQUESTS FAILED               0     0.00     0.00     0.00
```

Figure 89. Locking activity blocks from statistics trace

Figure 90 on page 872 shows a portion of the Accounting Trace, whichgives the same information for a particular application (suspensions **A** , timeouts **B** , deadlocks **C** , lock escalations **D**). It also shows the maximum number of concurrent page locks held and acquired during the trace (E and F). Review applications with a large number to see if this value can be lowered. This number

is the basis for the proper setting of LOCKS PER USER and, indirectly, LOCKS PER TABLE(SPACE).

```
LOCKING                 TOTAL
-------------------  --------
TIMEOUTS  B                 0
DEADLOCKS  C                0
ESCAL.(SHAR)  D             0
ESCAL.(EXCL)                0
MAX PG/ROW LCK HELD  E      2
LOCK REQUEST  F             8
UNLOCK REQST                2
QUERY REQST                 0
CHANGE REQST                5
OTHER REQST                 0
LOCK SUSPENS.               1
IRLM LATCH SUSPENS.         0
OTHER SUSPENS.              0
TOTAL SUSPENS.  A           1

DRAIN/CLAIM      TOTAL
------------  --------
DRAIN REQST          0
DRAIN FAILED         0
CLAIM REQST          4
CLAIM FAILED         0
```

Figure 90. Locking activity blocks from accounting trace

To determine the effect of lock suspensions on your applications, examine the class 3 LOCK/LATCH time in the Accounting Report (C in Figure 91 on page 874).

Scenario for analyzing concurrency

The concurrency problem analyzed in this section illustrates the approach described in "A general approach to problem analysis in DB2" on page 646. It follows the CONCURRENCY PROBLEM branch of Figure 65 on page 649 and makes use of OMEGAMON reports. In OMEGAMON, a report titled "Trace" corresponds to a single DB2 trace record; a report titled "Report" summarizes information from several trace records. This scenario makes use of:

- Accounting report - long
- Locking suspension report
- Lockout report
- Lockout trace

"Scenario description" includes a description of the scenario, an explanation of how each report helps to analyze the situation, and some general information about corrective approaches.

Scenario description

An application, which has recently been moved into production, is experiencing timeouts. Other applications have not been significantly affected in this example.

To investigate the problem, determine a period when the transaction is likely to time out. When that period begins:

1. Start the GTF.
2. Start the DB2 accounting classes 1, 2, and 3 to GTF to allow for the production of OMEGAMON accounting reports.
3. Stop GTF and the traces after about 15 minutes.

4. Produce and analyze the OMEGAMON accounting report - long.
5. Use the DB2 performance trace selectively for detailed problem analysis.

In some cases, the initial and detailed stages of tracing and analysis presented in this chapter can be consolidated into one. In other cases, the detailed analysis might not be required at all.

To analyze the problem, generally start with the accounting report - long. (If you have enough information from program and system messages, you can skip this first step.)

Accounting report

Figure 91 on page 874 shows a portion of the accounting report - long.

```
AVERAGE       APPL(CL.1)  DB2 (CL.2)  IFI (CL.5)    CLASS 3 SUSPENSIONS   AVERAGE TIME  AV.EVENT    HIGHLIGHTS  D
               A           B                                             C
------------  ----------  ----------  ----------    --------------------  ------------  --------    --------------------------
ELAPSED TIME    0.072929    0.029443         N/P    LOCK/LATCH(DB2+IRLM)      0.000011      0.04    #OCCURRENCES    :      193
 NONNESTED      0.072929    0.029443         N/A    SYNCHRON. I/O             0.010170      9.16    #ALLIEDS        :        0
 STORED PROC    0.000000    0.000000         N/A     DATABASE I/O             0.006325      8.16    #ALLIEDS DISTRIB:        0
 UDF            0.000000    0.000000         N/A     LOG WRITE I/O            0.003845      1.00    #DBATS          :      193
 TRIGGER        0.000000    0.000000         N/A    OTHER READ I/O            0.000000      0.00    #DBATS DISTRIB. :        0
                                                     OTHER WRTE I/O            0.000148      0.04    #NO PROGRAM DATA:        0
CPU TIME        0.026978    0.018994         N/P    SER.TASK SWTCH            0.000115      0.04    #NORMAL TERMINAT:      193
 AGENT          0.026978    0.018994         N/A     UPDATE COMMIT            0.000000      0.00    #ABNORMAL TERMIN:        0
  NONNESTED     0.026978    0.018994         N/P     OPEN/CLOSE               0.000000      0.00    #CP/X PARALLEL. :        0
  STORED PRC    0.000000    0.000000         N/A     SYSLGRNG REC             0.000115      0.04    #IO PARALLELISM :        0
  UDF           0.000000    0.000000         N/A     EXT/DEL/DEF              0.000000      0.00    #INCREMENT. BIND:        0
  TRIGGER       0.000000    0.000000         N/A     OTHER SERVICE            0.000000      0.00    #COMMITS        :      193
 PAR.TASKS      0.000000    0.000000         N/A    ARC.LOG(QUIES)            0.000000      0.00    #ROLLBACKS      :        0
                                                     ARC.LOG READ              0.000000      0.00    #SVPT REQUESTS  :        0
SUSPEND TIME    0.000000    0.010444         N/A    DRAIN LOCK                0.000000      0.00    #SVPT RELEASE   :        0
 AGENT               N/A    0.010444         N/A    CLAIM RELEASE             0.000000      0.00    #SVPT ROLLBACK  :        0
 PAR.TASKS           N/A    0.000000         N/A    PAGE LATCH                0.000000      0.00    MAX SQL CASC LVL:        0
 STORED PROC    0.000000         N/A         N/A    NOTIFY MSGS               0.000000      0.00    UPDATE/COMMIT   :    40.00
 UDF            0.000000         N/A         N/A    GLOBAL CONTENTION         0.000000      0.00    SYNCH I/O AVG.  : 0.001110
                                                     COMMIT PH1 WRITE I/O      0.000000      0.00
NOT ACCOUNT.         N/A    0.000004         N/A    ASYNCH CF REQUESTS        0.000000      0.00
DB2 ENT/EXIT         N/A      182.00         N/A    TOTAL CLASS 3             0.010444      9.28
EN/EX-STPROC         N/A        0.00         N/A
EN/EX-UDF            N/A        0.00         N/A
DCAPT.DESCR.         N/A         N/A         N/P
LOG EXTRACT.         N/A         N/A         N/P

GLOBAL       CONTENTION       L-LOCKS  AVERAGE TIME  AV.EVENT    GLOBAL       CONTENTION       P-LOCKS  AVERAGE TIME  AV.EVENT
-------------------------------------  ------------  --------    -------------------------------------  ------------  --------
L-LOCKS                                     0.000000      0.00    P-LOCKS                                     0.000000      0.00
 PARENT (DB,TS,TAB,PART)                    0.000000      0.00     PAGESET/PARTITION                          0.000000      0.00
 CHILD  (PAGE,ROW)                          0.000000      0.00     PAGE                                       0.000000      0.00
 OTHER                                      0.000000      0.00     OTHER                                      0.000000      0.00

SQL DML   AVERAGE    TOTAL    SQL DCL           TOTAL    SQL DDL    CREATE   DROP  ALTER    LOCKING                  AVERAGE    TOTAL
-------- -------- --------    -------------- --------    ---------- ------ ------ ------    ---------------------- -------- --------
SELECT      20.00     3860    LOCK TABLE            0    TABLE           0      0      0    TIMEOUTS                   0.00        0
INSERT       0.00        0    GRANT                 0    CRT TTABLE      0    N/A    N/A    DEADLOCKS                  0.00        0
UPDATE      30.00     5790    REVOKE                0    DCL TTABLE      0    N/A    N/A    ESCAL.(SHARED)             0.00        0
DELETE      10.00     1930    SET CURR.SQLID        0    AUX TABLE       0    N/A    N/A    ESCAL.(EXCLUS)             0.00        0
                              SET HOST VAR.         0    INDEX           0      0      0    MAX PG/ROW LOCKS HELD     43.34       47
DESCRIBE     0.00        0    SET CUR.DEGREE        0    TABLESPACE      0      0      0    LOCK REQUEST              63.82    12318
DESC.TBL     0.00        0    SET RULES             0    DATABASE        0      0      0    UNLOCK REQUEST            14.48     2794
PREPARE      0.00        0    SET CURR.PATH         0    STOGROUP        0      0      0    QUERY REQUEST              0.00        0
OPEN        10.00     1930    SET CURR.PREC.        0    SYNONYM         0      0    N/A    CHANGE REQUEST            33.35     6436
FETCH       10.00     1930    CONNECT TYPE 1        0    VIEW            0      0    N/A    OTHER REQUEST              0.00        0
CLOSE       10.00     1930    CONNECT TYPE 2        0    ALIAS           0      0    N/A    LOCK SUSPENSIONS           0.00        0
                              SET CONNECTION        0    PACKAGE       N/A      0    N/A    IRLM LATCH SUSPENSIONS     0.03        5
                              RELEASE               0    PROCEDURE       0      0      0    OTHER SUSPENSIONS          0.00        0
DML-ALL     90.00    17370    CALL                  0    FUNCTION        0      0      0    TOTAL SUSPENSIONS          0.03        5
                              ASSOC LOCATORS        0    TRIGGER         0      0    N/A
                              ALLOC CURSOR          0    DIST TYPE       0      0    N/A
                              HOLD LOCATOR          0    SEQUENCE        0      0      0
                              FREE LOCATOR          0
                              DCL-ALL               0    TOTAL           0      0      0
                                                         RENAME TBL      0
                                                         COMMENT ON      0
                                                         LABEL ON        0
```

Figure 91. Excerpt from accounting report - long

The accounting report - long shows the average elapsed times and the average number of suspensions per plan execution. In Figure 91:

- The class 1 average elapsed time **A** (AET) is 0.072929 seconds. The class 2 times show that 0.029443 seconds **B** of that are spent in DB2; the rest is spent in the application.
- The class 2 AET is spent mostly in lock or latch suspensions (LOCK/LATCH **C** is 0.000011 seconds).
- The HIGHLIGHTS section **D** of the report (upper right) shows #OCCURRENCES as 193; that is the number of accounting (IFCID 3) records.

Lock suspension report

To prepare for using the lock suspension report, start DB2 performance class 6 to GTF. Because that class traces only suspensions, it does not significantly reduce

performance. Figure 92 shows the OMEGAMON lock suspension report.

```
⋮
                                                          --SUSPEND REASONS-- ---------- R E S U M E   R E A S O N S ------
PRIMAUTH      --- L O C K   R E S O U R C E --- TOTAL    LOCAL  GLOB.  S.NFY ---- NORMAL ----  TIMEOUT/CANCEL  --- DEADLOCK --
 PLANNAME     TYPE      NAME                    SUSPENDS LATCH  IRLMQ  OTHER NMBR         AET NMBR         AET NMBR        AET
------------- --------- ----------------------- -------- -----  -----  ----- ---- ----------- ---- ----------- ---- ----------
FPB
 PARALLEL     PARTITION DB  =PARADABA                  2     2      0      0    0         N/C    2  303.277805    0        N/C
                        OB  =TAB1TS                         0      0      0
                        PART= 1
LOCKING REPORT COMPLETE
```

Figure 92. Portion of the OMEGAMON Lock suspension report

The lock suspension report shows:

- Which plans are suspended, by plan name within primary authorization ID. For statements bound to a package, see the information about the plan that executes the package.
- What IRLM requests and which lock types are causing suspensions.
- Whether suspensions are normally resumed or end in timeouts or deadlocks.
- What the average elapsed time (AET) per suspension is.

The report also shows the reason for the suspensions, as described in Table 159.

Table 159. Reasons for suspensions

Reason	Includes
LOCAL	Contention for a local resource
LATCH	Contention for latches within IRLM (with brief suspension)
GLOB.	Contention for a global resource
IRLMQ	An IRLM queued request
S.NFY	Intersystem message sending
OTHER	Page latch or drain suspensions, suspensions because of incompatible retained locks in data sharing, or a value for service use

The preceding list shows only the first reason for a suspension. When the original reason is resolved, the request could remain suspended for a second reason.

Each suspension results in either a normal resume, a timeout, or a deadlock.

The report shows that the suspension causing the delay involves access to partition 1 of table space PARADABA.TAB1TS by plan PARALLEL. Two LOCAL suspensions time out after an average of 5 minutes, 3.278 seconds (303.278 seconds).

Lockout report

Figure 93 on page 876 shows the OMEGAMON lockout report. This report shows that plan PARALLEL contends with the plan DSNESPRR. It also shows that contention is occurring on partition 1 of table space PARADABA.TAB1TS.

```
PRIMAUTH          --- L O C K   R E S O U R C E ---                    --------------- A G E N T S --------------
 PLANNAME         TYPE     NAME                    TIMEOUTS DEADLOCKS   MEMBER    PLANNAME  CONNECT   CORRID         HOLDER WAITER
------------------ --------- ----------------------- -------- ---------   --------  --------- --------  ------------   ------ ------
FPB
 PARALLEL         PARTITION DB  =PARADABA                 2         0   N/P       DSNESPRR  TSO       EOA                 2      0
                            OB  =TAB1TS
                            PART= 1
                  ** LOCKOUTS FOR PARALLEL   **           2         0
```

Figure 93. Portion of the OMEGAMON lockout report

Lockout trace

Figure 94 shows the OMEGAMON lockout trace.

For each contender, this report shows the database object, lock state (mode), and duration for each contention for a transaction lock.

```
⋮
PRIMAUTH CORRNAME CONNTYPE
ORIGAUTH CORRNMBR INSTANCE     EVENT TIMESTAMP           --- L O C K   R E S O U R C E ---
PLANNAME CONNECT               RELATED TIMESTAMP EVENT    TYPE      NAME                    EVENT SPECIFIC DATA
----------------------------- ----------------- -------- --------- ----------------------- ----------------------------------------
FPB      FPBPARAL TSO          15:25:27.23692350 TIMEOUT  PARTITION DB  =PARADABA           REQUEST =LOCK       UNCONDITIONAL
FPB      'BLANK'  AB09C533F92E N/P                                   OB  =TAB1TS             STATE   =S          ZPARM INTERVAL=  300
PARALLEL BATCH                                                       PART= 1                 DURATION=COMMIT     INTERV.COUNTER=    1
                                                                                             HASH    =X'000020E0'
                                                                                             ------------ HOLDERS/WAITERS -----------
                                                                                             HOLDER
                                                                                             LUW='BLANK'.IPSAQ421.AB09C51F32CB
                                                                                             MEMBER  =N/P        CONNECT =TSO
                                                                                             PLANNAME=DSNESPRR   CORRID=EOA
                                                                                             DURATION=COMMIT     PRIMAUTH=KARELLE
                                                                                             STATE   =X

KARL     KARL     TSO          15:30:32.97267562 TIMEOUT  PARTITION DB  =PARADABA           REQUEST =LOCK       UNCONDITIONAL
KARL     'BLANK'  AB09C65528E6 N/P                                   OB  =TAB1TS             STATE   =IS         ZPARM INTERVAL=  300
PARALLEL TSO                                                         PART= 1                 DURATION=COMMIT     INTERV.COUNTER=    1
                                                                                             HASH    =X'000020E0'
                                                                                             ------------ HOLDERS/WAITERS -----------
                                                                                             HOLDER
                                                                                             LUW='BLANK'.IPSAQ421.AB09C51F32CB
                                                                                             MEMBER  =N/P        CONNECT =TSO
                                                                                             PLANNAME=DSNESPRR   CORRID=EOA
                                                                                             DURATION=COMMIT     PRIMAUTH=DAVE
                                                                                             STATE   =X
                                                                                             ENDUSER =DAVEUSER
                                                                                             WSNAME  =DAVEWS
                                                                                             TRANS   =DAVES TRANSACTION
LOCKING TRACE COMPLETE
```

Figure 94. Portion of the OMEGAMON lockout trace

At this point in the investigation, the following information is known:

- The applications that contend for resources
- The page sets for which there is contention
- The impact, frequency, and type of the contentions

The application or data design must be reviewed to reduce the contention.

Corrective decisions

The preceding discussion is a general approach when lock suspensions are unacceptably long or timeouts occur. In such cases, the DB2 performance trace for locking and the OMEGAMON reports can be used to isolate the resource causing the suspensions. The lockout report identifies the resources involved. The lockout trace tells what contending process (agent) caused the timeout.

In Figure 92 on page 875, the number of suspensions is low (only 2) and both have ended in a timeout. Rather than use the DB2 performance trace for locking, use the

preferred option, DB2 statistics class 3 and DB2 performance trace class 1. Then produce the OMEGAMON locking timeout report to obtain the information necessary to reduce overheads.

For specific information about OMEGAMON reports and their usage, see *OMEGAMON Report Reference* and *Using IBM Tivoli OMEGAMON XE on z/OS*.

OMEGAMON online locking conflict display

You can monitor locking conflicts by using the OMEGAMON online locking conflict display. The display shows what is waiting for resources and what is holding resources when a thread is suspended.

Deadlock detection scenarios

The following section includes an examination of two different deadlock scenarios and an explanation of the use of the OMEGAMON deadlock detail report to determine the cause of the deadlock.

The OMEGAMON Locking Trace - Deadlock report formats the information contained in trace record IFCID 0172 (statistics class 3). The report outlines all the resources and agents involved in a deadlock and the significant locking parameters, such as lock state and duration, related to their requests.

These examples assume that statistics class 3 and performance class 1 are activated. Performance class 1 is activated to get IFCID 105 records, which contain the translated names for the database ID and the page set OBID.

The scenarios that follow use three of the DB2 sample tables, DEPT, PROJ, and ACT. They are all defined with LOCKSIZE ANY. Type 2 indexes are used to access all three tables. As a result, contention for locks is only on data pages.

Scenario 1: Two-way deadlock with two resources

In this classic deadlock example, two agents contend for resources; the result is a deadlock in which one of the agents is rolled back. Two transactions and two resources are involved.

First, transaction LOC2A acquires a lock on one resource while transaction LOC2B acquires a lock on another. Next, the two transactions each request locks on the resource held by the other.

The transactions execute as follows:

LOC2A

1. Declare and open a cursor for update on DEPT and fetch from page 2.
2. Declare and open a cursor for update on PROJ and fetch from page 8.
3. Update page 2.
4. Update page 8.
5. Close both cursors and commit.

LOC2B

1. Declare and open a cursor for update on PROJ and fetch from page 8.
2. Declare and open a cursor for update on DEPT and fetch from page 2.
3. Update page 8.
4. Update page 2.
5. Close both cursors and commit.

Events take place in the following sequence:

1. LOC2A obtains a U lock on page 2 in table DEPT, to open its cursor for update.
2. LOC2B obtains a U lock on a page 8 in table PROJ, to open its cursor for update.
3. LOC2A attempts to access page 8, to open its cursor but cannot proceed because of the lock held by LOC2B.
4. LOC2B attempts to access page 2, to open its cursor but cannot proceed because of the lock held by LOC2A.

DB2 selects one of the transactions and rolls it back, releasing its locks. That allows the other transaction to proceed to completion and release its locks also.

Figure 95 shows the OMEGAMON Locking Trace - Deadlock report that is produced for this situation.

The report shows that the only transactions involved came from plans LOC2A and LOC2B. Both transactions came in from BATCH.

```
⋮
PRIMAUTH CORRNAME CONNTYPE
ORIGAUTH CORRNMBR INSTANCE     EVENT TIMESTAMP           --- L O C K   R E S O U R C E ---
PLANNAME CONNECT               RELATED TIMESTAMP EVENT   TYPE      NAME                    EVENT SPECIFIC DATA
------------------------------ ----------------- -------- --------- ----------------------- ----------------------------------------
SYSADM   RUNLOC2A TSO          20:32:30.68850025 DEADLOCK                                   COUNTER =    2      WAITERS =    2
SYSADM   'BLANK'  AADD32FD8A8C N/P                                                           TSTAMP  =04/02/95 20:32:30.68
LOC2A    BATCH                                           DATAPAGE  DB  =DSN8D42A           HASH    =X'01060304'
   A                                                                OB  =DEPT              ---------------- BLOCKER IS HOLDER -----
                                                                    PAGE=X'000002'         LUW='BLANK'.EGTVLU2.AADD32FD8A8C
                                                                                           MEMBER  =DB1A       CONNECT =BATCH
                                                                                           PLANNAME=LOC2A      CORRID=RUNLOC2A
                                                                                           DURATION=MANUAL     PRIMAUTH=SYSADM
                                                                                           STATE   =U
                                                                                           ---------------- WAITER ----------------
                                                                                           LUW='BLANK'.EGTVLU2.AA65FEDC1022
                                                                                           MEMBER  =DB1A       CONNECT =BATCH
                                                                                           PLANNAME=LOC2B      CORRID=RUNLOC2B
                                                                                           DURATION=MANUAL     PRIMAUTH=KATHY
                                                                                           REQUEST =LOCK       WORTH   =    18
                                                                                           STATE   =U

                                                           DATAPAGE  DB  =DSN8D42A           HASH    =X'01060312'
                                                                    OB  =PROJ              ---------------- BLOCKER IS HOLDER -----
                                                                    PAGE=X'000008'         LUW='BLANK'.EGTVLU2.AA65FEDC1022
                                                                                           MEMBER  =DB1A       CONNECT =BATCH
                                                                                           PLANNAME=LOC2B      CORRID=RUNLOC2B
                                                                                           DURATION=MANUAL     PRIMAUTH=KATHY
                                                                                           STATE   =U
                                                                                           ---------------- WAITER -------*VICTIM*-
                                                                                           LUW='BLANK'.EGTVLU2.AADD32FD8A8C
                                                                                           MEMBER  =DB1A       CONNECT =BATCH
                                                                                           PLANNAME=LOC2A      CORRID=RUNLOC2A
                                                                                           DURATION=MANUAL     PRIMAUTH=SYSADM
                                                                                           REQUEST =LOCK       WORTH   =   17
                                                                                           STATE   =U
```

Figure 95. Deadlock scenario 1: Two transactions and two resources

The lock held by transaction 1 (LOC2A) is a data page lock on the DEPT table and is held in U state. (The value of MANUAL for duration means that, if the plan was bound with isolation level CS and the page was not updated, then DB2 is free to release the lock before the next commit point.)

Transaction 2 (LOC2B) was requesting a lock on the same resource, also of mode U and hence incompatible.

The specifications of the lock held by transaction 2 (LOC2B) are the same. Transaction 1 was requesting an incompatible lock on the same resource. Hence, the deadlock.

Finally, note that the entry in the trace, identified at **A**, is LOC2A. That is the selected thread (the "victim") whose work is rolled back to let the other proceed.

Scenario 2: Three-way deadlock with three resources

In this scenario, three agents contend for resources and the result is a deadlock in which one of the agents is rolled back. Three transactions and three resources are involved.

First, the three transactions each acquire a lock on a different resource. LOC3A then requests a lock on the resource held by LOC3B, LOC3B requests a lock on the resource held by LOC3C, and LOC3C requests a lock on the resource held by LOC3A.

The transactions execute as follows:

LOC3A

1. Declare and open a cursor for update on DEPT and fetch from page 2.
2. Declare and open a cursor for update on PROJ and fetch from page 8.
3. Update page 2.
4. Update page 8.
5. Close both cursors and commit.

LOC3B

1. Declare and open a cursor for update on PROJ and fetch from page 8.
2. Declare and open a cursor for update on ACT and fetch from page 6.
3. Update page 6.
4. Update page 8.
5. Close both cursors and commit.

LOC3C

1. Declare and open a cursor for update on ACT and fetch from page 6.
2. Declare and open a cursor for update on DEPT and fetch from page 2.
3. Update page 6.
4. Update page 2.
5. Close both cursors and commit.

Events take place in the following sequence:

1. LOC3A obtains a U lock on page 2 in DEPT, to open its cursor for update.
2. LOC3B obtains a U lock on page 8 in PROJ, to open its cursor for update.
3. LOC3C obtains a U lock on page 6 in ACT, to open its cursor for update.
4. LOC3A attempts to access page 8 in PROJ but cannot proceed because of the lock held by LOC3B.
5. LOC3B attempts to access page 6 in ACT cannot proceed because of the lock held by LOC3C.
6. LOC3C attempts to access page 2 in DEPT but cannot proceed, because of the lock held by LOC3A.

DB2 rolls back LOC3C and releases its locks. That allows LOC3B to complete and release the lock on PROJ so that LOC3A can complete. LOC3C can then retry.

Figure 96 on page 880 shows the OMEGAMON Locking Trace - Deadlock report produced for this situation.

```
⋮
PRIMAUTH CORRNAME CONNTYPE
ORIGAUTH CORRNMBR INSTANCE     EVENT TIMESTAMP            --- L O C K   R E S O U R C E ---
PLANNAME CONNECT               RELATED TIMESTAMP EVENT    TYPE      NAME                     EVENT SPECIFIC DATA
------------------------------ ----------------- -------- --------- ------------------------ ----------------------------------------
SYSADM   RUNLOC3C TSO          15:10:39.33061694 DEADLOCK                                      COUNTER =    3      WAITERS =    3
SYSADM   'BLANK'  AADE2CF16F34 N/P                                                             TSTAMP  =04/03/95 15:10:39.31
LOC3C    BATCH                                            DATAPAGE  DB  =DSN8D42A             HASH    =X'01060312'
                                                                    OB  =PROJ                 ---------------- BLOCKER IS HOLDER------
                                                                    PAGE=X'000008'            LUW='BLANK'.EGTVLU2.AAD15D373533
                                                                                              MEMBER  =DB1A       CONNECT =BATCH
                                                                                              PLANNAME=LOC3B      CORRID=RUNLOC3B
                                                                                              DURATION=MANUAL     PRIMAUTH=JULIE
                                                                                              STATE   =U
                                                                                              ---------------- WAITER ----------------
                                                                                              LUW='BLANK'.EGTVLU2.AB33745CE357
                                                                                              MEMBER  =DB1A       CONNECT =BATCH
                                                                                              PLANNAME=LOC3A      CORRID=RUNLOC3A
                                                                                              DURATION=MANUAL     PRIMAUTH=BOB
                                                                                              REQUEST =LOCK       WORTH   =   18
                                                                                              STATE   =U
                                                                                              ---------- BLOCKER IS HOLDER --*VICTIM*-
                                                                                              LUW='BLANK'.EGTVLU2.AAD15D373533
                                                                                              MEMBER  =DB1A       CONNECT =BATCH
                                                                                              PLANNAME=LOC3C      CORRID  =RUNLOC3C
                                                                                              DURATION=MANUAL     PRIMAUTH=SYSADM
                                                                                              STATE   =U
                                                                                              ---------------- WAITER ----------------
                                                                                              LUW='BLANK'.EGTVLU2.AB33745CE357
                                                                                              MEMBER  =DB1A       CONNECT =BATCH
                                                                                              PLANNAME=LOC3B      CORRID  =RUNLOC3B
                                                                                              DURATION=MANUAL     PRIMAUTH=JULIE
                                                                                              REQUEST =LOCK       WORTH   =   18
                                                                                              STATE   =U
                                                                                              ---------- BLOCKER IS HOLDER -----------
                                                                                              LUW='BLANK'.EGTVLU2.AAD15D373533
                                                                                              MEMBER  =DB1A       CONNECT =BATCH
                                                                                              PLANNAME=LOC3A      CORRID  =RUNLOC3A
                                                                                              DURATION=MANUAL     PRIMAUTH=BOB
                                                                                              STATE   =U
                                                                                              ---------------- WAITER -------*VICTIM*-
                                                                                              LUW='BLANK'.EGTVLU2.AB33745CE357
                                                                                              MEMBER  =DB1A       CONNECT =BATCH
                                                                                              PLANNAME=LOC3C      CORRID  =RUNLOC3C
                                                                                              DURATION=MANUAL     PRIMAUTH=SYSADM
                                                                                              REQUEST =LOCK       WORTH   =   18
                                                                                              STATE   =U
```

Figure 96. Deadlock scenario 2: Three transactions and three resources

Chapter 32. Using materialized query tables

The information in this chapter is Product-sensitive Programming Interface Information, as defined under "Notices" on page 1301.

Materialized query tables are tables that contain information that is derived and summarized from other tables. Materialized query tables pre-calculate and store the results of queries with expensive join and aggregation operations. By providing this summary information, materialized query tables can simplify query processing and greatly improve the performance of dynamic SQL queries. Materialized query tables are particularly effective in data warehousing applications.

Automatic query rewrite is the process DB2 uses to access data in a materialized query table. If you enable automatic query rewrite, DB2 determines if it can resolve a dynamic query or part of the query by using a materialized query table. If so, DB2 can rewrite the query to use the materialized query table instead of the underlying base tables. Keep in mind that a materialized query table can yield query results that are not current if the base tables change after the materialized query table is updated.

This chapter presents the following topics:

- "Introduction to materialized query tables and automatic query rewrite"
- "Defining a materialized query table" on page 883
- "Populating and maintaining materialized query tables" on page 886
- "Using multilevel security with materialized query tables" on page 888
- "Exploiting automatic query rewrite" on page 889
- "Materialized query table examples shipped with DB2" on page 898

Introduction to materialized query tables and automatic query rewrite

Because databases have grown substantially over the years, queries must operate over huge amounts of data. For example, in a data warehouse environment, decision-support queries might need to operate over 1 to 10 terabytes of data, performing multiple joins and complex aggregation. As the amount of data has increased, so has the demand for more interactive queries.

Despite the increasing amount of data, these queries still require a response time in the order of minutes or seconds. In some cases, the only solution is to pre-compute the whole or parts of each query. You can store these pre-computed results in a materialized query table. You can then use the materialized query table to answer these complicated queries when the system receives them. Using a process called automatic query rewrite, DB2 recognizes when it can transparently rewrite a submitted query to use the stored results in a materialized query table. By querying the materialized query table instead of computing the results from the underlying base tables, DB2 can process some complicated queries much more efficiently. If the estimated cost of the rewritten query is less than the estimated cost of the original query, DB2 uses the rewritten query.

© Copyright IBM Corp. 1982, 2008

Example of automatic query rewrite using a materialized query table

The following example shows how DB2 can use a materialized query table to improve the performance of a simple query. Although most uses of materialized query tables are much more complex than this example, this example does illustrate some basic concepts.

Suppose that you have a very large table named TRANS that contains one row for each transaction that a certain company processes. You want to tally the total amount of transactions by some time period. Although the table contains many columns, you are most interested in these four columns:

- YEAR, MONTH, DAY, which contain the date of a transaction
- AMOUNT, which contains the amount of the transaction

To total the amount of all transactions between 1995 and 2000, by year, you would use the following query:

```
SELECT YEAR, SUM(AMOUNT)
  FROM TRANS
  WHERE YEAR >= '1995' AND YEAR <= '2000'
  GROUP BY YEAR
  ORDER BY YEAR;
```

This query might be very expensive to run, particularly if the TRANS table is a very large table with millions of rows and many columns.

Now suppose that you define a materialized query table named STRANS by using the following CREATE TABLE statement:

```
CREATE TABLE STRANS AS
  (SELECT YEAR AS SYEAR,
          MONTH AS SMONTH,
          DAY AS SDAY,
          SUM(AMOUNT) AS SSUM
     FROM TRANS
     GROUP BY YEAR, MONTH, DAY)
     DATA INITIALLY DEFERRED REFRESH DEFERRED;
```

After you populate STRANS with a REFRESH TABLE statement, the table contains one row for each day of each month and year in the TRANS table.

Using the automatic query rewrite process, DB2 can rewrite the original query into a new query. The new query uses the materialized query table STRANS instead of the original base table TRANS:

```
SELECT SYEAR, SUM(SSUM)
  FROM STRANS
  WHERE SYEAR >= '1995' AND SYEAR <= '2000'
  GROUP BY SYEAR
  ORDER BY SYEAR
```

If you maintain data currency in the materialized query table STRANS, the rewritten query provides the same results as the original query. The rewritten query offers better response time and requires less CPU time.

Steps for using automatic query rewrite

To take advantage of using automatic query rewrite with materialized query tables, you should follow these steps:

1. Define materialized query tables.

2. Populate materialized query tables. Refresh materialized query tables periodically to maintain data currency with base tables. However, realize that refreshing materialized query tables can be an expensive process.
3. Enable automatic query rewrite, and exploit its functions by submitting read-only dynamic queries.
4. Evaluate the effectiveness of the materialized query tables. Drop under-utilized tables, and create new tables as necessary.

Defining a materialized query table

A materialized query table contains pre-computed data that is a result of a query. The query not only defines the table, but also helps to populate the table. When you define a materialized query table, specify these attributes:

- **Type**. You can define a materialized query table as user-maintained or system-maintained.
- **Refreshability**. You can update user-maintained and system-maintained materialized query table with the REFRESH TABLE statement. You can update user-maintained materialized query tables with the LOAD utility and the UPDATE, INSERT, and DELETE SQL statements.
- **Query optimization enablement**. You can enable or disable the use of a materialized query table in automatic query rewrite.

This section includes the following topics:

- "Creating a new materialized query table"
- "Registering an existing table as a materialized query table" on page 884
- "Altering a materialized query table" on page 886

Creating a new materialized query table

To create a new table as a materialized query table, use the CREATE TABLE statement, specifying a fullselect.

Example: The following CREATE TABLE statement defines a materialized query table named TRANSCNT. TRANSCNT summarizes the number of transactions in table TRANS by account, location, and year:

```
CREATE TABLE TRANSCNT (ACCTID, LOCID, YEAR, CNT) AS
  (SELECT ACCOUNTID, LOCATIONID, YEAR, COUNT(*)
     FROM TRANS
     GROUP BY ACCCOUNTID, LOCATIONID, YEAR )
     DATA INITIALLY DEFERRED
     REFRESH DEFERRED
     MAINTAINED BY SYSTEM
     ENABLE QUERY OPTIMIZATION;
```

The fullselect, together with the DATA INITIALLY DEFERRED clause and the REFRESH DEFERRED clause, defines the table as a materialized query table.

You can explicitly specify the column names of the materialized query table or allow DB2 to derive the column names from the fullselect. The column definitions of a materialized query table are the same as those for a declared global temporary table that is defined with the same fullselect.

You must include the DATA INITIALLY DEFERRED and REFRESH DEFERRED clauses when you define a materialized query table.

- DATA INITIALLY DEFERRED means that DB2 does not populate the materialized query table when you create the table.You must explicitly populate the materialized query table.
  - For system-maintained materialized query tables, populate the tables for the first time by using the REFRESH TABLE statement.
  - For user-maintained materialized query tables, populate the table by using the LOAD utility, INSERT statement, or REFRESH TABLE statement.
- REFRESH DEFERRED means that DB2 does not immediately update the data in the materialized query table when its base tables are updated. You can use the REFRESH TABLE statement at any time to update materialized query tables and maintain data currency with underlying base tables.

The MAINTAINED BY SYSTEM clause, which is the default, specifies that the materialized query table is a system-maintained materialized query table. You cannot update a system-maintained materialized query table by using the LOAD utility or the INSERT, UPDATE, or DELETE statements. You can update a system-maintained materialized query table only by using the REFRESH TABLE statement.

Use the MAINTAINED BY USER clause to specify that the table is a user-maintained materialized query table. You can update a user-maintained materialized query table by using the LOAD utility, the INSERT, UPDATE, and DELETE statements, as well as the REFRESH TABLE statement.

The ENABLE QUERY OPTIMIZATION clause, which is the default, specifies that DB2 can consider the materialized query table in automatic query rewrite. Alternatively, you can specify DISABLE QUERY OPTIMIZATION to indicate that DB2 cannot consider the materialized query table in automatic query rewrite. When you enable query optimization, DB2 is more restrictive of what you can select in the fullselect for a materialized query table.

Recommendation: When creating a user-maintained materialized query table, initially disable query optimization. Otherwise, DB2 might automatically rewrite queries to use the empty materialized query table. After you populate the user-maintained materialized query table, you can alter the table to enable query optimization.

The isolation level of the materialized table is the isolation level at which the CREATE TABLE statement is executed.

After you create a materialized query table, it looks and behaves like other tables in the database system, with a few exceptions. DB2 allows materialized query tables in database operations wherever it allows other tables, with a few restrictions. The restrictions are listed in the description of the CREATE TABLE statement in *DB2 SQL Reference*. As with any other table, you can create indexes on the materialized query table; however, the indexes that you create must not be unique. Instead, DB2 uses the materialized query table's definition to determine if it can treat the index as a unique index for query optimization.

For information about using the CREATE TABLE statement to create a materialized query table, see *DB2 SQL Reference*.

Registering an existing table as a materialized query table

You might already have manually created base tables that act like materialized query tables and have queries that directly access the base tables. These base tables

are often referred to as summary tables. To take advantage of automatic query rewrite for existing summary tables, you must use the ALTER TABLE statement to register them as materialized query tables.

Example: Assume that you have an existing summary table named TRANSCOUNT. TRANSCOUNT has four columns to track the number of transactions by account, location, and year. Assume that TRANSCOUNT was created with this CREATE TABLE statement:

```
CREATE TABLE TRANSCOUNT
   (ACCTID    INTEGER NOT NULL
    LOCID     INTEGER NOT NULL
    YEAR      INTEGER NOT NULL
    CNT       INTEGER NOT NULL);
```

The following SELECT statement then populated TRANSCOUNT with data that was derived from aggregating values in the TRANS table:

```
SELECT ACCTID, LOCID, YEAR, COUNT(*)
FROM TRANS
GROUP BY ACCTID, LOCID, YEAR ;
```

You could use the following ALTER TABLE statement to register TRANSCOUNT as a materialized query table. The statement specifies the ADD MATERIALIZED QUERY clause:

```
ALTER TABLE TRANSCOUNT ADD MATERIALIZED QUERY
  (SELECT ACCTID, LOCID, YEAR, COUNT(*) as cnt
     FROM TRANS
     GROUP BY ACCTID, LOCID, YEAR )
     DATA INITIALLY DEFERRED
     REFRESH DEFERRED
     MAINTAINED BY USER;
```

The fullselect must specify the same number of columns as the table you register as a materialized query table. The columns must have the same definitions and have the same column names in the same ordinal positions.

The DATA INITIALLY DEFERRED clause indicates that the table data is to remain the same when the ALTER statement completes. The MAINTAINED BY USER clause indicates that the table is user-maintained. You can continue to update the data in the table by using the LOAD utility or the INSERT, UPDATE, or DELETE statements. You can also use the REFRESH TABLE statement to update the data in the table. The table becomes immediately eligible for use in automatic query rewrite.

To ensure the accuracy of data that is used in automatic query rewrite, ensure that the summary table is current before registering it as a materialized query table. Alternatively, you can follow these steps:

- Register the summary table as a materialized query table with automatic query rewrite disabled.
- Update the newly registered materialized query table to refresh the data.
- Use the ALTER TABLE statement on the materialized query table to enable automatic query rewrite.

The isolation level of the materialized query table is the isolation level at which the ALTER TABLE statement is executed.

For more information about using the ALTER TABLE statement to register a base table as a materialized query table, see *DB2 SQL Reference*.

Altering a materialized query table

You can use the ALTER TABLE statement to perform the following actions on a materialized query table:

- Change the attributes of a materialized query table.

  You can enable or disable automatic query rewrite for a materialized query table with the ENABLE QUERY OPTIMIZATION or DISABLE QUERY OPTIMIZATION clause. You can change the type of materialized query table between system-maintained and user-maintained by using the MAINTAINED BY SYSTEM or MAINTAINED BY USER clause.

  Altering a materialized query table to enable it for query optimization makes the table immediately eligible for use in automatic query rewrite. You must ensure that the data in the materialized query table is current. Otherwise, automatic query rewrite might return results that are not current.

- Change a materialized query table into a base table.

  Example: Assume that you no longer want TRANSCOUNT to be a materialized query table. The following ALTER TABLE statement, which specifies the DROP MATERIALIZED QUERY clause, changes the materialized query table into a base table.

  ```
  ALTER TABLE TRANSCOUNT DROP MATERIALIZED QUERY;
  ```

  The column definitions and the data in the table do not change. However, DB2 can no longer use the table in automatic query rewrite. You can no longer update the table with the REFRESH TABLE statement. One reason you might want to change a materialized query table into a base table is to perform table operations that are restricted for a materialized query table.

  Example: You might want to rotate the partitions on your partitioned materialized query table. In order to rotate the partitions, you must change your materialized query table into a base table. While the table is a base table, you can rotate the partitions. After you rotate the partitions, you can change the table back to a materialized query table.

In addition to using the ALTER TABLE statement, you can change a materialized query table by dropping the table and recreating the materialized query table with a different definition.

Populating and maintaining materialized query tables

After you define a materialized query table, you need to maintain the accuracy of the data in the table. This maintenance includes populating the table for the first time and periodically refreshing the data in the table. You need to refresh the data because any changes that are made to the base tables are not automatically reflected in the materialized query table.

The only way to change the data in a system-maintained materialized query table is through the REFRESH TABLE statement. The INSERT, UPDATE, and DELETE statements, and the LOAD utility cannot refer to a system-maintained materialized query table as a target table. Therefore, a system-maintained materialized query table is read-only. Any view or cursor that is defined on a system-maintained materialized query table is read-only. However, for a user-maintained materialized query table, you can alter the data with the INSERT, UPDATE, and DELETE statements, and the LOAD utility. This section includes the following topics:

- "Using the REFRESH TABLE statement" on page 887
- "Using INSERT, UPDATE, DELETE, and LOAD for materialized query tables" on page 887

- "Collecting statistics on materialized query tables" on page 888

Using the REFRESH TABLE statement

When you create a materialized query table with the CREATE TABLE statement, the table is not immediately populated. You can populate the table for the first time by using the REFRESH TABLE statement.

Example: The following REFRESH TABLE statement populates a materialized query table named SALESCNT:

```
REFRESH TABLE SALESCNT;
```

You can also use the REFRESH TABLE statement to refresh the data in any materialized query table at any time. The REFRESH TABLE statement performs the following actions:

- Deletes all the rows in the materialized query table
- Executes the fullselect in the materialized query table definition to recalculate the data from the tables that are specified in the fullselect with the isolation level for the materialized query table
- Inserts the calculated result into the materialized query table
- Updates the DB2 catalog with a refresh timestamp and the cardinality of the materialized query table

Although the REFRESH TABLE statement involves both deleting and inserting data, DB2 completes these operations in a single commit scope. Therefore, if a failure occurs during execution of the REFRESH TABLE statement, DB2 rolls back all changes that the statement made.

The REFRESH TABLE statement is an explainable statement. The explain output contains rows for INSERT with the fullselect in the materialized query table definition.

Recommendation: Create materialized query tables in segmented table spaces. Because the REFRESH TABLE statement triggers **mass** delete, the statement performs better if you put the materialized query table in a segmented table space. (See "Using INSERT, UPDATE, DELETE, and LOAD for materialized query tables" for information about how to refresh user-maintained materialized query tables in partitioned tables spaces more efficiently.)

Using INSERT, UPDATE, DELETE, and LOAD for materialized query tables

For a user-maintained materialized query table, you can alter the data by using the INSERT, UPDATE, and DELETE statements, and the LOAD utility. You cannot use the INSERT, UPDATE, or DELETE statements, or the LOAD utility to change system-maintained materialized query tables.

Recommendation: Avoid the REFRESH TABLE statement. Because the REFRESH TABLE statement uses a fullselect to refresh a materialized query table, the statement can result in a long-running query. Therefore, using insert, update, delete, or load operations might be more efficient than using the REFRESH TABLE statement. For example, you might find it faster to generate the data for your materialized query table and execute the LOAD utility to populate the data.

Depending on the size and frequency of changes in base tables, you might use different strategies to refresh your materialized query tables. For example, for infrequent, minor changes to dimension tables, you could immediately propagate the changes to the materialized query tables by using triggers. For larger or more frequent changes, you might consider refreshing your user-maintained materialized query tables incrementally to improve performance.

Example: Assume that you need to add a large amount of data to a fact table. Then, you need to refresh your materialized query table to reflect the new data in the fact table. To do this, perform these steps:

- Collect and stage the new data in a separate table.
- Evaluate the new data and apply it to the materialized table as necessary.
- Merge the new data into the fact table

For an example of such code, see member DSNTEJ3M in DSN810.SDSNSAMP, which is shipped with DB2.

Collecting statistics on materialized query tables

When you run the REFRESH TABLE statement, the only statistic that DB2 updates for the materialized query table is the cardinality statistic. For optimal performance of materialized query tables, you need to provide DB2 with accurate catalog statistics for access path selection. To keep catalog statistics current, you should run the RUNSTATS utility after executing a REFRESH TABLE statement or after changing the materialized query table significantly. Otherwise, DB2 uses default or out-of-date statistics. The estimated performance of queries that are generated by automatic rewrite might inaccurately compare less favorably to the original query.

Using multilevel security with materialized query tables

Tables with multilevel security enabled contain a security label column, which is defined with the AS SECURITY LABEL clause. The values in the security label column indicate which users can access the data in each row. If source tables have multilevel security with row-level granularity enabled, some additional rules apply to working with the materialized query table and the source tables.

This section describes the rules that are specific to tables with multilevel security enabled. For more information about multilevel security, see “Multilevel security” on page 192.

Creating a materialized query table

If one or more source tables in the materialized query table definition contain a security label column, the following rules apply to creating a materialized query table:

- If only one source table contains a security label column, the following conditions apply:
  - You must define the security label column in the materialized query table definition with the AS SECURITY LABEL clause.
  - The materialized query table inherits the security label column from the source table.
  - The MAINATAINED BY USER option is allowed.
- If only one source table contains a security label column and the materialized query table is defined with the DEFINITION ONLY clause, the materialized

query table inherits the values in the security label column from the source table. However, the inherited column is not a security label column.

- If more than one source table contains a security label column, DB2 returns an error code and the materialized query table is not created.

Altering a source table

An ALTER TABLE statement to add a security label column to a table fails if the table is a source table for a materialized query table.

Refreshing a materialized query table

The REFRESH TABLE statement deletes the data in a materialized query table and then repopulates the materialized query table according to its table definition. During this refresh process, DB2 does not check for multilevel security with row-level granularity.

Exploiting automatic query rewrite

Automatic query rewrite compares user queries with the fullselect query that defined a materialized query table. It then determines whether the contents of a materialized query table overlap with the contents of a query. When an overlap exists, the query and the materialized query table are said to **match**. After discovering a match, DB2 rewrites the query to access the matched materialized query table instead of the one or more base tables that the original query specified. If a materialized query table overlaps with only part of a query, automatic query rewrite can use the partial match. Automatic query rewrite compensates for the non-overlapping parts of the query by accessing the tables that are specified in the original query.

Automatic query rewrite tries to search for materialized query tables that result in an access path with the lowest cost after the rewrite. DB2 compares the estimated costs of the rewritten query and of the original query and chooses the query with the lower estimated cost.

Many factors determine how well DB2 can exploit automatic query rewrite. This section describes the following factors:

- "Making materialized query tables eligible"
- "Query requirements and the rewrite process" on page 891
- "Recommendations for materialized query table and base table design" on page 897

Making materialized query tables eligible

The following criteria determine whether DB2 can consider using a materialized query table in automatic query rewrite:

- The definition of the materialized query table
- The isolation level of the materialized query table
- The value of two special registers for the materialized query table

To be eligible for use in automatic query rewrite, a materialized query table must be defined with the ENABLE QUERY OPTIMIZATION clause. Additionally, the isolation level of the table must be equal to or higher than the isolation level of the dynamic query being considered for automatic query rewrite. Finally, special registers CURRENT REFRESH AGE and CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION also affect whether a materialized query table is eligible for use.

The value in special register CURRENT REFRESH AGE represents a *refresh age*. The refresh age of a materialized query table is the time since the REFRESH TABLE statement last refreshed the table. (When you run the REFRESH TABLE statement, you update the timestamp in the REFRESH_TIME column in catalog table SYSVIEWS.) The special register CURRENT REFRESH AGE specifies the maximum refresh age that a materialized query table can have. Specifying the maximum age ensures that automatic query rewrite does not use materialized query tables with old data. In Version 8, the CURRENT REFRESH AGE has only two values: 0 or ANY. A value of 0 means that DB2 will consider no materialized query tables in automatic query rewrite. A value of ANY means that DB2 will consider all materialized query tables in automatic query rewrite.

The refresh age of a user-maintained materialized query table might not truly represent the freshness of the data in the table. In addition to the REFRESH TABLE statement, user-maintained query tables can be updated with the INSERT, UPDATE, and DELETE statements and the LOAD utility. Therefore, you can use the CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register to determine which type of materialized query tables, system-maintained or user-maintained, are considered in automatic query rewrite. The special register has four possible values that indicate which materialized query tables DB2 considers for automatic query rewrite:

SYSTEM DB2 considers only system-maintained materialized query tables.

USER DB2 considers only user-maintained materialized query tables.

ALL DB2 considers both types of materialized query tables.

NONE DB2 considers no materialized query tables.

The CURRENT REFRESH AGE and CURRENT MAINT TYPES fields on installation panel DSNTIP4 determine the initial values of the registers. The default value for the CURRENT REFRESH AGE field is 0, and the default value for CURRENT MAINT TYPES is SYSTEM. If your DB2 subsystem is used exclusively for data warehousing, your applications might tolerate data that is not current. If so, set the parameters on the installation panel to enable automatic query rewrite for both system-maintained and user-maintained materialized query tables by default.

Table 160 summarizes how to use the CURRENT REFRESH AGE and CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special registers together. The table shows which materialized query tables DB2 considers in automatic query rewrite.

Table 160. The relationship between CURRENT REFRESH AGE and CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special registers

Value of CURRENT REFRESH AGE	Value of CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION			
	SYSTEM	**USER**	**ALL**	**None**
ANY	All system-maintained materialized query tables	All user-maintained materialized query tables	All materialized query tables (both system-maintained and user-maintained)	None
0	None	None	None	None

You must populate a system-maintained materialized query table before DB2 considers it in automatic query rewrite.

Query requirements and the rewrite process

DB2 supports automatic query rewrite only for read-only, dynamic queries. DB2 cannot automatically rewrite statically bound queries. You can always refer to a materialized query table explicitly in a statically bound query or in a dynamically prepared query. However, you should consider updating materialized query table data more frequently if you frequently query the table directly. Also, allowing end users to refer directly to materialized query tables reduces an installation's flexibility of dropping and creating materialized query tables without affecting applications.

Query rewrite analysis

In general, DB2 considers automatic query rewrite at the query block level. A read-only, dynamic query can contain multiple query blocks. For example, it might contain a subselect of UNION or UNION ALL, temporarily materialized views, materialized table expressions, and subquery predicates. DB2 processes the query without automatic query rewrite if the query block is or contains any of the following items:

- A fullselect in the UPDATE SET statement.
- A fullselect in the INSERT statement.
- A fullselect in the materialized query table definition in the REFRESH TABLE statement.
- An outer join.
- A query block contains user-defined scalar or table functions with the EXTERNAL ACTION attribute or the NON-DETERMINISTIC attribute, or with the built-in function RAND.

If none of these items exist in the query block, DB2 considers automatic query rewrite. DB2 analyzes the query block in the user query and the fullselect in the materialized query table definition to determine if it can rewrite the query. The materialized query table must contain the data from the source tables (both columns and rows) that DB2 needs to satisfy the query. For DB2 to choose a rewritten query, the rewritten query must provide the same results as the user query. (DB2 assumes data currency in the materialized query table.) Furthermore, the rewritten query must offer better performance than the original user query.

DB2 performs a sophisticated analysis to determine whether it can obtain the results for the query from a materialized query table:

- DB2 compares the set of base tables that were used to populate the materialized query table to the set of base tables that are referenced by the user query. If these sets of tables share base tables in common, the query is a candidate for query rewrite.
- DB2 compares the predicates in the materialized query table fullselect to the predicates in the user query. The following factors influence the comparison:
  - The materialized query table fullselect might contain predicates that are not in the user query. If so, DB2 assumes that these predicates might have resulted in discarded rows when the materialized query table was refreshed. Thus, any rewritten query that makes use of the materialized query table might not give the correct results. The query is not a candidate for query rewrite.

    Exception: DB2 behavior differs if a predicate joins a common base table to an extra table that is unique to the materialized query table fullselect. The predicate does not result in discarded data if you define a referential constraint between the two base tables to make the predicate *lossless*.

However, the materialized query table fullselect must not have any local predicates that reference this extra table.

For an example of a lossless predicate, see Example 2 under "Examples of automatic query rewrite."

  - Referential constraints on the source tables are very important in determining whether automatic query rewrite uses a materialized query table.
  - Predicates are much more likely to match if you code the predicate in the user query so that it is the same or very similar to the predicate in the materialized query table fullselect. Otherwise, the matching process might fail on some complex predicates. For example, the matching process between the simple equal predicates such as COL1 = COL2 and COL2 = COL1 succeeds. Furthermore, the matching process between simple equal predicates such as COL1 * (COL2 + COL3) = COL5 and COL5 = (COL3 + COL2) * COL1 succeeds. However, the matching process between equal predicates such as (COL1 + 3) * 10 = COL2 and COL1 * 10 + 30 = COL2 fails.
  - The items in an IN-list predicate do not need to be in exactly the same order for predicate matching to succeed.
- DB2 compares GROUP BY clauses in the user query to GROUP BY clauses in the materialized query table fullselect. If the user query requests data at the same or higher grouping level as the data in the materialized query table fullselect, the materialized query table remains a candidate for query rewrite. DB2 uses functional dependency information and column equivalence in this analysis.
- DB2 compares the columns that are requested by the user query with the columns in the materialized query table. If DB2 can derive the result columns from one or more columns in the materialized query table, the materialized query table remains a candidate for query rewrite. DB2 uses functional dependency information and column equivalence in this analysis.
- DB2 examines predicates in the user query that are not identical to predicates in the materialized query table fullselect. Then, DB2 determines if it can derive references to columns in the base table from columns in the materialized query table instead. If DB2 can derive the result columns from the materialized query table, the materialized query table remains a candidate for query rewrite.

If all of the preceding analyses succeed, DB2 rewrites the user query. DB2 replaces all or some of the references to base tables with references to the materialized query table. If DB2 finds several materialized query tables that it can use to rewrite the query, it might use multiple tables simultaneously. If DB2 cannot use the tables simultaneously, it uses heuristic rules to choose which one to use. After DB2 writes the new query, DB2 determines the cost and the access path of that query. DB2 uses the rewritten query if the estimated cost of the rewritten query is less than the estimated cost of the original query. The rewritten query might give only approximate results if the data in the materialized query table is not up to date.

Examples of automatic query rewrite

The following examples help you understand how DB2 applies automatic query rewrite to improve query performance. Assume a scenario in which a data warehouse has a star schema. The star schema represents the data of a simplified credit card application, as shown in Figure 97 on page 893.

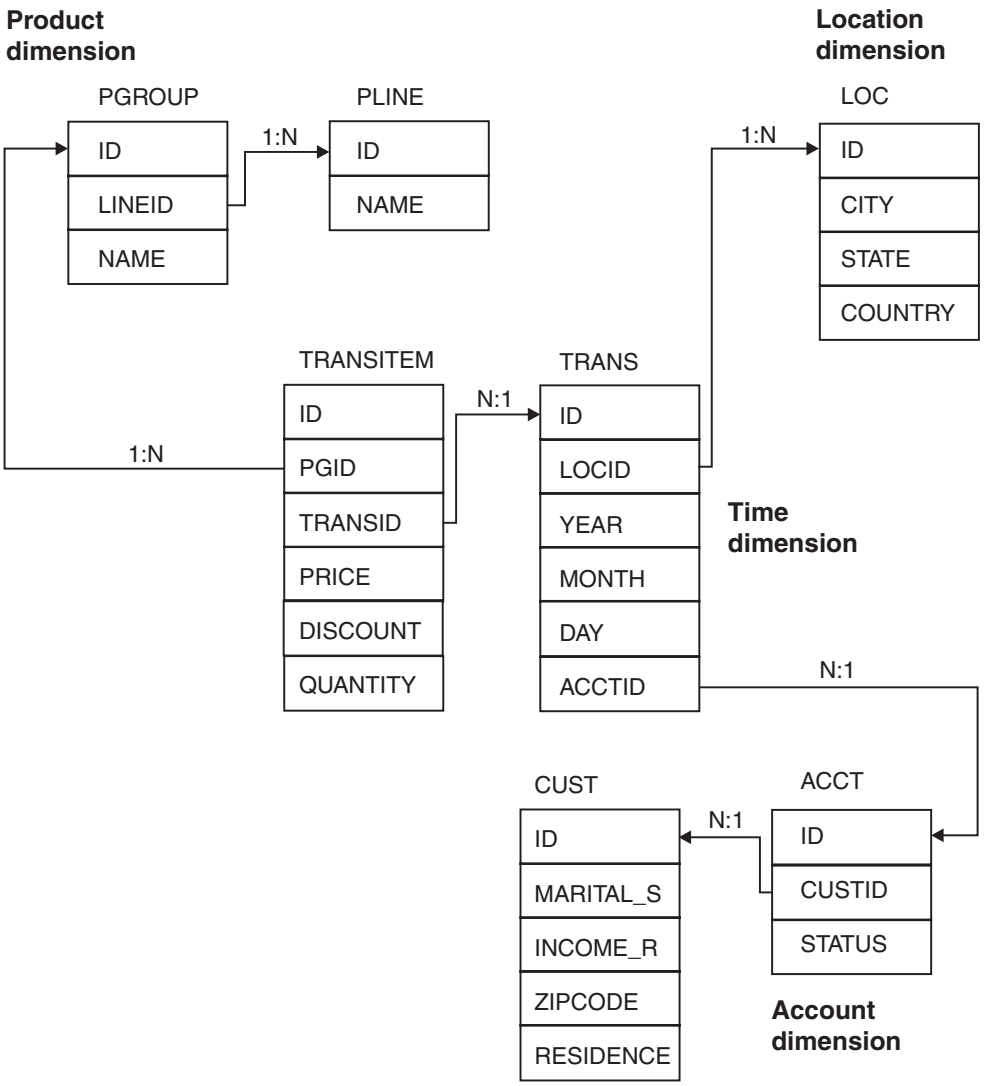


Figure 97. Multi-fact star schema. In this simplified credit card application, the fact tables TRANSITEM and TRANS form the hub of the star schema. The schema also contains four dimensions: product, location, account, and time.

The data warehouse records transactions that are made with credit cards. Each transaction consists of a set of items that are purchased together. At the center of the data warehouse are two large fact tables. TRANS records the set of credit card purchase transactions. TRANSITEM records the information about the items that are purchased. Together, these two fact tables are the hub of the star schema. The star schema is a multi-fact star schema because it contains these two fact tables. The fact tables are continuously updated for each new credit card transaction.

In addition to the two fact tables, the schema contains four dimensions that describe transactions: product, location, account, and time.

- The product dimension consists of two normalized tables, PGROUP and PLINE, that represent the product group and product line.
- The location dimension consists of a single, denormalized table, LOC, that contains city, state, and country.
- The account dimension consists of two normalized tables, ACCT and CUST, that represent the account and the customer.
- The time dimension consists of the TRANS table that contains day, month, and year.

Analysts of such a credit card application are often interested in the aggregation of the sales data. Their queries typically perform joins of one or more dimension tables with fact tables. The fact tables contain significantly more rows than the dimension tables, and complicated queries that involve large fact tables can be very costly. In many cases, you can use materialized query table to summarize and store information from the fact tables. Using materialized query tables can help you avoid costly aggregations and joins against large fact tables.

Example 1: An analyst submits the following query to count the number of transactions that are made in the United States for each credit card. The analyst requests the results grouped by credit card account, state, and year:

```
UserQ1
------
SELECT T.ACCTID, L.STATE, T.YEAR, COUNT(*) AS CNT
  FROM TRANS T, LOC L
  WHERE T.LOCID = L.ID AND
        L.COUNTRY = 'USA'
  GROUP BY T.ACCTID, L.STATE, T.YEAR;
```

Assume that the following CREATE TABLE statement created a materialized query table named TRANSCNT:

```
CREATE TABLE TRANSCNT AS
  (SELECT ACCTID, LOCID, YEAR, COUNT(*) AS CNT
     FROM TRANS
     GROUP BY ACCTID, LOCID, YEAR )
     DATA INITIALLY DEFERRED
     REFRESH DEFERRED;
```

If you enable automatic query rewrite, DB2 can rewrite UserQ1 as NewQ1. NewQ1 accesses the TRANSCNT materialized query table instead of the TRANS fact table.

```
NewQ1
-----
SELECT A.ACCTID, L.STATE, A.YEAR, SUM(A.CNT) AS CNT
  FROM TRANSCNT A, LOC L
  WHERE A.LOCID = L.ID     AND
        L.COUNTRY = 'USA'
  GROUP BY A.ACCTID, L.STATE, A.YEAR;
```

DB2 can use query rewrite in this case because of the following reasons:

- The TRANS table is common to both UserQ1 and TRANSCNT.
- DB2 can derive the columns of the query result from TRANSCNT.
- The GROUP BY in the query requests data that are grouped at a higher level than the level in the definition of TRANSCNT.

Because customers typically make several hundred transactions per year with most of them in the same city, TRANSCNT is about hundred times smaller than TRANS. Therefore, rewriting UserQ1 into a query that uses TRANSCNT instead of TRANS improves response time significantly.

Example 2: Assume that an analyst wants to find the number of televisions, with a price over 100 and a discount greater than 0.1, that were purchased by each credit card account. The analyst submits the following query:

```
UserQ2
------
SELECT T.ID, TI.QUANTITY * TI.PRICE * (1 - TI.DISCOUNT) AS AMT
  FROM TRANSITEM TI, TRANS T, PGROUP PG
  WHERE TI.TRANSID = T.ID  AND
```

```
        TI.PGID = PG.ID    AND
        TI.PRICE > 100     AND
        TI.DISCOUNT > 0.1  AND
        PG.NAME = 'TV';
```

If you define the following materialized query table TRANSIAB, DB2 can rewrite UserQ2 as NewQ2:

```
TRANSIAB
--------
CREATE TABLE TRANSIAB AS
(SELECT TI.TRANSID, TI.PRICE, TI.DISCOUNT, TI.PGID,
        L.COUNTRY, TI.PRICE * TI.QUANTITY as VALUE
   FROM TRANSITEM TI, TRANS T, LOC L
   WHERE TI.TRANSID = T.ID  AND
         T.LOCID = L.ID     AND
         TI.PRICE > 1       AND
         TI.DISCOUNT > 0.1)
   DATA INITIALLY DEFERRED
   REFRESH DEFERRED;


NewQ2
-----
SELECT A.TRANSID, A.VALUE * (1 - A.DISCOUNT) as AM
  FROM TRANSIAB A, PGROUP PG
  WHERE A.PGID = PG.ID     AND
        A.PRICE > 100      AND
        PG.NAME = 'TV';
```

DB2 can rewrite UserQ2 as a new query that uses materialized query table TRANSIAB because of the following reasons:

- Although the predicate T.LOCID = L.ID appears only in the materialized query table, it does not result in rows that DB2 might discard. The referential constraint between the TRANS.LOCID and LOC.ID columns makes the join between TRANS and LOC in the materialized query table definition lossless. The join is lossless only if the foreign key in the constraint is NOT NULL.
- The predicates TI.TRANSID = T.ID and TI.DISCOUNT > 0.1 appear in both the user query and the TRANSIAB fullselect.
- The fullselect predicate TI.PRICE >1 in TRANSIAB subsumes the user query predicate TI.PRICE > 100 in UserQ2. Because the fullselect predicate is more inclusive than the user query predicate, DB2 can compute the user query predicate from TRANSIAB.
- The user query predicate PG.NAME = 'TV' refers to a table that is not in the TRANSIAB fullselect. However, DB2 can compute the predicate from the PGROUP table. A predicate like PG.NAME='TV' does not disqualify other predicates in a query from qualifying for automatic query rewrite. In this case PGROUP is a relatively small dimension table, so a predicate that refers to the table is not overly costly.
- DB2 can derive the query result from the materialized query table definition, even when the derivation is not readily apparent:
  - DB2 derives T.ID in the query from T.TRANSID in the TRANSIAB fullselect. Although these two columns originate from different tables, they are equivalent because of the predicate T.TRANSID = T.ID. DB2 recognizes such column equivalency through join predicates. Thus, DB2 derives T.ID from T.TRANSID, and the query qualifies for automatic query rewrite.
  - DB2 derives AMT in the query UserQ2 from DISCOUNT and VALUE in the TRANSIAB fullselect.

Example 3: This example shows how DB2 matches GROUP BY items and aggregate functions between the user query and the materialized query table fullselect. Assume that an analyst submits the following query to find the average value of the transaction items for each year:

```
UserQ3
------
SELECT YEAR, AVG(QUANTITY * PRICE) AS AVGVAL
  FROM TRANSITEM TI, TRANS T
  WHERE TI.TRANSID = T.ID
  GROUP BY YEAR;
```

If you define the following materialized query table TRANSAVG, DB2 can rewrite UserQ3 as NewQ3:

```
TRANSAVG
--------
CREATE TABLE TRANSAVG AS
  (SELECT T.YEAR, T.MONTH, SUM(QUANTITY * PRICE) AS TOTVAL, COUNT(*) AS CNT
     FROM TRANSITEM TI, TRANS T
     WHERE TI.TRANSID = T.ID
     GROUP BY T.YEAR, T.MONTH )
     DATA INITIALLY DEFERRED
     REFRESH DEFERRED;


NewQ3
-----
SELECT YEAR, CASE WHEN SUM(CNT) = 0 THEN NULL
                  ELSE SUM(TOTVAL)/SUM(CNT)
                  END AS AVGVAL
  FROM TRANSAVG
  GROUP BY YEAR;
```

DB2 can rewrite UserQ3 as a new query that uses materialized query table TRANSAVG because of the following reasons:

- DB2 considers YEAR in the user query and YEAR in the materialized query table fullselect to match exactly.
- DB2 can derive the AVG function in the user query from the SUM function and the COUNT function in the materialized query table fullselect.
- The GROUP BY in the query NewQ3 requests data at a higher level than the level in the definition of TRANSAVG.
- DB2 can compute the yearly average in the user query by using the monthly sums and counts of transaction items in TRANSAVG. DB2 derives the yearly averages from the CNT and TOTVAL columns of the materialized query table by using a case expression.

Determining if query rewrite occurred

You can use EXPLAIN to determine if DB2 has rewritten a user query to use a materialized query table. When DB2 rewrites the query, the PLAN TABLE shows the name of the materialized query that DB2 uses. The value of the TABLE_TYPE column is M to indicate that the table is a materialized query table.

Example: Consider the following user query (from Example 3):

```
SELECT YEAR, AVG(QUANTITY * PRICE) AS AVGVAL
  FROM TRANSITEM TI, TRANS T
  WHERE TI.TRANSID = T.ID
  GROUP BY YEAR;
```

If DB2 rewrites the query to use a materialized query table, a portion of the plan table output might look like Table 161 on page 897.

Table 161. Plan table output for an example with a materialized query table

PLANNO	METHOD	TNAME	JOIN_TYPE	TABLE_TYPE
1	0	TRANSAVG	-	M
2	3	2	-	?

The value M in TABLE_TYPE indicates that DB2 used a materialized query table. TNAME shows that DB2 used the materialized query table named TRANSAVG.You can also obtain this information from a performance trace (IFCID 0022).

Recommendations for materialized query table and base table design

Implementing the following recommendations might improve the performance of your materialized query tables and the queries that use them. These recommendations, however, do not represent a complete design theory.

Materialized query table design

The following recommendations refer directly to designing your materialized query tables:

- Include aggregate functions strategically in the fullselect of a materialized query table definition:
  - Include COUNT(*) and SUM(expression).
  - Include SUM(expression*expression) only if you plan to query VAR(expression), STDDEV(expression), VAR_SAMP(expression), or STDDEV_SAMP(expression).
  - Include COUNT(expression) in addition to COUNT(*) if expression is nullable.
  - Include MIN(expression) and MAX(expression) if you will query them.
  - Do not include AVG(expression), VAR(expression), or STDDEV(expression) directly if you include either of the following parameter combinations:
    - SUM(expression), SUM(expression*expression), and COUNT(*)
    - SUM(expression), SUM(expression*expression), and COUNT(expression)

    DB2 can derive AVG(expression), VAR(expression), and STDDEV(expression) from SUM(expression), SUM(expression*expression), and the appropriate COUNT aggregate function.
- Include the foreign key of a dimension table in the GROUP BY clause of a materialized query table definition. For example, if you include PGROUP.ID, also include PGROUP.LINEID. Then DB2 can use the materialized query table to derive a summary at the LINEID level, without rejoining PGROUP.
- Include all the higher-level columns in the materialized query table if DB2 does not know the functional dependency in a denormalized dimension table. For example, if you include CITY in a GROUP BY clause, also include STATE and COUNTRY in the clause. Similarly, if you include MONTH in the GROUP BY clause, also include YEAR in the clause.
- Do not use the HAVING clause in your materialized query tables. A materialized query table with a HAVING clause in its definition has limited usability during query rewrite.
- Create indexes on materialized query tables as you would for base tables.

Base table design

The following recommendations refer to base table design strategies that might increase the performance and eligibility of your materialized query tables:

- Define referential integrity as either ENFORCED or NOT ENFORCED whenever possible. Also, define an index as unique if it is truly unique.
- Define all base table columns as NOT NULL if possible, so that COUNT(X) is the same as COUNT(*). Then you do not need to include COUNT(X) for each nullable column X in a materialized query table definition. If necessary, use a special value to replace NULL.
- Prefer normalized dimension design to denormalized dimension design in your base tables. When you use normalized dimension design, you do not need to include non-primary key columns in a materialized query table, thereby saving you storage space. DB2 compensates for the lack of non-primary key columns by deriving these columns through a re-join of the dimension table. If performance is a concern due to normalization, you can define materialized query tables to denormalize the snowflake dimensions, if necessary.

Materialized query table examples shipped with DB2

In addition to the examples shown in this chapter, DB2 provides a number of samples to help you design materialized query tables for automatic query rewrite. The samples are based on a data warehouse with a star schema database. The star schema contains one fact table, SALESFACT, and these four hierarchical dimensions:

- A time dimension that consists of one dimension table
- A product dimension that is a snowflake that consists of four fully normalized tables
- A location dimension that is a snowflake that consists of five partially normalized tables
- A customer dimension that is a snowflake that consists of four fully normalized tables

See member DSNTEJ3M in data set DSN810.SDSNSAMP for all of the code, including the following items:

- SQL statements to create and populate the star schema
- SQL statements to create and populate the materialized query tables
- Queries that DB2 rewrites to use the materialized query table

Chapter 33. Maintaining statistics in the catalog

Statistics that are stored in the DB2 catalog help DB2 to determine the access paths for your SQL statements and help to optimize performance. This chapter contains the following sections related to catalog statistics:

- "Statistics used for access path selection"
- "Setting default statistics for created temporary tables" on page 908
- "History statistics" on page 909
- "Gathering monitor statistics and update statistics" on page 912
- "Updating the catalog" on page 913
- "Querying the catalog for statistics" on page 916
- "Improving index and table space access" on page 916
- "Modeling your production system" on page 923

Other considerations for access path selection: In addition to considering catalog statistics, database design, and the SQL statement, the optimizer also considers the central processor model, number of central processors, buffer pool size, and RID pool size. The number of processors is used only when determining degrees of parallelism.

Access path selection uses buffer pool statistics for several calculations. Access path selection also considers the central processor model. These two factors can change your queries' access paths from one system to another, even if all the catalog statistics are identical. You should keep this in mind when migrating from a test system to a production system, or when modeling a new application.

Mixed central processor models in a data sharing group can also affect access path selection. For more information on data sharing, see *DB2 Data Sharing: Planning and Administration*.

Statistics used for access path selection

Table 162 lists the statistics in the DB2 catalog that are used for access path selection, the values that trigger the use of a default value, and the corresponding defaults.

Information in the catalog tables SYSTABLES and SYSTABLESPACE tells how much data is in your table and how many pages hold data. Information in SYSINDEXES lets you compare the available indexes on a table to determine which one is the most efficient for a query. SYSCOLUMNS and SYSCOLDIST provide information to estimate filter factors for predicates.

Table 162. Catalog data used for access path selection or collected by RUNSTATS

Column name	Set by RUNSTATS?	User can update?	Used for access paths? [1]	Description
In every table updated by RUNSTATS:				

© Copyright IBM Corp. 1982, 2008

Table 162. Catalog data used for access path selection or collected by RUNSTATS (continued)

Column name	Set by RUNSTATS?	User can update?	Used for access paths? [1]	Description
STATSTIME	Yes	Yes	No	If updated most recently by RUNSTATS, the date and time of that update, not updatable in SYSINDEXPART and SYSTABLEPART. Used for access path selection for SYSCOLDIST if duplicate column values exist for the same column (by user insertion).
SYSIBM.SYSCOLDIST				
CARDF	Yes	Yes	Yes	The number of distinct values for the column group, -1 if TYPE is F
COLGROUPCOLNO	Yes	Yes	Yes	The set of columns associated with the statistics. Contains an empty string if NUMCOLUMNS = 1.
COLVALUE	Yes	Yes	Yes	Frequently occurring value in the distribution
FREQUENCYF	Yes	Yes	Yes	A number which, multiplied by 100, gives the percentage of rows that contain the value of COLVALUE. For example, 1 means 100% of the rows contain the value and .15 indicates that 15% of the rows contain the value.
NUMCOLUMNS	Yes	Yes	Yes	The number of columns associated with the statistics. The **default value** is 1.
TYPE	Yes	Yes	Yes	The type of statistics gathered, either cardinality (C) or frequent value (F)
SYSIBM.SYSCOLDISTSTATS: contains statistics by partition				
CARDF	Yes	Yes	No	The number of distinct values for the column group, -1 if TYPE is F
COLGROUPCOLNO	Yes	Yes	No	The set of columns associated with the statistics
COLVALUE	Yes	Yes	No	Frequently occurring value in the distribution
FREQUENCYF	Yes	Yes	No	A number which, multiplied by 100, gives the percentage of rows that contain the value of COLVALUE. For example, 1 means 100% of the rows contain the value and .15 indicates that 15% of the rows contain the value.
KEYCARDDATA	Yes	Yes	No	The internal representation of the estimate of the number of distinct values in the partition.
NUMCOLUMNS	Yes	Yes	No	The number of columns associated with the statistics. The **default value** is 1.
TYPE	Yes	Yes	No	The type of statistics gathered, either cardinality (C) or frequent value (F)
SYSIBM.SYSCOLSTATS: contains statistics by partition				

Table 162. Catalog data used for access path selection or collected by RUNSTATS (continued)

Column name	Set by RUNSTATS?	User can update?	Used for access paths? [1]	Description
COLCARD	Yes	Yes	No	The number of distinct values in the partition. Do not update this column manually without first updating COLCARDDATA to a value of length 0.
COLCARDDATA	Yes	Yes	No	The internal representation of the estimate of the number of distinct values in the partition. A value appears here only if RUNSTATS TABLESPACE is run on the partition. Otherwise, this column contains a string of length 0, indicating that the actual value is in COLCARD.
HIGHKEY	Yes	Yes	No	First 2000 bytes of the highest value of the column within the partition. Blank if LOB column.
HIGH2KEY	Yes	Yes	No	First 2000 bytes of the second highest value of the column within the partition. Blank if LOB column.
LOWKEY	Yes	Yes	No	First 2000 bytes of the lowest value of the column within the partition. Blank if LOB column.
LOW2KEY	Yes	Yes	No	First 2000 bytes of the second lowest value of the column within the partition. Blank if LOB column.
SYSIBM.SYSCOLUMNS				
COLCARDF	Yes	Yes	Yes	Estimated number of distinct values in the column, -1 to trigger DB2's use of the **default value** (25) and -2 for the first column of an index of an auxiliary table
HIGH2KEY	Yes	Yes	Yes	First 2000 bytes of the second highest value in this column. Blank for auxiliary index.
LOW2KEY	Yes	Yes	Yes	First 2000 bytes of the second lowest value in this column. Blank for auxiliary index.
SYSIBM.SYSINDEXES				
AVGKEYLEN	Yes	Yes	No	Average key length
CLUSTERED	Yes	Yes	No	Whether the table is actually clustered by the index. Blank for auxiliary index.
CLUSTERING	No	No	Yes	Whether the index was created using CLUSTER

Table 162. Catalog data used for access path selection or collected by RUNSTATS (continued)

Column name	Set by RUNSTATS?	User can update?	Used for access paths? [1]	Description
CLUSTERRATIOF	Yes	Yes	Yes	A number which, when multiplied by 100, gives the percentage of rows in clustering order. For example, 1 indicates that all rows are in clustering order and .87825 indicates that 87.825% of the rows are in clustering order. For a partitioned index, it is the weighted average of all index partitions in terms of the number of rows in the partition. For an auxiliary index it is -2. If this columns contains the default, 0, DB2 uses the value in CLUSTERRATIO, a percentage, for access path selection.
FIRSTKEYCARDF	Yes	Yes	Yes	Number of distinct values of the first key column, or an estimate if updated while collecting statistics on a single partition, -1 to trigger DB2's use of the **default value** (25)
FULLKEYCARDF	Yes	Yes	Yes	Number of distinct values of the full key, -1 to trigger DB2's use of the **default value** (25)
NLEAF	Yes	Yes	Yes	Number of active leaf pages in the index, -1 to trigger DB2's use of the **default value** (SYSTABLES.CARD/300)
NLEVELS	Yes	Yes	Yes	Number of levels in the index tree, -1 to trigger DB2's use of the **default value** (2)
SPACEF	Yes	Yes	No	Disk storage in KB
SYSIBM.SYSINDEXPART: contains statistics for space utilization				
AVGKEYLEN	Yes	Yes	No	Average key length.
CARDF	Yes	No	No	Number of rows or LOBs referenced by the index or partition
DSNUM	Yes	Yes	No	Number of data sets
EXTENTS	Yes	Yes	No	Number of data set extents (when there are multiple pieces, the value is for the extents in the last data set)
FAROFFPOSF	Yes	No	No	Number of referenced rows far from the optimal position because of an insert into a full page
LEAFDIST	Yes	No	No	100 times the number of pages between successive leaf pages.
LEAFFAR	Yes	Yes	No	Number of leaf pages located physically far away from previous leaf pages for successive active leaf pages accessed in an index scan. See "LEAFNEAR and LEAFFAR columns" on page 921 for more information.

Table 162. Catalog data used for access path selection or collected by RUNSTATS (continued)

Column name	Set by RUNSTATS?	User can update?	Used for access paths? [1]	Description
LEAFNEAR	Yes	Yes	No	Number of leaf pages located physically near previous leaf pages for successive active leaf pages. See "LEAFNEAR and LEAFFAR columns" on page 921 for more information.
LIMITKEY	No	No	Yes	The limit key of the partition in an internal format, 0 if the index is not partitioned
NEAROFFPOSF	Yes	No	No	Number of referenced rows near but not at the optimal position because of an insert into a full page
PQTY	Yes	No	No	The primary space allocation in 4K blocks for the data set
PSEUDO_DEL_ENTRIES	Yes	Yes	No	Number of pseudo deleted keys
SECQTYI	Yes	No	No	Secondary space allocation in units of 4 KB, stored in integer format instead of small integer format supported by SQTY. If a storage group is not used, the value is 0.
SPACE	Yes	No	No	The number of KB of space currently allocated for all extents (contains the accumulated space used by all pieces if a page set contains multiple pieces)
SQTY	Yes	No	No	The secondary space allocation in 4 KB blocks for the data set
SPACEF	Yes	Yes	No	Disk storage in KB
SYSIBM.SYSINDEXSTATS: contains statistics by partition				
CLUSTERRATIOF	Yes	Yes	No	A number which, when multiplied by 100, gives the percentage of rows in clustering order. For example, 1 indicates that all rows are in clustering order and .87825 indicates that 87.825% of the rows are in clustering order.
FIRSTKEYCARDF	Yes	Yes	No	Number of distinct values of the first key column, or an estimate if updated while collecting statistics on a single partition
FULLKEYCARDDATA	Yes	Yes	No	The internal representation of the number of distinct values of the full key
FULLKEYCARDF	Yes	Yes	No	Number of distinct values of the full key
KEYCOUNTF	Yes	Yes	No	Number of rows in the partition, -1 to trigger DB2's use of the value in KEYCOUNT
NLEAF	Yes	Yes	No	Number of leaf pages in the index
NLEVELS	Yes	Yes	No	Number of levels in the index tree
SYSIBM.SYSLOBSTATS: contains LOB table space statistics				
AVGSIZE	Yes	Yes	No	Average size of a LOB in bytes
FREESPACE	Yes	Yes	No	The number of KB of available space in the LOB table space

Table 162. Catalog data used for access path selection or collected by RUNSTATS (continued)

Column name	Set by RUNSTATS?	User can update?	Used for access paths? [1]	Description
ORGRATIO	Yes	Yes	No	The percentage of organization in the LOB table space. A value of 100 indicates perfect organization of the LOB table space. A value of 1 indicates that the LOB table space is disorganized. A value of 0.00 indicates that the LOB table space is totally disorganized. An empty table space has an ORGRATIO value of 100.00.
SYSIBM.SYSROUTINES: Contains statistics for table functions. See "Updating catalog statistics" on page 799 for more information about using these statistics.				
CARDINALITY	No	Yes	Yes	The predicted cardinality of a table function, -1 to trigger DB2's use of the **default value** (10 000)
INITIAL_INSTS	No	Yes	Yes	Estimated number of instructions executed the first and last time the function is invoked, -1 to trigger DB2's use of the **default value** (40 000)
INITIAL_IOS	No	Yes	Yes	Estimated number of IOs performed the first and last time the function is invoked, -1 to trigger DB2's use of the **default value** (0)
INSTS_PER_INVOC	No	Yes	Yes	Estimated number of instructions per invocation, -1 to trigger DB2's use of the **default value** (4 000)
IOS_PER_INVOC	No	Yes	Yes	Estimated number of IOs per invocation, -1 to trigger DB2's use of the **default value** (0)
SYSIBM.SYSTABLEPART: contains statistics for space utilization				
AVGROWLEN	Yes	No	No	Average row length
CARDF	Yes	No	No	Total number of rows in the table space or partition. For LOB table spaces, the number of LOBs in the table space.
DSNUM	Yes	Yes	No	Number of data sets
EXTENTS	Yes	Yes	No	Number of data set extents (when there are multiple pieces, the value is for the extents in the last data set)
FARINDREF	Yes	No	No	Number of rows relocated far from their original page
NEARINDREF	Yes	No	No	Number of rows relocated near their original page
PAGESAVE	Yes	No	No	Percentage of pages, times 100, saved in the table space or partition as a result of using data compression
PERCACTIVE	Yes	No	No	Percentage of space occupied by active rows, containing actual data from active tables, -2 for LOB table spaces

Table 162. Catalog data used for access path selection or collected by RUNSTATS (continued)

Column name	Set by RUNSTATS?	User can update?	Used for access paths? [1]	Description
PERCDROP	Yes	No	No	For nonsegmented table spaces, the percentage of space occupied by rows of data from dropped tables; for segmented table spaces, 0
PQTY	Yes	No	No	The primary space allocation in 4K blocks for the data set
SECQTYI	Yes	No	No	Secondary space allocation in units of 4 KB, stored in integer format instead of small integer format supported by SQTY. If a storage group is not used, the value is 0.
SPACE	Yes	No	No	The number of KB of space currently allocated for all extents (contains the accumulated space used by all pieces if a page set contains multiple pieces)
SPACEF	Yes	Yes	No	Disk storage in KB
SQTY	Yes	No	No	The secondary space allocation in 4K blocks for the data set
SYSIBM.SYSTABLES:				
AVGROWLEN	Yes	Yes	No	Average row length of the table specified in the table space
CARDF	Yes	Yes	Yes	Total number of rows in the table or total number of LOBs in an auxiliary table, -1 to trigger DB2's use of the **default value** (10 000)
EDPROC	No	No	Yes	Nonblank value if an edit exit routine is used
NPAGES	Yes	Yes	Yes	Total number of pages on which rows of this table appear, -1 to trigger DB2's use of the **default value** (CEILING(1 + CARD/20))
NPAGESF	Yes	Yes	Yes	Number of pages used by the table
PCTPAGES	Yes	Yes	No	For nonsegmented table spaces, percentage of total pages of the table space that contain rows of the table; for segmented table spaces, the percentage of total pages in the set of segments assigned to the table that contain rows of the table
PCTROWCOMP	Yes	Yes	Yes	Percentage of rows compressed within the total number of active rows in the table
SPACEF	Yes	Yes	No	Disk storage in KB
SYSIBM.SYSTABLESPACE:				
AVGROWLEN	Yes	Yes	No	Average row length

Table 162. Catalog data used for access path selection or collected by RUNSTATS (continued)

Column name	Set by RUNSTATS?	User can update?	Used for access paths? [1]	Description
NACTIVEF	Yes	Yes	Yes	Number of active pages in the table space, the number of pages touched if a cursor is used to scan the entire file, 0 to trigger DB2's use of the value in the NACTIVE column instead. If NACTIVE contains 0, DB2 uses the **default value** (CEILING(1 + CARD/20)).
SPACE	Yes	No	No	Disk storage in KB
SPACEF	Yes	Yes	No	Disk storage in KB
SYSIBM.SYSTABSTATS: contains statistics by partition				
CARDF	Yes	Yes	Yes	Total number of rows in the partition, -1 to trigger DB2's use of the value in the CARD column. If CARD is -1, DB2 uses a **default value**(10 000)
NACTIVE	Yes	Yes	No	Number of active pages in the partition
NPAGES	Yes	Yes	Yes	Total number of pages on which rows of the partition appear, -1 to trigger DB2's use of the **default value** (CEILING(1 + CARD/20))
PCTPAGES	Yes	Yes	No	Percentage of total active pages in the partition that contain rows of the table
PCTROWCOMP	Yes	Yes	No	Percentage of rows compressed within the total number of active rows in the partition, -1 to trigger DB2's use of the **default value** (0)

Note: [1] Statistics on LOB-related values are not used for access path selection. SYSCOLDISTSTATS and SYSINDEXSTATS are not used for parallelism access paths. SYSCOLSTATS information (CARD, HIGHKEY, LOWKEY, HIGH2KEY, and LOW2KEY) is used to determine the degree of parallelism.

Filter factors and catalog statistics

DB2 needs an accurate estimate of the number of rows that will qualify after applying each predicate to determine optimal access paths. When multiple tables are accessed, filtering also affects the cost of join order and join method. The catalog tables SYSIBM.SYSCOLUMNS and SYSIBM.SYSCOLDIST are the main source of statistics for calculating predicate filter factors. The following columns are particularly important:

- SYSCOLUMNS.COLCARDF indicates whether statistics exist for a column or not. A positive value is an estimate of the number of distinct values in the column (cardinality). A value of '-1' results in the use of default statistics.

  The value of COLCARDF generated by RUNSTATS TABLESPACE is an estimate determined by a sampling method. If you know a more accurate number for COLCARDF, you can supply it by updating the catalog. If the column is the first column of an index, the value generated by RUNSTATS INDEX is exact.
- Columns in SYSCOLDIST contain statistics about the frequency (or distribution) of values for a single column. It can also contain statistics about the cardinality of a group of columns and the frequency of values for that group.

When frequency statistics do not exist, DB2 assumes that the data is uniformly distributed and all values in the column occur with the same frequency. This assumption can lead to an inaccurate estimate of the number of qualifying rows if the data is skewed, which can result in performance problems.

Example: Assume that a column (AGE_CATEGORY) contains five distinct values (COLCARDF), each of which occur with the following frequencies:

```
AGE_CATEGORY            FREQUENCY
------------            ---------
INFANT                   5%
CHILD                   15%
ADOLESCENT              25%
ADULT                   40%
SENIOR                  15%
```

Without this frequency information, DB2 would use a default filter factor of 1/5 (1/COLCARDF), or 20%, to estimate the number of rows that qualify for predicate AGE_CATEGORY=ADULT. However, the actual frequency of that age category is 40%. Thus, the number of qualifying rows is underestimated by 50%.

When collecting statistics about indexes, you can specify the KEYCARD option of RUNSTATS to collect cardinality statistics on the specified indexes. You can also specify the FREQVAL option with KEYCARD to specify whether distribution statistics are collected for what number of concatenated index columns. By default, distribution statistics are collected on the first column of each index for the 10 most frequently occurring values. FIRSTKEYCARDF and FULLKEYCARDF are also collected by default.

The value of FULLKEYCARDF generated by RUNSTATS on a DPSI type index is an estimate determined by a sampling method. If you know a more accurate number for FULLKEYCARDF, you can supply it by updating the catalog.

When collecting statistics at the table level, you can specify the COLUMN option of RUNSTATS to collect cardinality statistics on just the specified columns. You can also use the COLGROUP option to specify a group of columns for which to collect cardinality statistics. If you use the FREQVAL option with COLGROUP, you can also collect distribution statistics for the column group.

To limit the resources required to collect statistics, you only need to collect column cardinality and frequency statistics that have changed. For example, a column on GENDER is likely to have a COLCARDF of 2, with M and F as the possible values. It is unlikely that the cardinality for this column will ever change. The distribution of the values in the column may or may not change often, depending on the volatility of the data.

Recommendation: If query performance is not satisfactory, consider the following actions:

- Collect cardinality statistics on all columns that are used as predicates in a WHERE clause.
- Collect frequencies for all columns with a low cardinality that are used as COL op literal predicates.
- Collect frequencies for a column when the column can contain default data, the default data is skewed, and the column is used as a COL op literal predicate.
- Collect KEYCARD on all candidate indexes.
- Collect column group statistics on all join columns.

- LOW2KEY and HIGH2KEY columns are limited to storing the first 2000 bytes of a key value. If the column is nullable, values are limited to 1999 bytes.
- The closer SYSINDEXES.CLUSTERRATIOF is to 100% (a value of 1), the more closely the ordering of the index entries matches the physical ordering of the

table rows. Refer to Figure 98 on page 918 to see how an index with a high cluster ratio differs from an index with a low cluster ratio.

For information about using the RUNSTATS utility, see Part 2 of *DB2 Utility Guide and Reference*.

Statistics for partitioned table spaces

For a partitioned table space, DB2 keeps statistics separately by partition and also collectively for the entire table space. Table 163 shows the catalog tables that contain statistics by partition and, for each one, the table that contains the corresponding aggregate statistics.

Table 163. The catalog tables that contain statistics by partition and the table that contains the corresponding aggregate statistics

Statistics by partition are in	Aggregate statistics are in
SYSTABSTATS	SYSTABLES
SYSINDEXSTATS	SYSINDEXES
SYSCOLSTATS	SYSCOLUMNS
SYSCOLDISTSTATS	SYSCOLDIST

If you run RUNSTATS for separate partitions of a table space, DB2 uses the results to update the aggregate statistics for the entire table space. For recommendations about running RUNSTATS on separate partitions, see "Gathering monitor statistics and update statistics" on page 912. (You should either run RUNSTATS once on the entire object before collecting statistics on separate partitions or use the appropriate option to ensure that the statistics are aggregated appropriately, especially if some partitions are not loaded with data.)

Setting default statistics for created temporary tables

When preparing an SQL statement that refers to a created temporary table, if the table has been instantiated, DB2 uses the cardinality and number of pages maintained for that table in storage. If the table has not been instantiated, DB2 looks at the CARDF and NPAGES columns of the SYSTABLES row for the created temporary table. These values are normally -1 because RUNSTATS cannot run against a created temporary table.

You can establish default statistical values for the cardinality and number of pages if you can estimate the normal cardinality and number of pages that used the values for a particular created temporary table. You can manually update the values in the CARDF and NPAGES columns of the SYSTABLES row for the created temporary table. These values become the default values used if more accurate values are not available or more accurate values cannot be used. The more accurate values are available only for dynamic SQL statements that are prepared after the instantiation of the created temporary table, but within the same unit of work. These more accurate values are not used if the result of the dynamic bind is destined for the Dynamic Statement Cache.

History statistics

Several catalog tables provide historical statistics for other catalog tables. These catalog history tables include:

- SYSIBM.SYSCOLDIST_HIST
- SYSIBM.SYSCOLUMNS_HIST
- SYSIBM.SYSINDEXES_HIST
- SYSIBM.SYSINDEXPART_HIST
- SYSIBM.SYSINDEXSTATS_HIST
- SYSIBM.SYSLOBSTATS_HIST
- SYSIBM.SYSTABLEPART_HIST
- SYSIBM.SYSTABLES_HIST
- SYSIBM.SYSTABSTATS_HIST

For example, SYSIBM.SYSTABLESPACE_HIST provides statistics for activity in SYSIBM.SYSTABLESPACE, SYSIBM.SYSTABLEPART_HIST provides statistics for activity in SYSIBM.SYTABLEPART, and so on.

When DB2 adds or changes rows in a catalog table, DB2 might also write information about the rows to the corresponding catalog history table. Although the catalog history tables are not identical to their counterpart tables, they do contain the same columns for access path information and space utilization information. The history statistics provide a way to study trends, to determine when utilities, such as REORG, should be run for maintenance, and to aid in space management.

Table 164 lists the catalog data that are collected for historical statistics. For information on how to gather these statistics, see "Gathering monitor statistics and update statistics" on page 912.

Table 164. Catalog data collected for historical statistics

Column name	Provides access path statistics[1]	Provides space statistics	Description
SYSIBM.SYSCOLDIST_HIST			
CARDF	Yes	No	Number of distinct values gathered
COLGROUPCOLNO	Yes	No	Identifies the columns involved in multi-column statistics
COLVALUE	Yes	No	Frequently occurring value in the key distribution
FREQUENCYF	Yes	No	A number, which multiplied by 100, gives the percentage of rows that contain the value of COLVALUE
NUMCOLUMNS	Yes	No	Number of columns involved in multi-column statistics
TYPE	Yes	No	Type of statistics gathered, either cardinality (c) or frequent value (F)
SYSIBM.SYCOLUMNS_HIST			
COLCARDF	Yes	No	Estimated number of distinct values in the column
HIGH2KEY	Yes	No	Second highest value of the column, or blank
LOW2KEY	Yes	No	Second lowest value of the column, or blank
SYSIBM.SYSINDEXES_HIST			

Table 164. Catalog data collected for historical statistics (continued)

Column name	Provides access path statistics[1]	Provides space statistics	Description
CLUSTERING	Yes	No	Whether the index was created with CLUSTER
CLUSTERRATIOF	Yes	No	A number, when multiplied by 100, gives the percentage of rows in the clustering order
FIRSTKEYCARDF	Yes	No	Number of distinct values in the first key column
FULLKEYCARDF	Yes	No	Number of distinct values in the full key
NLEAF	Yes	No	Number of active leaf pages
NLEVELS	Yes	No	Number of levels in the index tree
SYSIBM.SYSINDEXPART_HIST			
CARDF	No	Yes	Number of rows or LOBs referenced
DSNUM	No	Yes	Number of data sets
EXTENTS	No	Yes	Number of data set extents (when there are multiple pieces, the value is for the extents in the last data set)
FAROFFPOSF	No	Yes	Number of rows referenced far from the optimal position
LEAFDIST	No	Yes	100 times the number of pages between successive leaf pages
LEAFFAR	No	Yes	Number of leaf pages located physically far away from previous leaf pages for successive active leaf pages accessed in an index scan
LEAFNEAR	No	Yes	Number of leaf pages located physically near previous leaf pages for successive active leaf pages
NEAROFFPOSF	No	Yes	Number of rows referenced near but not at the optimal position
PQTY	No	Yes	Primary space allocation in 4K blocks for the data set
PSEUDO_DEL_ENTRIES	No	Yes	Number of pseudo deleted keys
SECQTYI	No	Yes	Secondary space allocation in 4K blocks for the data set.
SPACEF	No	Yes	Disk storage in KB
SYSIBM.SYSINDEXSTATS_HIST			
CLUSTERRATIO	Yes	No	A number, which when multiplied by 100, gives the percentage of rows in the clustering order
FIRSTKEYCARDF	Yes	No	Number of distinct values of the first key column
FULLKEYCARDF	Yes	No	Number of distinct values of the full key
KEYCOUNTF	Yes	No	Total number of rows in the partition
NLEAF	Yes	No	Number of leaf pages
NLEVELS	Yes	No	Number of levels in the index tree
SYSIBM.SYSLOBSTATS_HIST			
FREESPACE	No	Yes	The amount of free space in the LOB table space

Table 164. Catalog data collected for historical statistics (continued)

Column name	Provides access path statistics[1]	Provides space statistics	Description
ORGRATIO	No	Yes	The percentage of organization in the LOB table space. A value of 100 indicates perfect organization of the LOB table space. A value of 1 indicates that the LOB table space is disorganized. A value of 0.00 indicates that the LOB table space is totally disorganized. An empty table space has an ORGRATIO value of 100.00.
SYSIBM.SYSTABLEPART_HIST			
CARDF	No	Yes	Number of rows in the table space or partition
DSNUM	No	Yes	Number of data sets
EXTENTS	No	Yes	Number of data set extents (when there are multiple pieces, the value is for the extents in the last data set)
FARINDREF	No	Yes	Number of rows relocated far from their original position
NEARINDREF	No	Yes	Number of rows relocated near their original position
PAGESAVE	No	Yes	Percentage of pages saved by data compression
PERCACTIVE	No	Yes	Percentage of space occupied by active pages
PERCDROP	No	Yes	Percentage of space occupied by pages from dropped tables
PQTY	No	Yes	Primary space allocation in 4K blocks for the data set
SECQTYI	No	Yes	Secondary space allocation in 4K blocks for the data set.
SPACEF	No	Yes	The number of KB of space currently used
SYSIBM.SYSTABLES_HIST			
AVGROWLEN	No	Yes	Average row length of the table specified in the table space
CARDF	Yes	No	Number of rows in the table or number of LOBs in an auxiliary table
NPAGESF	Yes	No	Number of pages used by the table
PCTPAGES	No	Yes	Percentage of pages that contain rows
PCTROWCOMP	Yes	No	Percentage of active rows compressed
SYSIBM.SYSTABSTATS_HIST			
CARDF	Yes	No	Number of rows in the partition
NPAGES	Yes	No	Total number of pages with rows

Note: [1] The access path statistics in the history tables are collected for historical purposes and are not used for access path selection.

Gathering monitor statistics and update statistics

The DB2 utility RUNSTATS can update the DB2 catalog tables with statistical information about data and indexes. For a list of the catalog columns for which RUNSTATS collects statistics, see Table 162 on page 899. For instructions on using RUNSTATS, see Part 2 of *DB2 Utility Guide and Reference*.

You can choose which DB2 catalog tables you want RUNSTATS to update: those used to optimize the performance of SQL statements or those used by database administrators to assess the status of a particular table space or index. You can monitor these catalog statistics in conjunction with EXPLAIN to make sure that your queries access data efficiently.

After you use the LOAD, REBUILD INDEX, or REORG utilities, you can gather statistics inline with those utilities by using the STATISTICS option.

Why gather statistics: Maintaining your statistics is a critical part of performance monitoring and tuning. DB2 must have correct statistical information to make the best choices for the access path.

When to gather statistics: To ensure that information in the catalog is current, gather statistics in situations in which the data or index changes significantly, such as in the following situations:

- After loading a table and before binding application plans and packages that access the table.
- After creating an index with the CREATE INDEX statement, to update catalog statistics related to the new index. (Before an application can use a new index, you must rebind the application plan or package.)
- After reorganizing a table space or an index. Then rebind plans or packages for which performance remains a concern. See "Whether to rebind after gathering statistics" on page 923 for more information. (It is not necessary to rebind after reorganizing a LOB table space, because those statistics are not used for access path selection.)
- After heavy insert, update, and delete activity. Again, rebind plans or packages for which performance is critical.
- Periodically. By comparing the output of one execution with previous executions, you can detect a performance problem early.
- Against the DB2 catalog to provide DB2 with more accurate information for access path selection of users' catalog queries.

To obtain information from the catalog tables, use a SELECT statement, or specify REPORT YES when you invoke RUNSTATS. When used routinely, RUNSTATS provides data about table spaces and indexes over a period of time. For example, when you create or drop tables or indexes or insert many rows, run RUNSTATS to update the catalog. Then rebind your applications so that DB2 can choose the most efficient access paths.

Collecting statistics by partition: You can collect statistics for a single data partition or index partition. This information allows you to avoid the cost of running utilities against unchanged partitions. When you run utilities by partition, DB2 uses the results to update the aggregate statistics for the entire table space or index. If statistics do not exist for each separate partition, DB2 can calculate the aggregate statistics only if the utilities are executed with the FORCEROLLUP YES keyword (or FORCEROLLUP keyword is omitted and the value of the STATISTICS

ROLLUP field on installation panel DSNTIPO is YES). If you do not use the keyword or installation panel field setting to force the roll up of the aggregate statistics, you must run utilities once on the entire object before running utilities on separate partitions.

Collecting history statistics: When you collect statistics with RUNSTATS or gather them inline with the LOAD, REBUILD, or REORG utilities, you can use the HISTORY option to collect history statistics. With the HISTORY option, the utility stores the statistics that were updated in the catalog tables in history records in the corresponding catalog history tables. (For information on the catalog data that is collected for history statistics, seeTable 164 on page 909.)

To remove old statistics that are no longer needed in the catalog history tables, use the MODIFY STATISTICS utility or the SQL DELETE statement. Deleting outdated information from the catalog history tables can help improve the performance of processes that access the data in these tables.

Recommendations for performance:

- To reduce the processor consumption WHEN collecting column statistics, use the SAMPLE option. The SAMPLE option allows you to specify a percentage of the rows to examine for column statistics. Consider the effect on access path selection before choosing sampling. There is likely to be little or no effect on access path selection if the access path has a matching index scan and very few predicates. However, if the access path joins of many tables with matching index scans and many predicates, the amount of sampling can affect the access path. In these cases, start with 25 percent sampling and see if there is a negative effect on access path selection. If not, you could consider reducing the sampling percent until you find the percent that gives you the best reduction in processing time without negatively affecting the access path.
- To reduce the elapsed time of gathering statistics immediately after a LOAD, REBUILD INDEX, or REORG, gather statistics inline with those utilities by using the STATISTICS option.

Updating the catalog

If you have sufficient privileges, you can change all of the values listed in Table 162 on page 899 by executing SQL UPDATE statements.

Running RUNSTATS after UPDATE: If you change values in the catalog and later run RUNSTATS to update those values, your changes are lost.

Recommendation: Keep track of the changes you make and of the plans or packages that have an access path change due to changed statistics.

Correlations in the catalog

The following relationships exist among certain columns of the catalog tables:

- Columns within table SYSCOLUMNS
- Columns in the tables SYSCOLUMNS and SYSINDEXES
- Columns in the tables SYSCOLUMNS and SYSCOLDIST
- Columns in the tables SYSCOLUMNS, SYSCOLDIST, and SYSINDEXES
- Columns with table space statistics and columns for partition-level statistics, as described in “Statistics for partitioned table spaces” on page 908.

If you plan to update some values, keep in mind the following correlations:

- COLCARDF and FIRSTKEYCARDF. For a column that is the first column of an index, those two values are equal. If the index has only that one column, the two values are also equal to the value of FULLKEYCARDF.
- COLCARDF, LOW2KEY, and HIGH2KEY. If the COLCARDF value is not '-1', DB2 assumes that statistics exist for the column. In particular, it uses the values of LOW2KEY and HIGH2KEY in calculating filter factors. If COLDARDF = 1 or if COLCARDF = 2, DB2 uses HIGH2KEY and LOW2KEY as domain statistics, and generates frequencies on HIGH2KEY and LOW2KEY.
- CARDF in SYSCOLDIST. CARDF is related to COLCARDF in SYSIBM.SYSCOLUMNS and to FIRSTKEYCARDF and FULLKEYCARDF in SYSIBM.SYSINDEXES. CARDF must be the minimum of the following:
  - A value between FIRSTKEYCARDF and FULLKEYCARDF if the index contains the same set of columns
  - A value between MAX(COLCARDF of each column in the column group) and the product of multiplying together the COLCARDF of each column in the column group

  Example: Assume the following set of statistics:

  ```
  CARDF = 1000
  NUMCOLUMNS = 3
  COLGROUPCOLNO = 2,3,5

  INDEX1 on columns 2,3,5,7,8
  FIRSTKEYCARDF = 100                CARDF must be between 100
  FULLKEYCARDF = 10000                                and 10000

  column 2 COLCARDF = 100
  column 3 COLCARDF =  50
  column 5 COLCARDF =  10
  ```

  The range between FIRSTKEYCARDF and FULLKEYCARDF is 100 and 10 000. The maximum of the COLCARDF values is 50 000. Thus, the allowable range is between 100 and 10 000.
- CARDF in SYSTABLES. CARDF must be equal to or larger than any of the other cardinalities, SUCH AS COLCARDF, FIRSTKEYCARDF, FULLKEYCARDF, and CARDF in SYSIBM.SYSCOLDIST.
- FREQUENCYF and COLCARDF or CARDF. The number of frequencies collected must be less than or equal to COLCARDF for the column or CARDF for the column group.
- FREQUENCYF. The sum of frequencies collected for a column or column group must be less than or equal to 1.

Recommendation for COLCARDF and FIRSTKEYCARDF

On partitioned indexes, RUNSTATS INDEX calculates the number of distinct column values and saves it in SYSCOLSTATS.COLCARD by partition. When the statistics by partition are used to form the aggregate, the aggregate might not be exact because some column values might occur in more than one partition. Without scanning all parts of the index, DB2 cannot detect that overlap. The overlap never skews COLCARD by more than the number of partitions, which is usually not a problem for large values. For small values, update the aggregate COLCARDF value in SYSCOLUMNS because DB2 uses the COLCARDF value when determining access paths.

The exception and the remedy for COLCARD and COLCARDF are also true for the FIRSTKEYCARDF column in SYSIBM.SYSINDEXES and the FIRSTKEYCARDF column in SYSIBM.SYSINDEXSTATS.

Recommendation for HIGH2KEY and LOW2KEY

If you update the COLCARDF value for a column, also update HIGH2KEY and LOW2KEY for the column. HIGH2KEY and LOW2KEY are defined as VARCHAR(2000); thus, an UPDATE statement must provide a character or hexadecimal value. Entering a character value is quite straightforward: SET LOW2KEY = 'ALAS', for instance. But to enter a numeric, date, or time value you must use the hexadecimal value of the DB2 internal format. See "Dates, times, and timestamps for edit and validation routines" on page 1108 and "DB2 codes for numeric data in edit and validation routines" on page 1110. Be sure to allow for a null indicator in keys that allow nulls. See also "Null values for edit procedures, field procedures, and validation routines" on page 1107.

If you update the COLCARDF value for a column, also update HIGH2KEY and LOW2KEY for the column. HIGH2KEY and LOW2KEY are defined as VARCHAR(2000). To update HIGH2KEY and LOW2KEY:

- Specify a character or hexadecimal value on the UPDATE statement.

  Entering a character value is quite straightforward: SET LOW2KEY = 'ALAS', for example. But to enter a numeric, date, or time value you must use the hexadecimal value of the DB2 internal format. See "Dates, times, and timestamps for edit and validation routines" on page 1108 and "DB2 codes for numeric data in edit and validation routines" on page 1110. Be sure to allow for a null indicator in keys that allow nulls. See also "Null values for edit procedures, field procedures, and validation routines" on page 1107.
- Ensure that the padding characteristics for the columns are the same as the padding characteristics for COLCARDF.

  For example, if the statistics values that are collected for COLCARDF are padded, then the statistics collected for HIGH2KEY or LOW2KEY must also be padded. You can check the STATS_FORMAT column in SYSCOLUMNS and SYSCOLSTATS to determine whether the statistics gathered for these columns are padded or not.

Statistics for distributions

Statistics for distributions are stored in the catalog tables SYSCOLDIST and SYSCOLDISTSTATS. Although you can use RUNSTATS to collect distribution statistics on any column or group of columns, by default, DB2 inserts the 10 most frequent values for the first key column of an index as well as the first and last key values. See Part 2 of *DB2 Utility Guide and Reference* for information about collecting more statistics related to columns that are correlated.

You can insert, update, or delete distribution information for any column in these catalog tables. However, to enter a numeric, date, or time value you must use the hexadecimal value of the DB2 internal format. See "Dates, times, and timestamps for edit and validation routines" on page 1108 and "DB2 codes for numeric data in edit and validation routines" on page 1110. Be sure to allow for a null indicator in keys that allow nulls. See also "Null values for edit procedures, field procedures, and validation routines" on page 1107.

Recommendation for using the TIMESTAMP column

Statistics gathered by RUNSTATS include timestamps. Every row updated or inserted during a particular invocation of RUNSTATS contains the same timestamp value. Update column STATSTIME whenever you update statistics in the catalog, so that you can always determine when they were last updated.

Querying the catalog for statistics

The following SELECT statements show how to retrieve some of the important statistics for access path selection. The catalog queries shown here are included in DSNTESP in SDSNSAMP and can be used as input to SPUFI. See Chapter 34, "Using EXPLAIN to improve SQL performance," on page 927 for more information about how these statistics are used in access path selection. See Appendix F of *DB2 SQL Reference* for the table definitions and descriptions of all DB2 catalog tables.

Product-sensitive Programming Interface

To access information about your data and how it is organized, use the following queries:

```
SELECT CREATOR, NAME, CARDF, NPAGES, PCTPAGES
  FROM SYSIBM.SYSTABLES
  WHERE DBNAME = 'xxx'
  AND TYPE = 'T';

SELECT NAME, UNIQUERULE, CLUSTERRATIOF, FIRSTKEYCARDF, FULLKEYCARDF,
  NLEAF, NLEVELS, PGSIZE
  FROM SYSIBM.SYSINDEXES
  WHERE DBNAME = 'xxx';

SELECT NAME, DBNAME, NACTIVE, CLOSERULE, LOCKRULE
  FROM SYSIBM.SYSTABLESPACE
  WHERE DBNAME = 'xxx';

SELECT NAME, TBNAME, COLCARDF, HIGH2KEY, LOW2KEY, HEX(HIGH2KEY),
  HEX(LOW2KEY)
  FROM SYSIBM.SYSCOLUMNS
  WHERE TBCREATOR = 'xxx' AND COLCARDF <> -1;

SELECT NAME, FREQUENCYF, COLVALUE, HEX(COLVALUE), CARDF,
  COLGROUPCOLNO, HEX(COLGROUPCOLNO), NUMCOLUMNS, TYPE
  FROM SYSIBM.SYSCOLDIST
  WHERE TBNAME = 'ttt'
  ORDER BY NUMCOLUMNS, NAME, COLGROUPCOLNO, TYPE, FREQUENCYF DESC;

SELECT NAME, TSNAME, CARD, NPAGES
   FROM SYSIBM.SYSTABSTATS
   WHERE DBNAME='xxx';
```

End of Product-sensitive Programming Interface

If the statistics in the DB2 catalog no longer correspond to the true organization of your data, you should reorganize the necessary tables, run RUNSTATS, and rebind the plans or packages that contain any affected queries. See "When to reorganize indexes and table spaces" on page 920 and the description of REORG in Part 2 of *DB2 Utility Guide and Reference* for information on how to determine which table spaces and indexes qualify for reorganization. This includes the DB2 catalog table spaces as well as user table spaces. Then DB2 has accurate information to choose appropriate access paths for your queries. Use the EXPLAIN statement to verify the chosen access paths for your queries.

Improving index and table space access

Statistics from the DB2 catalog help determine the most economical access path. The statistics described in this section are used to determine index access cost and are found in the corresponding columns of the SYSIBM.SYSINDEXES catalog table.

The statistics show distribution of data within the allocated space, from which you can judge clustering and the need to reorganize.

Space utilization statistics can also help you make sure that access paths that use the index or table space are as efficient as possible. By reducing gaps between leaf pages in an index, or to ensure that data pages are close together, you can reduce sequential I/Os.

Recommendation: To provide the most accurate data, gather statistics routinely to provide data about table spaces and indexes over a period of time.

Recommendation: Run RUNSTATS some time **after** reorganizing the data or indexes. By gathering the statistics after you reorganize, you ensure that access paths reflect a more "average" state of the data.

This section describes the following topics:

- "How clustering affects access path selection"
- "What other statistics provide index costs" on page 919
- "When to reorganize indexes and table spaces" on page 920
- "Whether to rebind after gathering statistics" on page 923

How clustering affects access path selection

In general, CLUSTERRATIOF gives an indication of how closely the order of the index entries on the index leaf pages matches the actual ordering of the rows on the data pages. The closer CLUSTERRATIOF is to 100%, the more closely the ordering of the index entries matches the actual ordering of the rows on the data pages. The actual formula is quite complex and accounts for indexes with many duplicates; in general, for a given index, the more duplicates, the higher the CLUSTERRATIOF value.

Here are some things to remember about the effect of CLUSTERRATIOF on access paths:

- CLUSTERRATIOF is an important input to the cost estimates that are used to determine whether an index is used for an access path, and, if so, which index to use.
- If the access is INDEXONLY, then this value does not apply.
- The higher the CLUSTERRATIOF value, the lower the cost of referencing data pages during an index scan is.
- For an index that has a CLUSTERRATIOF less than 80%, sequential prefetch is not used to access the data pages.
- A slight reduction in CLUSTERRATIOF for a table with a large number of rows can represent a much more significant number of unclustered rows than for a table with a small number of rows.

  Example: A CLUSTERRATIOF of 99% for a table with 100 000 000 rows represents 100 000 unclustered rows. Whereas, the CLUSTERRATIOF of 95% for a table with 100 000 rows represents 5000 unclustered rows.

Figure 98 on page 918 shows an index scan on an index with a high cluster ratio. Compare that with Figure 99 on page 919, which shows an index scan on an index with a low cluster ratio.

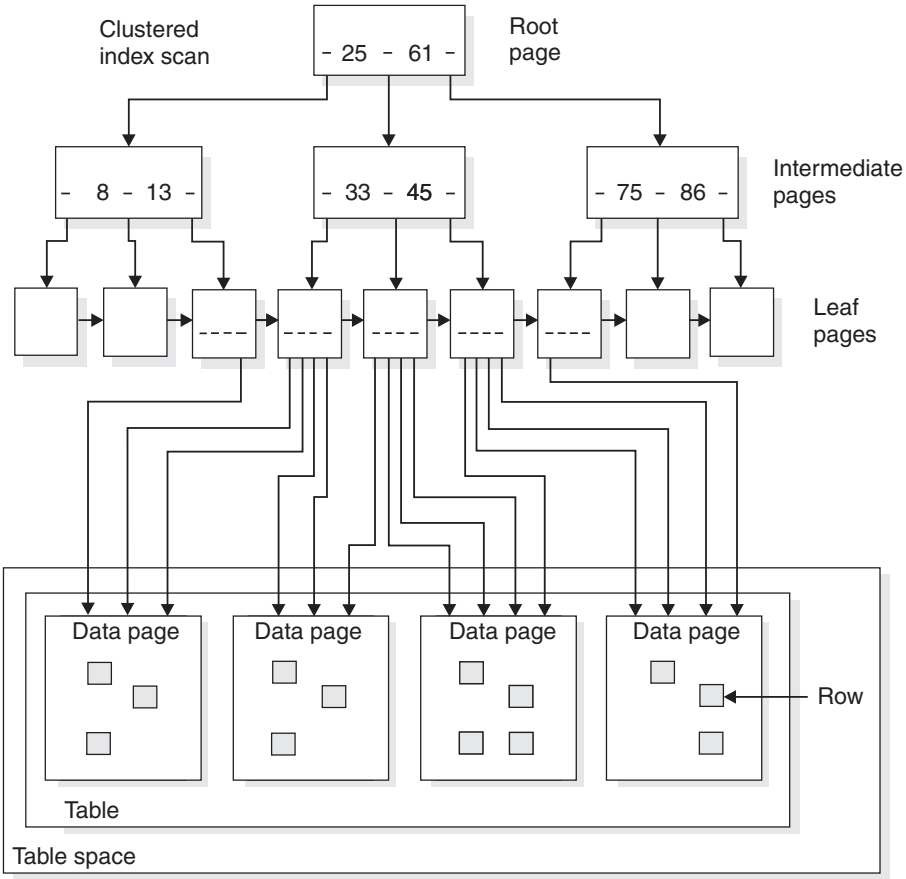


Figure 98. A clustered index scan. This figure assumes that the index is 100% clustered.

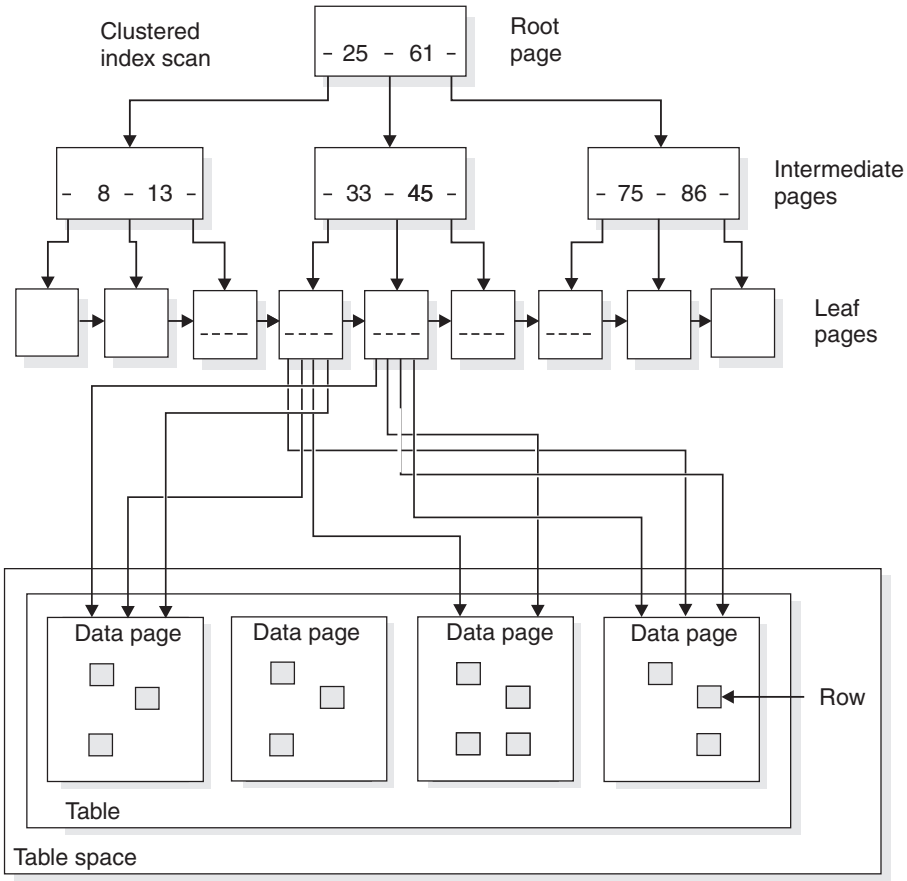


Figure 99. A nonclustered index scan. In some cases, DB2 can access the data pages in order even when a nonclustered index is used.

What other statistics provide index costs

The following statistics in SYSINDEXES also give information about costs to process the index.

FIRSTKEYCARDF: The number of distinct values of the first index key column. When an indexable equal predicate is specified on the first index key column, 1/FIRSTKEYCARDF is the filter factor for the predicate and the index. The higher the number is, the less the cost is.

FULLKEYCARDF: The number of distinct values for the entire index key. When indexable equal predicates are specified on all the index key columns, 1/FULLKEYCARDF is the filter factor for the predicates and the index. The higher the number is, the less the cost is.

When the number of matching columns is greater than 1 and less than the number of index key columns, the filtering of the index is located between 1/FIRSTKEYCARDF and 1/FULLKEYCARDF.

NLEAF: The number of active leaf pages in the index. NLEAF is a portion of the cost to scan the index. The smaller the number is, the less the cost is. It is also less when the filtering of the index is high, which comes from FIRSTKEYCARDF, FULLKEYCARDF, and other indexable predicates.

NLEVELS: The number of levels in the index tree. NLEVELS is another portion of the cost to traverse the index. The same conditions as NLEAF apply. The smaller the number is, the less the cost is.

When to reorganize indexes and table spaces

Data that is organized well physically can improve the performance of access paths that rely on index or data scans. Well-organized data can also help reduce the amount of disk storage used by the index or table space. If your main reason for reorganizing is performance, the best way to determine when to reorganize is to watch your statistics for increased I/O, getpages, and processor consumption. When performance degrades to an unacceptable level, analyze the statistics described in the guidelines in this section to help you develop your own rules for when to reorganize in your particular environment. Consider the following general guidelines for when to run the REORG utility:

- ***Using useful catalog queries:*** Catalog queries you can use to help you determine when to reorganize are included in DSNTESP in SDSNSAMP and can be used as input to SPUFI.
- ***Using the REORG utility to determine whether to reorganize:*** The REORG utility imbeds the function of catalog queries. If a query returns a certain result (you can use the default or supply your own), REORG will either reorganize or not. Optionally, you can have REORG run a report instead of actually doing the reorganization.

  The REORG options that imbed these catalog queries are:

  – OFFPOSLIMIT and INDREFLIMIT options of REORG TABLESPACE

  – LEAFDISTLIMIT of REORG INDEX

  The REORG utility does not imbed any function to help you determine when to reorganize LOB table spaces.
- ***Reorganizing after table definition changes:*** Another time to consider reorganizing data to improve performance is when ALTER TABLE statements have been used to add a column to the table or to change the data types or the lengths of existing columns. Such changes cause the table space to be placed in advisory REORG-pending (AREO*) status.

  In the case of changing the definition of existing column, the table space is placed in AREO* status because the existing data is not immediately converted to its new definition. Reorganizing the table space causes the rows to be reloaded with the data converted to the new definition. Until the table space is reorganized, the changes must be tracked and applied as the data is accessed, possibly degrading performance. For example, depending on the number of changes, you may see decreased performance for dynamic SQL queries, updates and deletes, other ALTER statements (especially those that are run concurrently). In addition, running multiple REORG and LOAD utilities concurrently may perform slower or create timeouts. It may also take longer to unload a table that has had many changes prior to it being reorganized.

See Part 2 of *DB2 Utility Guide and Reference* for more information about the REORG utility.

Reorganizing Indexes

To understand index organization, you must understand the LEAFNEAR and LEAFFAR columns of SYSIBM.SYSINDEXPART. This section describes how to interpret those values and then describes some rules of thumb for determining when to reorganize the index.

LEAFNEAR and LEAFFAR columns: The LEAFNEAR and LEAFFAR columns of SYSIBM.SYSINDEXPART measure the disorganization of physical leaf pages by indicating the number of pages that are not in an optimal position. Leaf pages can have page gaps whenever index pages are deleted or when there are index leaf page splits caused by an insert that cannot fit onto a full page. If the key cannot fit on the page, DB2 moves half the index entries onto a new page, which might be far away from the "home" page.

Figure 100 shows the logical and physical view of an index.

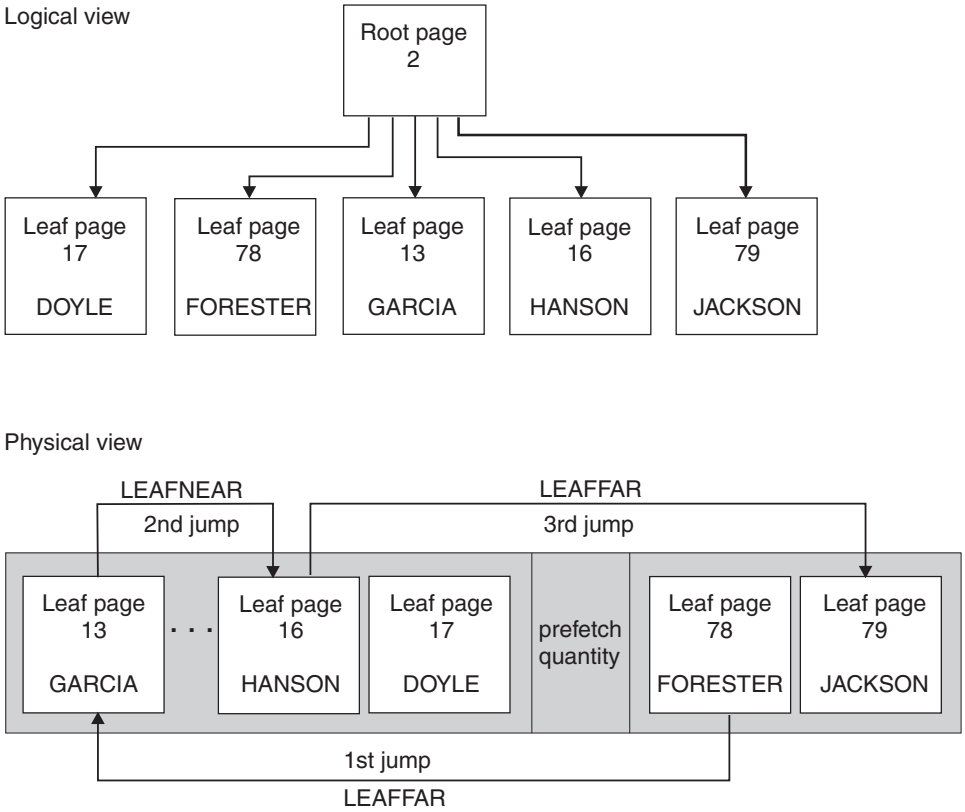


Figure 100. Logical and physical views of an index in which LEAFNEAR=1 and LEAFFAR=2

The logical view at the top of the figure shows that for an index scan four leaf pages need to be scanned to access the data for FORESTER through JACKSON. The physical view at the bottom of the figure shows how the pages are physically accessed. The first page is at physical leaf page 78, and the other leaf pages are at physical locations 79, 13, and 16. A jump forward or backward of more than one page represents non-optimal physical ordering. LEAFNEAR represents the number of jumps within the prefetch quantity, and LEAFFAR represents the number of jumps outside the prefetch quantity. In this example, assuming that the prefetch quantity is 32, there are two jumps outside the prefetch quantity. A jump from page 78 to page 13, and one from page 16 to page 79. Thus, LEAFFAR is 2. Because of the jump within the prefetch quantity from page 13 to page 16, LEAFNEAR is 1.

LEAFNEAR has a smaller impact than LEAFFAR because the LEAFNEAR pages, which are located within the prefetch quantity, are typically read by prefetch without incurring extra I/Os.

The optimal value of the LEAFNEAR and LEAFFAR catalog columns is zero. However, immediately after you run the REORG and gather statistics, LEAFNEAR for a large index might be greater than zero. A non-zero value could be caused by free pages that result from the FREEPAGE option on CREATE INDEX, non-leaf pages, or various system pages; the jumps over these pages are included in LEAFNEAR.

Recommendations: Fields in the SYSIBM.INDEXSPACESTATS table can help you determine when to reorganize an index.

Consider running REORG INDEX when any of the following conditions are true. These conditions are based on values in the SYSIBM.INDEXSPACESTATS table.

- REORGPSEUDODELETES/TOTALENTRIES > 10% in a data sharing environment, or REORGPSEUDODELETES/TOTALENTRIES > 5% in a data sharing environment.
- REORGLEAFFAR/NACTIVE > 10%
- REORGINSERTS/TOTALENTRIES > 25%
- REORGDELETES/TOTALENTRIES > 25%
- REORGAPPENDINSERT/TOTALENTRIES > 20%
- EXTENTS > 254

You should also consider running REORG if one of the following conditions are true:

- The index is in the advisory REORG-pending state (AREO*) as a result of an ALTER statement.
- An index is in the advisory REBUILD-pending state (ARBDP) as a result an ALTER statement.

Reorganizing table spaces

Fields in the SYSIBM.TABLESPACESTATS table can help you determine when to reorganize a table space.

Consider running REORG TABLESPACE when any of the following conditions are true. These conditions are based on values in the SYSIBM.TABLESPACESTATS table.

- REORGUNCLUSTINS/TOTALROWS > 10%

  Do not use REORGUNCLUSTINS to determine if you should run REORG if access to the table space is predominantly random access.
- (REORGNEARINDREF+REORGFARINDREF)/TOTALROWS > 5% in a data sharing environment, or (REORGNEARINDREF+REORGFARINDREF)/ TOTALROWS >10% in non-data sharing environment
- REORGINSERTS/TOTALROWS > 25%
- REORGDELETES/TOTALROWS > 25%
- EXTENTS > 254
- REORGDISORGLOB/TOTALROWS > 50%
- SPACE > 2 * (DATASIZE / 1024)
- REORGMASSDELETE > 0

You should also consider running REORG if one of the following conditions are true:

- The table space is in the advisory REORG-pending state (AREO*) as a result of an ALTER TABLE statement.

- An index on a table in the table space is in the advisory REBUILD-pending state (ARBDP) as result an ALTER TABLE statement.

Reorganizing LOB table spaces

SYSIBM.SYSLOBSTATS contains information about how the data in the table space is physically stored. Consider running REORG on the LOB table space when the percentage in the ORGRATIO column is less than 80.

Additionally, you can use real-time statistics to identify DB2 objects that should be reorganized, have their statistics updated, or be image copied.

Whether to rebind after gathering statistics

It is not always necessary to rebind all applications after you gather statistics. A rebind is necessary only if the access path statistics change significantly from the last time you bound the applications and if performance suffers as a result.

When performance degrades to an unacceptable level, analyze the statistics described in the recommendations in this section to help you develop your own guidelines for when to rebind.

Consider the following guidelines regarding when to rebind:

- CLUSTERRATIOF changes to less or more than 80% (a value of 0.80).
- NLEAF changes more than 20% from the previous value.
- NLEVELS changes (only if it was more than a two-level index to begin with).
- NPAGES changes more than 20% from the previous value.
- NACTIVEF changes more than 20% from the previous value.
- The range of HIGH2KEY to LOW2KEY range changes more than 20% from the range previously recorded.
- Cardinality changes more than 20% from previous range.
- Distribution statistics change the majority of the frequent column values.

Modeling your production system

To see what access paths your production queries will use, consider updating the catalog statistics on your test system to be the same as your production system.

To do that, run RUNSTATS on your production tables to get current statistics for access path selection. Then retrieve them and use them to build SQL statements to update the catalog of the test system.

Example: You can use queries similar to the following queries to build those statements. To successfully model your production system, the table definitions must be the same on both the test and production systems. For example, they must have the same creator, name, indexes, number of partitions, and so on.

Use the following statements to update SYSTABLESPACE, SYSTABLES, SYSINDEXES, and SYSCOLUMNS:

```
SELECT DISTINCT 'UPDATE SYSIBM.SYSTABLESPACE SET NACTIVEF='
CONCAT STRIP(CHAR(NACTIVEF))
CONCAT',NACTIVE='CONCAT STRIP(CHAR(NACTIVE))
CONCAT ' WHERE NAME=''' CONCAT TS.NAME
CONCAT ''' AND DBNAME =''''CONCAT TS.DBNAME CONCAT''''*'
FROM SYSIBM.SYSTABLESPACE TS, SYSIBM.SYSTABLES TBL
WHERE TS.NAME = TSNAME
```

```
      AND TBL.CREATOR IN (table creator_list)
      AND TBL.NAME IN (table_list)
      AND (NACTIVEF >=0 OR NACTIVE  >=0);

SELECT 'UPDATE SYSIBM.SYSTABLES SET CARDF='
CONCAT  STRIP(CHAR(CARDF))
CONCAT',NPAGES='CONCAT STRIP(CHAR(NPAGES))
CONCAT',PCTROWCOMP='CONCAT STRIP(CHAR(PCTROWCOMP))
CONCAT ' WHERE NAME='''CONCAT NAME
CONCAT ''' AND CREATOR ='''CONCAT CREATOR CONCAT'''*'
FROM SYSIBM.SYSTABLES WHERE
CREATOR IN (creator_list)
        AND NAME IN (table_list)
        AND CARDF >= 0;

SELECT 'UPDATE SYSIBM.SYSINDEXES SET FIRSTKEYCARDF='
CONCAT  STRIP(CHAR(FIRSTKEYCARDF))
CONCAT ',FULLKEYCARDF='CONCAT STRIP(CHAR(FULLKEYCARDF))
CONCAT',NLEAF='CONCAT STRIP(CHAR(NLEAF))
CONCAT',NLEVELS='CONCAT STRIP(CHAR(NLEVELS))
CONCAT',CLUSTERRATIO='CONCAT STRIP(CHAR(CLUSTERRATIO))
CONCAT',CLUSTERRATIOF='CONCAT STRIP(CHAR(CLUSTERRATIOF))
CONCAT' WHERE NAME='''CONCAT NAME
CONCAT ''' AND CREATOR ='''CONCAT CREATOR CONCAT'''*'
FROM SYSIBM.SYSINDEXES
WHERE TBCREATOR IN (creator_list)
      AND TBNAME IN (table_list)
      AND FULLKEYCARDF >= 0;

SELECT 'UPDATE SYSIBM.SYSCOLUMNS SET COLCARDF='
CONCAT STRIP(CHAR(COLCARDF))
CONCAT',HIGH2KEY= X''' CONCAT HEX(HIGH2KEY)
CONCAT''',LOW2KEY= X''' CONCAT HEX(LOW2KEY)
CONCAT''' WHERE TBNAME='''' CONCAT TBNAME CONCAT  ''' AND COLNO='
CONCAT STRIP(CHAR(COLNO))
CONCAT ' AND TBCREATOR ='''' CONCAT TBCREATOR CONCAT'''*'
FROM SYSIBM.SYSCOLUMNS
WHERE TBCREATOR IN (creator_list)
      AND TBNAME IN (table_list)
      AND COLCARDF >= 0;
```

SYSTABSTATS and SYSCOLDIST require deletes and inserts.

Delete statistics from SYSTABSTATS on the test subsystem for the specified tables by using the following statement:

```
DELETE FROM (TEST_SUBSYSTEM).SYSTABSTATS
   WHERE OWNER IN (creator_list)
   AND   NAME IN (table_list)
```

Use INSERT statements to repopulate SYSTABSTATS with production statistics that are generated from the following statement:

```
SELECT 'INSERT INTO SYSIBM.SYSTABSTATS'
CONCAT '(CARD,NPAGES,PCTPAGES,NACTIVE,PCTROWCOMP'
CONCAT ',STATSTIME,IBMREQD,DBNAME,TSNAME,PARTITION'
CONCAT ',OWNER,NAME,CARDF) VALUES('
CONCAT  STRIP(CHAR(CARD))             CONCAT ' ,'
CONCAT  STRIP(CHAR(NPAGES))           CONCAT ' ,'
CONCAT  STRIP(CHAR(PCTPAGES))         CONCAT ' ,'
CONCAT  STRIP(CHAR(NACTIVE))          CONCAT ' ,'
CONCAT  STRIP(CHAR(PCTROWCOMP))       CONCAT ' ,'
CONCAT  '''' CONCAT CHAR(STATSTIME)   CONCAT ''' ,'
CONCAT  '''' CONCAT IBMREQD           CONCAT ''' ,'
CONCAT  '''' CONCAT STRIP(DBNAME)     CONCAT ''' ,'
CONCAT  '''' CONCAT STRIP(TSNAME)     CONCAT ''' ,'
CONCAT  STRIP(CHAR(PARTITION))        CONCAT ' ,'
```

```
CONCAT  '''' CONCAT STRIP(OWNER)      CONCAT ''' ,'
CONCAT '''' CONCAT STRIP(NAME)        CONCAT ''' ,'
CONCAT  STRIP(CHAR(CARDF))            CONCAT  ')*'
FROM  SYSIBM.SYSTABSTATS
      WHERE OWNER IN (creator_list)
      AND   NAME IN (table_list);
```

Delete statistics from SYSCOLDIST on the test subsystem for the specified tables by using the following statement:

```
DELETE FROM (TEST_SUBSYSTEM).SYSCOLDIST
   WHERE TBOWNER IN (creator_list)
   AND   TBNAME IN (table_list);
```

Use INSERT statements to repopulate SYSCOLDIST with production statistics that are generated from the following statement:

```
SELECT 'INSERT INTO SYSIBM.SYSCOLDIST '
CONCAT '(FREQUENCY,STATSTIME,IBMREQD,TBOWNER'
CONCAT ',TBNAME,NAME,COLVALUE,TYPE,CARDF,COLGROUPCOLNO'
CONCAT ',NUMCOLUMNS,FREQUENCYF) VALUES( '
CONCAT  STRIP(CHAR(FREQUENCY))            CONCAT ' ,'
CONCAT  '''' CONCAT CHAR(STATSTIME)       CONCAT ''' ,'
CONCAT  '''' CONCAT IBMREQD               CONCAT ''' ,'
CONCAT  '''' CONCAT STRIP(TBOWNER)        CONCAT ''' ,'
CONCAT  '''' CONCAT STRIP(TBNAME)         CONCAT ''','
CONCAT  '''' CONCAT STRIP(NAME)           CONCAT ''' ,'
CONCAT  'X''' CONCAT STRIP(HEX(COLVALUE)) CONCAT ''' ,'
CONCAT  '''' CONCAT TYPE                  CONCAT ''' ,'
CONCAT  STRIP(CHAR(CARDF))                CONCAT ' ,'
CONCAT  'X'''CONCAT STRIP(HEX(COLGROUPCOLNO)) CONCAT ''' ,'
CONCAT  CHAR(NUMCOLUMNS)                  CONCAT ' ,'
CONCAT  STRIP(CHAR(FREQUENCYF))           CONCAT  ')*'
FROM  SYSIBM.SYSCOLDIST
      WHERE TBOWNER IN (creator_list)
      AND   TBNAME IN (table_list);
```

Note about SPUFI:

- If you use SPUFI to execute the preceding SQL statements, you might need to increase the default maximum character column width to avoid truncation.
- Asterisks (*) appear in the examples to avoid having the semicolon interpreted as the end of the SQL statement. Edit the result to change the asterisk to a semicolon.

Access path differences from test to production: When you bind applications on the test system with production statistics, access paths should be similar but still may be different to what you see when the same query is bound on your production system. The access paths from test to production could be different for the following possible reasons:

- The processor models are different.
- The number of processors are different. (Differences in the number of processors can affect the degree of parallelism that is obtained.)
- The buffer pool sizes are different.
- The RID pool sizes are different.
- Data in SYSIBM.SYSCOLDIST is mismatched. (This mismatch occurs only if some of the previously mentioned steps mentioned are not followed exactly).
- The service levels are different.
- The values of optimization subsystem parameters, such as STARJOIN, NPGTHRSH, and PARAMDEG (MAX DEGREE on installation panel DSNTIP4) are different.

- The use of techniques such as optimization hints and volatile tables are different.

Tools to help: If your production system is accessible from your test system, you can use DB2 PM EXPLAIN on your test system to request EXPLAIN information from your production system. This request can reduce the need to simulate a production system by updating the catalog.

You can also use the DB2 Visual Explain feature to display the current PLAN_TABLE output or the graphed access paths for statements within any particular subsystem from your workstation environment. For example, if you have your test system on one subsystem and your production system on another subsystem, you can visually compare the PLAN_TABLE outputs or access paths simultaneously with some window or view manipulation. You can then access the catalog statistics for certain referenced objects of an access path from either of the displayed PLAN_TABLEs or access path graphs. For information on using Visual Explain, see DB2 Visual Explain online help.

Chapter 34. Using EXPLAIN to improve SQL performance

The information under this heading, up to the end of this chapter, is Product-sensitive Programming Interface and Associated Guidance Information, as defined in "Notices" on page 1301.

Definitions and purpose: EXPLAIN is a monitoring tool that produces information about the following:

- A plan, package, or SQL statement when it is bound. The output appears in a table that you create called PLAN_TABLE, which is also called a *plan table*. For experienced users, you can use PLAN_TABLE to give optimization hints to DB2. See "Giving optimization hints to DB2" on page 801 for more information.
- An estimated cost of executing an SQL SELECT, INSERT, UPDATE, or DELETE statement. The output appears in a table that you create called DSN_STATEMNT_TABLE, which is also called a *statement table*. For more information about statement tables, see "Estimating a statement's cost" on page 981.
- User-defined functions referred to in the statement, including the specific name and schema. The output appears in a table that you create called DSN_FUNCTION_TABLE, which is also called a *function table*. For more information about function tables, see Part 3 of *DB2 Application Programming and SQL Guide*.

Other tools: The following tools can help you tune SQL queries:

- **DB2 Visual Explain**

  Visual Explain is a graphical workstation feature of DB2 that provides:
  - An easy-to-understand display of a selected access path
  - Suggestions for changing an SQL statement
  - An ability to invoke EXPLAIN for dynamic SQL statements
  - An ability to provide DB2 catalog statistics for referenced objects of an access path
  - A subsystem parameter browser with keyword 'Find' capabilities

  For information about using DB2 Visual Explain, which is a separately packaged CD-ROM provided with your DB2 UDB for z/OS Version 8 license, see *DB2 Visual Explain online help*.
- **OMEGAMON Performance Expert**

  OMEGAMON is a performance monitoring tool that formats performance data. OMEGAMON combines information from EXPLAIN and from the DB2 catalog. It displays access paths, indexes, tables, table spaces, plans, packages, DBRMs, host variable definitions, ordering, table access and join sequences, and lock types. Output is presented in a dialog rather than as a table, making the information easy to read and understand. DB2 Performance Monitor (DB2 PM) performs some of the functions of OMEGAMON Perfomance Expert. For more information about OMEGAMON and DB2 PM, see "OMEGAMON" on page 1196.
- **DB2 Estimator**

  DB2 Estimator for Windows is an easy-to-use, stand-alone tool for estimating the performance of DB2 UDB for z/OS applications. You can use it to predict the performance and cost of running the applications, transactions, SQL statements,

© Copyright IBM Corp. 1982, 2008

triggers, and utilities. For instance, you can use DB2 Estimator for estimating the impact of adding or dropping an index from a table, estimating the change in response time from adding processor resources, and estimating the amount of time a utility job will take to run. DB2 Estimator for Windows can be downloaded from the Web.

- DB2-supplied EXPLAIN stored procedure. Users with authority to run EXPLAIN directly can obtain access path information by calling the DB2-supplied EXPLAIN stored procedure. For more information about the DB2-supplied EXPLAIN stored procedure, see Appendix J, "DB2-supplied stored procedures," on page 1255.

Chapter overview: This chapter includes the following topics:

- "Obtaining PLAN_TABLE information from EXPLAIN"
- "Asking questions about data access" on page 939
- "Interpreting access to a single table" on page 947
- "Interpreting access to two or more tables (join)" on page 955
- "Interpreting data prefetch" on page 969
- "Determining sort activity" on page 973
- "Processing for views and nested table expressions" on page 975
- "Estimating a statement's cost" on page 981

See also Chapter 35, "Parallel operations and query performance," on page 987.

Obtaining PLAN_TABLE information from EXPLAIN

The information in PLAN_TABLE can help you to:

- Design databases, indexes, and application programs
- Determine when to rebind an application
- Determine the access path chosen for a query

For each access to a single table, EXPLAIN tells you if an index access or table space scan is used. If indexes are used, EXPLAIN tells you how many indexes and index columns are used and what I/O methods are used to read the pages. For joins of tables, EXPLAIN tells you which join method and type are used, the order in which DB2 joins the tables, and when and why it sorts any rows.

The primary use of EXPLAIN is to observe the access paths for the SELECT parts of your statements. For UPDATE and DELETE WHERE CURRENT OF, and for INSERT, you receive somewhat less information in your plan table. And some accesses EXPLAIN does not describe: for example, the access to LOB values, which are stored separately from the base table, and access to parent or dependent tables needed to enforce referential constraints.

The access paths shown for the example queries in this chapter are intended only to illustrate those examples. If you execute the queries in this chapter on your system, the access paths chosen can be different.

Steps to obtain PLAN_TABLE information: Use the following overall steps to obtain information from EXPLAIN:

1. Have appropriate access to a plan table. To create the table, see "Creating PLAN_TABLE" on page 929.
2. Populate the table with the information you want. For instructions, see "Populating and maintaining a plan table" on page 936.
3. Select the information you want from the table. For instructions, see "Reordering rows from a plan table" on page 939.

Creating PLAN_TABLE

Before you can use EXPLAIN, a PLAN_TABLE must be created to hold the results of EXPLAIN. A copy of the statements that are needed to create the table are in the DB2 sample library, under the member name DSNTESC. (Unless you need the information that they provide, you do not need to create a function table or statement table to use EXPLAIN.)

Figure 101 on page 930 shows most current format of a plan table, which consists of 58 columns. Table 165 on page 932 shows the content of each column.

```
CREATE TABLE userid.PLAN_TABLE
       (QUERYNO             INTEGER        NOT NULL,
        QBLOCKNO            SMALLINT       NOT NULL,
        APPLNAME            CHAR(8)        NOT NULL,
|       PROGNAME            VARCHAR(128)   NOT NULL,
        PLANNO              SMALLINT       NOT NULL,
        METHOD              SMALLINT       NOT NULL,
|       CREATOR             VARCHAR(128)   NOT NULL,
|       TNAME               VARCHAR(128)   NOT NULL,
        TABNO               SMALLINT       NOT NULL,
        ACCESSTYPE          CHAR(2)        NOT NULL,
        MATCHCOLS           SMALLINT       NOT NULL,
|       ACCESSCREATOR       VARCHAR(128)   NOT NULL,
|       ACCESSNAME          VARCHAR(128)   NOT NULL,
        INDEXONLY           CHAR(1)        NOT NULL,
        SORTN_UNIQ          CHAR(1)        NOT NULL,
        SORTN_JOIN          CHAR(1)        NOT NULL,
        SORTN_ORDERBY       CHAR(1)        NOT NULL,
        SORTN_GROUPBY       CHAR(1)        NOT NULL,
        SORTC_UNIQ          CHAR(1)        NOT NULL,
        SORTC_JOIN          CHAR(1)        NOT NULL,
        SORTC_ORDERBY       CHAR(1)        NOT NULL,
        SORTC_GROUPBY       CHAR(1)        NOT NULL,
        TSLOCKMODE          CHAR(3)        NOT NULL,
        TIMESTAMP           CHAR(16)       NOT NULL,
|       REMARKS             VARCHAR(762)   NOT NULL,
|       PREFETCH            CHAR(1)        NOT NULL WITH DEFAULT,
#       COLUMN_FN_EVAL      CHAR(1)        NOT NULL WITH DEFAULT,
#       MIXOPSEQ            SMALLINT       NOT NULL WITH DEFAULT,
#       VERSION             VARCHAR(64)    NOT NULL WITH DEFAULT,
#       COLLID              VARCHAR(128)   NOT NULL WITH DEFAULT,
        ACCESS_DEGREE       SMALLINT       ,
        ACCESS_PGROUP_ID    SMALLINT       ,
        JOIN_DEGREE         SMALLINT       ,
        JOIN_PGROUP_ID      SMALLINT       ,
        SORTC_PGROUP_ID     SMALLINT       ,
        SORTN_PGROUP_ID     SMALLINT       ,
        PARALLELISM_MODE    CHAR(1)        ,
        MERGE_JOIN_COLS     SMALLINT       ,
#       CORRELATION_NAME    VARCHAR(128)   ,
        PAGE_RANGE          CHAR(1)        NOT NULL WITH DEFAULT,
        JOIN_TYPE           CHAR(1)        NOT NULL WITH DEFAULT,
        GROUP_MEMBER        CHAR(8)        NOT NULL WITH DEFAULT,
|       IBM_SERVICE_DATA    VARCHAR(254)   FOR BIT DATA NOT NULL WITH DEFAULT,
        WHEN_OPTIMIZE       CHAR(1)        NOT NULL WITH DEFAULT,
        QBLOCK_TYPE         CHAR(6)        NOT NULL WITH DEFAULT,
        BIND_TIME           TIMESTAMP      NOT NULL WITH DEFAULT,
|       OPTHINT             VARCHAR(128)   NOT NULL WITH DEFAULT,
|       HINT_USED           VARCHAR(128)   NOT NULL WITH DEFAULT,
        PRIMARY_ACCESSTYPE  CHAR(1)        NOT NULL WITH DEFAULT,
        PARENT_QBLOCKNO     SMALLINT       NOT NULL WITH DEFAULT,
        TABLE_TYPE          CHAR(1)        ,
|       TABLE_ENCODE        CHAR(1)        NOT NULL WITH DEFAULT,
|       TABLE_SCCSID        SMALLINT       NOT NULL WITH DEFAULT,
|       TABLE_MCCSID        SMALLINT       NOT NULL WITH DEFAULT,
|       TABLE_DCCSID        SMALLINT       NOT NULL WITH DEFAULT,
|       ROUTINE_ID          INTEGER        NOT NULL WITH DEFAULT,
|       CTEREF              SMALLINT       NOT NULL WITH DEFAULT,
|       STMTTOKEN           VARCHAR(240))
    IN database-name.table-space-name
    CCSID EBCDIC;
```

Figure 101. 58-column format of PLAN_TABLE

Your plan table can use many other formats with fewer columns, as shown in Figure 102. However, use the 58-column format because it gives you the most information. If you alter an existing plan table with fewer than 58 columns to the 58-column format:

- If they exist, change the data type of columns: PROGNAME, CREATOR, TNAME, ACCESSTYPE, ACCESSNAME, REMARKS, COLLID, CORRELATION_NAME, IBM_SERVICE_DATA, OPTHINT, and HINT_USED. Use the values shown in Figure 101 on page 930.
- Add the missing columns to the table. Use the column definitions shown in Figure 101 on page 930. For most columns added, specify NOT NULL WITH DEFAULT so that default values are included for the rows in the table. However, as the figure shows, certain columns do allow nulls. Do not specify those columns as NOT NULL WITH DEFAULT.

```
QUERYNO         INTEGER       NOT NULL
QBLOCKNO        SMALLINT      NOT NULL
APPLNAME        CHAR(8)       NOT NULL
PROGNAME        CHAR(8)       NOT NULL
PLANNO          SMALLINT      NOT NULL
METHOD          SMALLINT      NOT NULL
CREATOR         CHAR(8)       NOT NULL
TNAME           CHAR(18)      NOT NULL
TABNO           SMALLINT      NOT NULL
ACCESSTYPE      CHAR(2)       NOT NULL
MATCHCOLS       SMALLINT      NOT NULL
ACCESSCREATOR   CHAR(8)       NOT NULL
ACCESSNAME      CHAR(18)      NOT NULL
INDEXONLY       CHAR(1)       NOT NULL
SORTN_UNIQ      CHAR(1)       NOT NULL
SORTN_JOIN      CHAR(1)       NOT NULL
SORTN_ORDERBY   CHAR(1)       NOT NULL
SORTN_GROUPBY   CHAR(1)       NOT NULL
SORTC_UNIQ      CHAR(1)       NOT NULL
SORTC_JOIN      CHAR(1)       NOT NULL
SORTC_ORDERBY   CHAR(1)       NOT NULL
SORTC_GROUPBY   CHAR(1)       NOT NULL
TSLOCKMODE      CHAR(3)       NOT NULL
TIMESTAMP       CHAR(16)      NOT NULL
REMARKS         VARCHAR(254)  NOT NULL
  ----------------25 column format---------------
PREFETCH        CHAR(1)       NOT NULL WITH DEFAULT
COLUMN_FN_EVAL  CHAR(1)       NOT NULL WITH DEFAULT
MIXOPSEQ        SMALLINT      NOT NULL WITH DEFAULT
  ----------------28 column format---------------
VERSION         VARCHAR(64)   NOT NULL WITH DEFAULT
COLLID          CHAR(18)      NOT NULL WITH DEFAULT
  ----------------30 column format---------------
ACCESS_DEGREE      SMALLINT
ACCESS_PGROUP_ID   SMALLINT
JOIN_DEGREE        SMALLINT
JOIN_PGROUP_ID     SMALLINT
   ------------------34 column format----------------
SORTC_PGROUP_ID    SMALLINT
SORTN_PGROUP_ID    SMALLINT
PARALLELISM_MODE   CHAR(1)
MERGE_JOIN_COLS    SMALLINT
CORRELATION_NAME   CHAR(18)
PAGE_RANGE         CHAR(1)       NOT NULL WITH DEFAULT
JOIN_TYPE          CHAR(1)       NOT NULL WITH DEFAULT
GROUP_MEMBER       CHAR(8)       NOT NULL WITH DEFAULT
IBM_SERVICE_DATA   VARCHAR(254)  NOT NULL WITH DEFAULT
  ------------------43 column format----------------
WHEN_OPTIMIZE      CHAR(1)       NOT NULL WITH DEFAULT
QBLOCK_TYPE        CHAR(6)       NOT NULL WITH DEFAULT
BIND_TIME          TIMESTAMP     NOT NULL WITH DEFAULT
  ------------------46 column format----------------
OPTHINT            CHAR(8)       NOT NULL WITH DEFAULT
HINT_USED          CHAR(8)       NOT NULL WITH DEFAULT
PRIMARY_ACCESSTYPE CHAR(1)       NOT NULL WITH DEFAULT
   ------------------49 column format----------------
PARENT_QBLOCKNO    SMALLINT      NOT NULL WITH DEFAULT
TABLE_TYPE         CHAR(1)
   ------------------51 column format----------------
```

Figure 102. Formats of PLAN_TABLE prior to the 58-column format

Table 165 on page 932 shows the descriptions of the columns in PLAN_TABLE.

Table 165. Descriptions of columns in PLAN_TABLE

Column Name	Description
QUERYNO	A number intended to identify the statement being explained. For a row produced by an EXPLAIN statement, specify the number in the QUERYNO clause. For a row produced by non-EXPLAIN statements, specify the number using the QUERYNO clause, which is an optional part of the SELECT, INSERT, UPDATE and DELETE statement syntax. Otherwise, DB2 assigns a number based on the line number of the SQL statement in the source program. When the values of QUERYNO are based on the statement number in the source program, values greater than 32767 are reported as 0. However, in a very long program, the value is not guaranteed to be unique. If QUERYNO is not unique, the value of TIMESTAMP is unique.
QBLOCKNO	A number that identifies each query block within a query. The value of the numbers are not in any particular order, nor are they necessarily consecutive.
APPLNAME	The name of the application plan for the row. Applies only to embedded EXPLAIN statements executed from a plan or to statements explained when binding a plan. Blank if not applicable.
PROGNAME	The name of the program or package containing the statement being explained. Applies only to embedded EXPLAIN statements and to statements explained as the result of binding a plan or package. Blank if not applicable.
PLANNO	The number of the step in which the query indicated in QBLOCKNO was processed. This column indicates the order in which the steps were executed.
METHOD	A number (0, 1, 2, 3, or 4) that indicates the join method used for the step: **0** First table accessed, continuation of previous table accessed, or not used. **1** *Nested loop* join. For each row of the present composite table, matching rows of a new table are found and joined. **2** *Merge scan* join. The present composite table and the new table are scanned in the order of the join columns, and matching rows are joined. **3** Sorts needed by ORDER BY, GROUP BY, SELECT DISTINCT, UNION, a quantified predicate, or an IN predicate. This step does not access a new table. **4** *Hybrid* join. The current composite table is scanned in the order of the join-column rows of the new table. The new table is accessed using list prefetch.
CREATOR	The creator of the new table accessed in this step, blank if METHOD is 3.
TNAME	The name of a table, materialized query table, created or declared temporary table, materialized view, or materialized table expression. The value is blank if METHOD is 3. The column can also contain the name of a table in the form DSNWFQB(*qblockno*). DSNWFQB(*qblockno*) is used to represent the intermediate result of a UNION ALL or an outer join that is materialized. If a view is merged, the name of the view does not appear.
TABNO	Values are for IBM use only.

Table 165. Descriptions of columns in PLAN_TABLE (continued)

Column Name	Description
ACCESSTYPE	The method of accessing the new table: **I** By an index (identified in ACCESSCREATOR and ACCESSNAME) **I1** By a one-fetch index scan **M** By a multiple index scan (followed by MX, MI, or MU) **MI** By an intersection of multiple indexes **MU** By a union of multiple indexes **MX** By an index scan on the index named in ACCESSNAME **N** By an index scan when the matching predicate contains the IN keyword **R** By a table space scan **RW** By a work file scan of the result of a materialized user-defined table function **T** By a sparse index (star join work files) **V** By buffers for an INSERT statement within a SELECT **blank** Not applicable to the current row
MATCHCOLS	For ACCESSTYPE I, I1, N, or MX, the number of index keys used in an index scan; otherwise, 0.
ACCESSCREATOR	For ACCESSTYPE I, I1, N, or MX, the creator of the index; otherwise, blank.
ACCESSNAME	For ACCESSTYPE I, I1, N, or MX, the name of the index; otherwise, blank.
INDEXONLY	Whether access to an index alone is enough to carry out the step, or whether data too must be accessed. Y=Yes; N=No. For exceptions, see "Is the query satisfied using only the index? (INDEXONLY=Y)" on page 942.
SORTN_UNIQ	Whether the new table is sorted to remove duplicate rows. Y=Yes; N=No.
SORTN_JOIN	Whether the new table is sorted for join method 2 or 4. Y=Yes; N=No.
SORTN_ORDERBY	Whether the new table is sorted for ORDER BY. Y=Yes; N=No.
SORTN_GROUPBY	Whether the new table is sorted for GROUP BY. Y=Yes; N=No.
SORTC_UNIQ	Whether the composite table is sorted to remove duplicate rows. Y=Yes; N=No.
SORTC_JOIN	Whether the composite table is sorted for join method 1, 2 or 4. Y=Yes; N=No.
SORTC_ORDERBY	Whether the composite table is sorted for an ORDER BY clause or a quantified predicate. Y=Yes; N=No.
SORTC_GROUPBY	Whether the composite table is sorted for a GROUP BY clause. Y=Yes; N=No.
TSLOCKMODE	An indication of the mode of lock to be acquired on either the new table, or its table space or table space partitions. If the isolation can be determined at bind time, the values are: **IS** Intent share lock **IX** Intent exclusive lock **S** Share lock **U** Update lock **X** Exclusive lock **SIX** Share with intent exclusive lock **N** UR isolation; no lock If the isolation cannot be determined at bind time, then the lock mode determined by the isolation at run time is shown by the following values. **NS** For UR isolation, no lock; for CS, RS, or RR, an S lock. **NIS** For UR isolation, no lock; for CS, RS, or RR, an IS lock. **NSS** For UR isolation, no lock; for CS or RS, an IS lock; for RR, an S lock. **SS** For UR, CS, or RS isolation, an IS lock; for RR, an S lock. The data in this column is right justified. For example, IX appears as a blank followed by I followed by X. If the column contains a blank, then no lock is acquired.
TIMESTAMP	Usually, the time at which the row is processed, to the last .01 second. If necessary, DB2 adds .01 second to the value to ensure that rows for two successive queries have different values.

Table 165. Descriptions of columns in PLAN_TABLE (continued)

Column Name	Description
REMARKS	A field into which you can insert any character string of 762 or fewer characters.
PREFETCH	Whether data pages are to be read in advance by prefetch: **S** Pure sequential prefetch **L** Prefetch through a page list **D** Possible candidate for dynamic prefetch **blank** Unknown or no prefetch
COLUMN_FN_EVAL	When an SQL aggregate function is evaluated. R = while the data is being read from the table or index; S = while performing a sort to satisfy a GROUP BY clause; blank = after data retrieval and after any sorts.
MIXOPSEQ	The sequence number of a step in a multiple index operation. **1, 2, ... n** For the steps of the multiple index procedure (ACCESSTYPE is MX, MI, or MU.) **0** For any other rows (ACCESSTYPE is I, I1, M, N, R, or blank.)
VERSION	The version identifier for the package. Applies only to an embedded EXPLAIN statement executed from a package or to a statement that is explained when binding a package. Blank if not applicable.
COLLID	The collection ID for the package. Applies only to an embedded EXPLAIN statement that is executed from a package or to a statement that is explained when binding a package. Blank if not applicable. The value DSNDYNAMICSQLCACHE indicates that the row is for a cached statement.
Note: The following nine columns, from ACCESS_DEGREE through CORRELATION_NAME, contain the null value if the plan or package was bound using a plan table with fewer than 43 columns. Otherwise, each of them can contain null if the method it refers to does not apply.	
ACCESS_DEGREE	The number of parallel tasks or operations activated by a query. This value is determined at bind time; the actual number of parallel operations used at execution time could be different. This column contains 0 if there is a host variable.
ACCESS_PGROUP_ID	The identifier of the parallel group for accessing the new table. A parallel group is a set of consecutive operations, executed in parallel, that have the same number of parallel tasks. This value is determined at bind time; it could change at execution time.
JOIN_DEGREE	The number of parallel operations or tasks used in joining the composite table with the new table. This value is determined at bind time and can be 0 if there is a host variable. The actual number of parallel operations or tasks used at execution time could be different.
JOIN_PGROUP_ID	The identifier of the parallel group for joining the composite table with the new table. This value is determined at bind time; it could change at execution time.
SORTC_PGROUP_ID	The parallel group identifier for the parallel sort of the composite table.
SORTN_PGROUP_ID	The parallel group identifier for the parallel sort of the new table.
PARALLELISM_MODE	The kind of parallelism, if any, that is used at bind time: **I** Query I/O parallelism **C** Query CP parallelism **X** Sysplex query parallelism
MERGE_JOIN_COLS	The number of columns that are joined during a merge scan join (Method=2).
CORRELATION_NAME	The correlation name of a table or view that is specified in the statement. If there is no correlation name, then the column is null.
PAGE_RANGE	Whether the table qualifies for page range screening, so that plans scan only the partitions that are needed. Y = Yes; blank = No.

Table 165. Descriptions of columns in PLAN_TABLE (continued)

Column Name	Description
JOIN_TYPE	The type of join: **F** FULL OUTER JOIN **L** LEFT OUTER JOIN **S** STAR JOIN **blank** INNER JOIN or no join RIGHT OUTER JOIN converts to a LEFT OUTER JOIN when you use it, so that JOIN_TYPE contains L.
GROUP_MEMBER	The member name of the DB2 that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.
IBM_SERVICE_DATA	Values are for IBM use only.
WHEN_OPTIMIZE	When the access path was determined: **blank** At bind time, using a default filter factor for any host variables, parameter markers, or special registers. **B** At bind time, using a default filter factor for any host variables, parameter markers, or special registers; however, the statement is reoptimized at run time using input variable values for input host variables, parameter markers, or special registers. The bind option REOPT(ALWAYS) or REOPT(ONCE) must be specified for reoptimization to occur. **R** At run time, using input variables for any host variables, parameter markers, or special registers. The bind option REOPT(ALWAYS) or REOPT(ONCE) must be specified for this to occur.
QBLOCK_TYPE	For each query block, an indication of the type of SQL operation performed. For the outermost query, this column identifies the statement type. Possible values: **SELECT** SELECT **INSERT** INSERT **UPDATE** UPDATE **DELETE** DELETE **SELUPD** SELECT with FOR UPDATE OF **DELCUR** DELETE WHERE CURRENT OF CURSOR **UPDCUR** UPDATE WHERE CURRENT OF CURSOR **CORSUB** Correlated subselect or fullselect **NCOSUB** Noncorrelated subselect or fullselect **TABLEX** Table expression **TRIGGR** WHEN clause on CREATE TRIGGER **UNION** UNION **UNIONA** UNION ALL
BIND_TIME	For static SQL statements, the time at which the plan or package for this statement or query block was bound. For cached dynamic SQL statements, the time at which the which the statement entered the cache. For static and cached dynamic SQL statements, this is a full-precision timestamp value. For non-cached dynamic SQL statements, this is the value contained in the TIMESTAMP column of PLAN_TABLE appended by 4 zeroes.
OPTHINT	A string that you use to identify this row as an optimization hint for DB2. DB2 uses this row as input when choosing an access path.
HINT_USED	If DB2 used one of your optimization hints, it puts the identifier for that hint (the value in OPTHINT) in this column.

Table 165. Descriptions of columns in PLAN_TABLE (continued)

Column Name	Description
PRIMARY_ACCESSTYPE	Indicates whether direct row access will be attempted first: **D** DB2 will try to use direct row access. If DB2 cannot use direct row access at run time, it uses the access path described in the ACCESSTYPE column of PLAN_TABLE. See "Is direct row access possible? (PRIMARY_ACCESSTYPE = D)" on page 942 for more information. **blank** DB2 will not try to use direct row access.
PARENT_QBLOCKNO	A number that indicates the QBLOCKNO of the parent query block.
TABLE_TYPE	The type of new table: **B** Buffers for an INSERT statement within a SELECT **C** Common table expression **F** Table function **M** Materialized query table **Q** Temporary intermediate result table (not materialized). For the name of a view or nested table expression, a value of Q indicates that the materializaition was virtual and not actual. Materialization can be virtual when the view or nested table expression definition contains a UNION ALL that is not distributed. **R** Recursive common table expression **T** Table **W** Work file The value of the column is null if the query uses GROUP BY, ORDER BY, or DISTINCT, which requires an implicit sort.
TABLE_ENCODE	The encoding scheme of the table. If the table has a single CCSID set, possible values are: **A** ASCII **E** EBCDIC **U** Unicode M is the value of the column when the table contains muliple CCSID set, the value of the column is M.
TABLE_SCCSID	The SBCS CCSID value of the table. If column TABLE_ENCODE is M, the value is 0.
TABLE_MCCSID	The mixed CCSID value of the table. If column TABLE_ENCODE is M, the value is 0
TABLE_DCCSID	The DBCS CCSID value of the table. If column TABLE_ENCODE is M, the value is 0.
ROUTINE_ID	Values are for IBM use only.
CTEREF	If the referenced table is a common table expression, the value is the top-level query block number.
STMTTOKEN	User-specified statement token.

Populating and maintaining a plan table

For the two distinct ways to populate a plan table, see:

- "Executing the SQL statement EXPLAIN" on page 937
- "Binding with the option EXPLAIN(YES)" on page 937

When you populate the plan table through EXPLAIN, any INSERT triggers on the table are not activated. If you insert rows yourself, then those triggers are activated.

For a variation on executing the EXPLAIN statement, see "Executing EXPLAIN under DB2 QMF" on page 937.

For tips on maintaining a growing plan table, see "Maintaining a plan table" on page 939.

Executing the SQL statement EXPLAIN

You can populate a PLAN_TABLE by executing the SQL statement EXPLAIN. In the statement, specify a single explainable SQL statement in the FOR clause.

If an alias is defined on a PLAN_TABLE that was created with a different authorization ID, and you have the appropriate SELECT and INSERT privileges, you can populate that PLAN_TABLE even if you do not own it.

You can execute EXPLAIN either statically from an application program, or dynamically, using QMF or SPUFI. For instructions and for details of the authorization that you need on PLAN_TABLE, see *DB2 SQL Reference*.

Binding with the option EXPLAIN(YES)

You can populate a plan table when you bind or rebind a plan or package. Specify the option EXPLAIN(YES). EXPLAIN obtains information about the access paths for all explainable SQL statements in a package or the DBRMs of a plan. The information appears in table *package_owner*.PLAN_TABLE or *plan_owner*.PLAN_TABLE. For dynamically prepared SQL, the qualifier of PLAN_TABLE is the current SQLID.

If the plan owner or the package owner has an alias on a PLAN_TABLE that was created by another owner, *other_owner*.PLAN_TABLE is populated instead of *package_owner*.PLAN_TABLE or *plan_owner*.PLAN_TABLE.

Performance considerations: EXPLAIN as a bind option should not be a performance concern. The same processing for access path selection is performed, regardless of whether you use EXPLAIN(YES) or EXPLAIN (NO). With EXPLAIN(YES), there is only a small amount of overhead processing to put the results in a plan table.

If a plan or package that was previously bound with EXPLAIN(YES) is automatically rebound, the value of field EXPLAIN PROCESSING on installation panel DSNTIPO determines whether EXPLAIN is run again during the automatic rebind. Again, there is a small amount of overhead for inserting the results into a plan table.

EXPLAIN for remote binds: A remote requester that accesses DB2 can specify EXPLAIN(YES) when binding a package at the DB2 server. The information appears in a plan table at the server, not at the requester. If the requester does not support the propagation of the option EXPLAIN(YES), rebind the package at the requester with that option to obtain access path information. You cannot get information about access paths for SQL statements that use private protocol.

Executing EXPLAIN under DB2 QMF

You can use DB2 QMF to display the results of EXPLAIN to the terminal. You can create your own form to display the output or use the default form for DB2 QMF.

Use parameter markers for host variables: If you have host variables in a predicate for an original query in a static application and if you are using DB2 QMF or SPUFI to execute EXPLAIN for the query, in most cases, use parameter markers where you use host variables in the original query. If you use a literal value instead, you might see different access paths for your static and dynamic queries. For instance, compare the following queries:

```
                        QMF Query Using Parameter   QMF Query Using
Original Static SQL       Marker                       Literal

DECLARE C1               EXPLAIN PLAN SET            EXPLAIN PLAN SET
CURSOR FOR               QUERYNO=1 FOR               QUERYNO=1 FOR
SELECT *                 SELECT *                    SELECT *
 FROM T1                  FROM T1                    FROM T1
 WHERE C1 > HOST VAR.     WHERE C1 > ?               WHERE C1 > 10
```

Using the literal '10' would likely produce a different filter factor and maybe a different access path from the original static SQL. (A filter factor is the proportion of rows that remain after a predicate has "filtered out" the rows that do not satisfy it. For more information about filter factors, see "Predicate filter factors" on page 760.) The parameter marker behaves just like a host variable, in that the predicate is assigned a default filter factor.

When to use a literal: If you know that the static plan or package was bound with REOPT(ALWAYS) and you have some idea of what is returned in the host variable, it can be more accurate to include the literal in the DB2 QMF EXPLAIN. REOPT(ALWAYS) means that DB2 will replace the value of the host variable with the true value at run time and then determine the access path. For more information about REOPT(ALWAYS) see "Changing the access path at run time" on page 774.

Expect these differences: Even when using parameter markers, you could see different access paths for static and dynamic queries. DB2 assumes that the value that replaces a parameter marker has the same length and precision as the column it is compared to. That assumption determines whether the predicate is stage 1 indexable or stage 2, which is always nonindexable.

If the column definition and the host variable definition are both strings, the predicate becomes stage 1 but not indexable when any of the following conditions are true:

- The column definition is CHAR or VARCHAR, and the host variable definition is GRAPHIC or VARGRAPHIC.
- The column definition is GRAPHIC or VARGRAPHIC, the host variable definition is CHAR or VARCHAR, and the length of the column definition is less than the length of the host variable definition.
- Both the column definition and the host variable definition are CHAR or VARCHAR, the length of the column definition is less than the length of the host variable definition, and the comparison operator is any operator other than "=".
- Both the column definition and the host variable definition are GRAPHIC or VARGRAPHIC, the length of the column definition is less than the length of the host variable definition, and the comparison operator is any operator other than "=".

The predicate becomes stage 2 when any of the following conditions are true:

- The column definition is DECIMAL(p,s), where p>15, and the host variable definition is REAL or FLOAT.
- The column definition is CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC, and the host variable definition is DATE, TIME, or TIMESTAMP.

Maintaining a plan table

DB2 adds rows to PLAN_TABLE as you choose; it does not automatically delete rows. To clear the table of obsolete rows, use DELETE, just as you would for deleting rows from any table. You can also use DROP TABLE to drop a plan table completely.

Reordering rows from a plan table

Several processes can insert rows into the same plan table. To understand access paths, you must retrieve the rows for a particular query in an appropriate order.

Retrieving rows for a plan

The rows for a particular plan are identified by the value of APPLNAME. The following query to a plan table returns the rows for all the explainable statements in a plan in their logical order:

```
SELECT * FROM JOE.PLAN_TABLE
  WHERE APPLNAME = 'APPL1'
  ORDER BY TIMESTAMP, QUERYNO, QBLOCKNO, PLANNO, MIXOPSEQ;
```

The result of the ORDER BY clause shows whether there are:

- Multiple QBLOCKNOs within a QUERYNO
- Multiple PLANNOs within a QBLOCKNO
- Multiple MIXOPSEQs within a PLANNO

All rows with the same non-zero value for QBLOCKNO and the same value for QUERYNO relate to a step within the query. QBLOCKNOs are not necessarily executed in the order shown in PLAN_TABLE. But within a QBLOCKNO, the PLANNO column gives the substeps in the order they execute.

For each substep, the TNAME column identifies the table accessed. Sorts can be shown as part of a table access or as a separate step.

What if QUERYNO=0? For entries that contain QUERYNO=0, use the timestamp, which is guaranteed to be unique, to distinguish individual statements.

Retrieving rows for a package

The rows for a particular package are identified by the values of PROGNAME, COLLID, and VERSION. Those columns correspond to the following four-part naming convention for packages:

```
LOCATION.COLLECTION.PACKAGE_ID.VERSION
```

COLLID gives the COLLECTION name, and PROGNAME gives the PACKAGE_ID. The following query to a plan table return the rows for all the explainable statements in a package in their logical order:

```
SELECT * FROM JOE.PLAN_TABLE
  WHERE PROGNAME = 'PACK1' AND COLLID = 'COLL1' AND VERSION =  'PROD1'
  ORDER BY QUERYNO, QBLOCKNO, PLANNO, MIXOPSEQ;
```

Asking questions about data access

When you examine your EXPLAIN results, try to answer the following questions:

- "Is access through an index? (ACCESSTYPE is I, I1, N or MX)" on page 940
- "Is access through more than one index? (ACCESSTYPE=M)" on page 940
- "How many columns of the index are used in matching? (MATCHCOLS=n)" on page 941
- "Is the query satisfied using only the index? (INDEXONLY=Y)" on page 942

- "Is direct row access possible? (PRIMARY_ACCESSTYPE = D)" on page 942
- "Is a view or nested table expression materialized?" on page 944
- "Was a scan limited to certain partitions? (PAGE_RANGE=Y)" on page 944
- "What kind of prefetching is expected? (PREFETCH = L, S, D, or blank)" on page 945
- "Is data accessed or processed in parallel? (PARALLELISM_MODE is I, C, or X)" on page 945
- "Are sorts performed?" on page 945
- "Is a subquery transformed into a join?" on page 946
- "When are aggregate functions evaluated? (COLUMN_FN_EVAL)" on page 946
- "How many index screening columns are used?" on page 946
- "Is a complex trigger WHEN clause used? (QBLOCKTYPE=TRIGGR)" on page 946

As explained in this section, they can be answered in terms of values in columns of a plan table.

Is access through an index? (ACCESSTYPE is I, I1, N or MX)

If the column ACCESSTYPE in the plan table has one of those values, DB2 uses an index to access the table that is named in column TNAME. The columns ACCESSCREATOR and ACCESSNAME identify the index. For a description of methods of using indexes, see "Index access paths" on page 950.

Is access through more than one index? (ACCESSTYPE=M)

The value M indicates that DB2 uses a set of indexes to access a single table. A set of rows in the plan table contain information about the multiple index access. The rows are numbered in column MIXOPSEQ in the order of execution of steps in the multiple index access. (If you retrieve the rows in order by MIXOPSEQ, the result is similar to postfix arithmetic notation.)

Both of the following examples have these indexes: IX1 on T(C1) and IX2 on T(C2).

Example: Suppose that you issue the following SELECT statement:

```
SELECT * FROM T
  WHERE C1 = 1 AND C2 = 1;
```

DB2 processes the query by performing the following steps:

1. DB2 retrieves all the qualifying record identifiers (RIDs) where C1=1, by using index IX1.
2. DB2 retrieves all the qualifying RIDs where C2=1, by using index IX2. The intersection of these lists is the final set of RIDs.
3. DB2 accesses the data pages that are needed to retrieve the qualified rows by using the final RID list.

The plan table for this example is shown in Table 166.

Table 166. PLAN_TABLE output for example with intersection (AND) operator

TNAME	ACCESS-TYPE	MATCH-COLS	ACCESS-NAME	INDEX-ONLY	PREFETCH	MIXOP-SEQ
T	M	0		N	L	0
T	MX	1	IX1	Y		1

Table 166. PLAN_TABLE output for example with intersection (AND) operator (continued)

TNAME	ACCESS-TYPE	MATCH-COLS	ACCESS-NAME	INDEX-ONLY	PREFETCH	MIXOP-SEQ
T	MX	1	IX2	Y		2
T	MI	0		N		3

Example: Suppose that you issue the following SELECT statement:

```
SELECT * FROM T
  WHERE C1 BETWEEN 100 AND 199 OR
        C1 BETWEEN 500 AND 599;
```

In this case, the same index can be used more than once in a multiple index access because more than one predicate could be matching. DB2 processes the query by performing the following steps:

1. DB2 retrieves all RIDs where C1 is between 100 and 199, using index IX1.
2. DB2 retrieves all RIDs where C1 is between 500 and 599, again using IX1. The union of those lists is the final set of RIDs.
3. DB2 retrieves the qualified rows by using the final RID list.

The plan table for this example is shown in Table 167.

Table 167. PLAN_TABLE output for example with union (OR) operator

TNAME	ACCESS-TYPE	MATCH-COLS	ACCESS-NAME	INDEX-ONLY	PREFETCH	MIXOP-SEQ
T	M	0		N	L	0
T	MX	1	IX1	Y		1
T	MX	1	IX1	Y		2
T	MU	0		N		3

How many columns of the index are used in matching? (MATCHCOLS=n)

If MATCHCOLS is 0, the access method is called a *nonmatching index scan*. All the index keys and their RIDs are read.

If MATCHCOLS is greater than 0, the access method is called a *matching index scan*: the query uses predicates that match the index columns.

In general, the matching predicates on the leading index columns are equal or IN predicates. The predicate that matches the final index column can be an equal, IN, NOT NULL, or range predicate (<, <=, >, >=, LIKE, or BETWEEN).

The following example illustrates matching predicates:

```
SELECT * FROM EMP
  WHERE JOBCODE = '5' AND SALARY > 60000 AND LOCATION = 'CA';

INDEX XEMP5 on (JOBCODE, LOCATION, SALARY, AGE);
```

The index XEMP5 is the chosen access path for this query, with MATCHCOLS = 3. Two equal predicates are on the first two columns and a range predicate is on the third column. Though the index has four columns in the index, only three of them can be considered matching columns.

Is the query satisfied using only the index? (INDEXONLY=Y)

In this case, the method is called *index-only access*. For a SELECT operation, all the columns needed for the query can be found in the index and DB2 does not access the table. For an UPDATE or DELETE operation, only the index is required to read the selected row.

Index-only access to data is not possible for any step that uses list prefetch, which is described under "What kind of prefetching is expected? (PREFETCH = L, S, D, or blank)" on page 945. Index-only access is not possible for padded indexes when varying-length data is returned or a VARCHAR column has a LIKE predicate, unless the VARCHAR FROM INDEX field of installation panel DSNTIP4 is set to YES and plan or packages have been rebound to pick up the change. See Part 2 of *DB2 Installation Guide* for more information. Index-only access is always possible for nonpadded indexes.

If access is by more than one index, INDEXONLY is Y for a step with access type MX, because the data pages are not actually accessed until all the steps for intersection (MI) or union (MU) take place.

When an SQL application uses index-only access for a ROWID column, the application claims the table space or table space partition. As a result, contention may occur between the SQL application and a utility that drains the table space or partition. Index-only access to a table for a ROWID column is not possible if the associated table space or partition is in an incompatible restrictive state. For example, an SQL application can make a read claim on the table space only if the restrictive state allows readers.

Is direct row access possible? (PRIMARY_ACCESSTYPE = D)

If an application selects a row from a table that contains a ROWID column, the row ID value implicitly contains the location of the row. If you use that row ID value in the search condition of subsequent SELECTs, DELETEs, or UPDATEs, DB2 might be able to navigate directly to the row. This access method is called *direct row access*.

Direct row access is very fast, because DB2 does not need to use the index or a table space scan to find the row. Direct row access can be used on any table that has a ROWID column.

To use direct row access, you first select the values of a row into host variables. The value that is selected from the ROWID column contains the location of that row. Later, when you perform queries that access that row, you include the row ID value in the search condition. If DB2 determines that it can use direct row access, it uses the row ID value to navigate directly to the row.

Which predicates qualify for direct row access?

For a query to qualify for direct row access, the search condition must be a Boolean term *stage 1* predicate that fits one of these descriptions:

1. A simple Boolean term predicate of the form COL=*noncolumn expression*, where COL has the ROWID data type and *noncolumn expression* contains a row ID
2. A simple Boolean term predicate of the form COL IN *list*, where COL has the ROWID data type and the values in *list* are row IDs, and an index is defined on COL
3. A compound Boolean term that combines several simple predicates using the AND operator, and one of the simple predicates fits description 1 or 2

However, just because a query qualifies for direct row access does not mean that access path is always chosen. If DB2 determines that another access path is better, direct row access is not chosen.

Examples: In the following predicate example, ID is a ROWID column in table T1. A unique index exists on that ID column. The host variables are of the ROWID type.

```
WHERE ID IN (:hv_rowid1,:hv_rowid2,:hv_rowid3)
```

The following predicate also qualifies for direct row access:

```
WHERE ID = ROWID(X'F0DFD230E3C0D80D81C201AA0A280100000000000203')
```

Searching for propagated rows: If rows are propagated from one table to another, do not expect to use the same row ID value from the source table to search for the same row in the target table, or vice versa. This does not work when direct row access is the access path chosen.

Example: Assume that the host variable in the following statement contains a row ID from SOURCE:

```
SELECT * FROM TARGET
  WHERE ID = :hv_rowid
```

Because the row ID location is not the same as in the source table, DB2 will probably not find that row. Search on another column to retrieve the row you want.

Reverting to ACCESSTYPE

Although DB2 might plan to use direct row access, circumstances can cause DB2 to not use direct row access at run time. DB2 remembers the location of the row as of the time it is accessed. However, that row can change locations (such as after a REORG) between the first and second time it is accessed, which means that DB2 cannot use direct row access to find the row on the second access attempt. Instead of using direct row access, DB2 uses the access path that is shown in the ACCESSTYPE column of PLAN_TABLE.

If the predicate you are using to do direct row access is not indexable and if DB2 is unable to use direct row access, then DB2 uses a table space scan to find the row. This can have a profound impact on the performance of applications that rely on direct row access. Write your applications to handle the possibility that direct row access might not be used. Some options are to:

- Ensure that your application does not try to remember ROWID columns across reorganizations of the table space.

  When your application commits, it releases its claim on the table space; it is possible that a REORG can run and move the row, which disables direct row access. Plan your commit processing accordingly; use the returned row ID value before committing, or re-select the row ID value after a commit is issued.

  If you are storing ROWID columns from another table, update those values after the table with the ROWID column is reorganized.
- Create an index on the ROWID column, so that DB2 can use the index if direct row access is disabled.
- Supplement the ROWID column predicate with another predicate that enables DB2 to use an existing index on the table. For example, after reading a row, an application might perform the following update:

```
EXEC SQL UPDATE EMP
SET SALARY = :hv_salary + 1200
  WHERE EMP_ROWID = :hv_emp_rowid
  AND EMPNO = :hv_empno;
```

If an index exists on EMPNO, DB2 can use index access if direct access fails. The additional predicate ensures DB2 does not revert to a table space scan.

Using direct row access and other access methods

Parallelism: Direct row access and parallelism are mutually exclusive. If a query qualifies for both direct row access and parallelism, direct row access is used. If direct row access fails, DB2 does not revert to parallelism; instead it reverts to the backup access type (as designated by column ACCESSTYPE in the PLAN_TABLE). This might result in a table space scan. To avoid a table space scan in case direct row access fails, add an indexed column to the predicate.

RID list processing: Direct row access and RID list processing are mutually exclusive. If a query qualifies for both direct row access and RID list processing, direct row access is used. If direct row access fails, DB2 does not revert to RID list processing; instead it reverts to the backup access type.

Is a view or nested table expression materialized?

When the column TNAME names a view or nested table expression and column TABLE_TYPE contains a W, it indicates that the view or nested table expression is materialized. *Materialization* means that the data rows that are selected by the view or nested table expression are put into a work file that is to be processed like a table. (For a more detailed description of materialization, see "Processing for views and nested table expressions" on page 975.)

Was a scan limited to certain partitions? (PAGE_RANGE=Y)

DB2 can limit a scan of data in a partitioned table space to one or more partitions. The method is called a *limited partition scan*. The query must provide a predicate on the first key column of the partitioning index. DB2 can use all leading columns of the partitioning key. The rules for using a partitioning column to limit partitions are the same as the rules for determining a matching column on an index. If a partitioning column is a candidate to be a matching column, that column can limit partitions.

A limited partition scan can be combined with other access methods. For example, consider the following query:

```
SELECT .. FROM T
   WHERE (C1 BETWEEN '2002' AND '3280'
   OR C1 BETWEEN '6000' AND '8000')
   AND C2 = '6';
```

Assume that table T has a partitioned index on column C1 and that values of C1 between 2002 and 3280 all appear in partitions 3 and 4 and the values between 6000 and 8000 appear in partitions 8 and 9. Assume also that T has another index on column C2. DB2 could choose any of these access methods:

- A matching index scan on column C1. The scan reads index values and data only from partitions 3, 4, 8, and 9. (PAGE_RANGE=N)
- A matching index scan on column C2. (DB2 might choose that if few rows have C2=6.) The matching index scan reads all RIDs for C2=6 from the index on C2 and corresponding data pages from partitions 3, 4, 8, and 9. (PAGE_RANGE=Y)

- A table space scan on T. DB2 avoids reading data pages from any partitions except 3, 4, 8 and 9. (PAGE_RANGE=Y)

What kind of prefetching is expected? (PREFETCH = L, S, D, or blank)

Prefetching is a method of determining in advance that a set of data pages is about to be used and then reading the entire set into a buffer with a single asynchronous I/O operation. If the value of PREFETCH is:

- S, the method is called *sequential prefetch*. The data pages that are read in advance are sequential. A table space scan always uses sequential prefetch. An index scan might not use it. For a more complete description, see "Sequential prefetch (PREFETCH=S)" on page 969.
- L, the method is called *list prefetch*. One or more indexes are used to select the RIDs for a list of data pages to be read in advance; the pages need not be sequential. Usually, the RIDs are sorted. The exception is the case of a hybrid join (described under "Hybrid join (METHOD=4)" on page 961) when the value of column SORTN_JOIN is N. For a more complete description, see "List prefetch (PREFETCH=L)" on page 970.
- D, the method is called *dynamic prefetch*. DB2 expects that the pages to be accessed will be sufficiently nonsequential to invoke dynamic prefetch.
- Blank, prefetching is not expected. However, depending on the pattern of the page access, data can be prefetched at execution time through a process called *sequential detection*. For a description of that process, see "Sequential detection at execution time" on page 972.

Is data accessed or processed in parallel? (PARALLELISM_MODE is I, C, or X)

Parallel processing applies only to read-only queries.

If mode is:	DB2 plans to use:
I	Parallel I/O operations
C	Parallel CP operations
X	Sysplex query parallelism

Non-null values in columns ACCESS_DEGREE and JOIN_DEGREE indicate to what degree DB2 plans to use parallel operations. At execution time, however, DB2 might not actually use parallelism, or it might use fewer operations in parallel than were originally planned. For a more complete description, see Chapter 35, "Parallel operations and query performance," on page 987. For more information about Sysplex query parallelism, see Chapter 6 of *DB2 Data Sharing: Planning and Administration*.

Are sorts performed?

SORTN_JOIN and SORTC_JOIN: SORTN_JOIN indicates that the new table of a join is sorted before the join. (For hybrid join, this is a sort of the RID list.) When SORTN_JOIN and SORTC_JOIN are both 'Y', two sorts are performed for the join. The sorts for joins are indicated on the same row as the new table access.

METHOD 3 sorts: These are used for ORDER BY, GROUP BY, SELECT DISTINCT, UNION, or a quantified predicate. A quantified predicate is 'col = ANY (fullselect)' or 'col = SOME (fullselect)'. They are indicated on a separate row. A single row of the plan table can indicate two sorts of a composite table, but only one sort is actually done.

SORTC_UNIQ and SORTC_ORDERBY: SORTC_UNIQ indicates a sort to remove duplicates, as might be needed by a SELECT statement with DISTINCT or UNION. SORTC_ORDERBY usually indicates a sort for an ORDER BY clause. But SORTC_UNIQ and SORTC_ORDERBY also indicate when the results of a noncorrelated subquery are sorted, both to remove duplicates and to order the results. One sort does both the removal and the ordering.

Is a subquery transformed into a join?

For better access paths, DB2 sometimes transforms subqueries into joins, as described in "Conditions for DB2 to transform a subquery into a join" on page 782. A plan table shows that a subquery is transformed into a join by the value in column QBLOCKNO.

- If the subquery is not transformed into a join, that means it is executed in a separate operation, and its value of QBLOCKNO is greater than the value for the outer query.
- If the subquery is transformed into a join, it and the outer query have the same value of QBLOCKNO. A join is also indicated by a value of 1, 2, or 4 in column METHOD.

When are aggregate functions evaluated? (COLUMN_FN_EVAL)

When the aggregate functions are evaluated is based on the access path chosen for the SQL statement.

- If the ACCESSTYPE column is I1, then a MAX or MIN function can be evaluated by one access of the index named in ACCESSNAME.
- For other values of ACCESSTYPE, the COLUMN_FN_EVAL column tells when DB2 is evaluating the aggregate functions.

Value	Functions are evaluated ...
S	While performing a sort to satisfy a GROUP BY clause
R	While the data is being read from the table or index
blank	After data retrieval and after any sorts

Generally, values of R and S are considered better for performance than a blank.

Use variance and standard deviation with care: The VARIANCE and STDDEV functions are always evaluated late (that is, COLUMN_FN_EVAL is blank). This causes other functions in the same query block to be evaluated late as well. For example, in the following query, the sum function is evaluated later than it would be if the variance function was not present:

```
SELECT SUM(C1), VARIANCE(C1) FROM T1;
```

How many index screening columns are used?

The plan table does not provide the answer to this question. However, you can use Visual Explain to determine how many index screening columns are used. For information about Visual Explain, see DB2 Visual Explain online help.

Is a complex trigger WHEN clause used? (QBLOCKTYPE=TRIGGR)

The plan table does not report simple trigger WHEN clauses, such as WHEN (N.C1 < 5). However, the plan table does report complex trigger WHEN clauses,

which are clauses that involve other base tables and transition tables. The QBLOCK_TYPE column of the top level query block shows TRIGGR to indicate a complex trigger WHEN clause.

Example: Consider the following trigger:

```
CREATE TRIGGER REORDER
       AFTER UPDATE OF ON_HAND, MAX_STOCKED ON PARTS
       REFERENCING NEW TABLE AS NT OLD AS O
       FOR EACH STATEMENT MODE DB2SQL
       WHEN (O.ON_HAND < (SELECT MAX(ON_HAND) FROM NT))
          BEGIN ATOMIC
              INSERT INTO ORDER_LOG VALUES (O.PARTNO, O.ON_HAND);
          END
```

Table 168 shows the corresponding plan table for the WHEN clause.

Table 168. Plan table for the WHEN clause

QBLOCKNO	PLANNO	TABLE	ACCESSTYPE	QBLOCK_TYPE	PARENT_QBLOCKNO
1	1			TRIGGR	0
2	1	NT	R	NCOSUB	1

Interpreting access to a single table

The following sections describe different access paths that values in a plan table can indicate, along with suggestions for supplying better access paths for DB2 to choose from:

- Table space scans (ACCESSTYPE=R PREFETCH=S)
- "Overview of index access" on page 948
- "Index access paths" on page 950
- "UPDATE using an index" on page 954

Table space scans (ACCESSTYPE=R PREFETCH=S)

Table space scan is most often used for one of the following reasons:

- Access is through a created temporary table. (Index access is not possible for created temporary tables.)
- A matching index scan is not possible because an index is not available, or no predicates match the index columns.
- A high percentage of the rows in the table is returned. In this case, an index is not really useful because most rows need to be read anyway.
- The indexes that have matching predicates have low cluster ratios and are therefore efficient only for small amounts of data.

Assume that table T has no index on C1. The following is an example that uses a table space scan:

```
SELECT * FROM T WHERE C1 = VALUE;
```

In this case, at least every row in T must be examined to determine whether the value of C1 matches the given value.

Table space scans of nonsegmented table spaces

DB2 reads and examines every page in the table space, regardless of which table the page belongs to. It might also read pages that have been left as free space and space not yet reclaimed after deleting data.

Table space scans of segmented table spaces

If the table space is segmented, DB2 first determines which segments need to be read. It then reads only the segments in the table space that contain rows of T. If the prefetch quantity, which is determined by the size of your buffer pool, is greater than the SEGSIZE and if the segments for T are not contiguous, DB2 might read unnecessary pages. Use a SEGSIZE value that is as large as possible, consistent with the size of the data. A large SEGSIZE value is best to maintain clustering of data rows. For very small tables, specify a SEGSIZE value that is equal to the number of pages required for the table.

Recommendation for SEGSIZE value: Table 169 summarizes the recommendations for SEGSIZE, depending on how large the table is.

Table 169. Recommendations for SEGSIZE

Number of pages	SEGSIZE recommendation
≤ 28	4 to 28
> 28 < 128 pages	32
≥ 128 pages	64

Table space scans of partitioned table spaces

Partitioned table spaces are nonsegmented. A table space scan on a partitioned table space is more efficient than on a nonpartitioned table space. DB2 takes advantage of the partitions by a limited partition scan, as described under "Was a scan limited to certain partitions? (PAGE_RANGE=Y)" on page 944.

Table space scans and sequential prefetch

Regardless of the type of table space, DB2 plans to use sequential prefetch for a table space scan. For a segmented table space, DB2 might not actually use sequential prefetch at execution time if it can determine that fewer than four data pages need to be accessed. For guidance on monitoring sequential prefetch, see "Sequential prefetch (PREFETCH=S)" on page 969.

If you do not want to use sequential prefetch for a particular query, consider adding to it the clause OPTIMIZE FOR 1 ROW.

Overview of index access

Indexes can provide efficient access to data. In fact, that is the only purpose of nonunique indexes. Unique indexes have the additional purpose of ensuring that key values are unique.

Using indexes to avoid sorts

As well as providing selective access to data, indexes can also order data, sometimes eliminating the need for sorting. Some sorts can be avoided if index keys are in the order needed by ORDER BY, GROUP BY, a join operation, or DISTINCT in a aggregate function. In other cases, as when list prefetch is used, the index does not provide useful ordering, and the selected data might have to be sorted.

When it is absolutely necessary to prevent a sort, consider creating an index on the column or columns necessary to provide that ordering. Consider also using the clause OPTIMIZE FOR 1 ROW to discourage DB2 from choosing a sort for the access path.

Consider the following query:

```
SELECT C1,C2,C3 FROM T
  WHERE C1 > 1
    ORDER BY C1 OPTIMIZE FOR 1 ROW;
```

An ascending index on C1 or an index on (C1,C2,C3) could eliminate a sort. For more information about OPTIMIZE FOR n ROWS, see "Minimizing overhead for retrieving few rows: OPTIMIZE FOR *n* ROWS" on page 790.

Backward index scan: In some cases, DB2 can use a backward index scan on a descending index to avoid a sort on ascending data. Similarly, an ascending index can be used to avoid a sort on descending data. For DB2 to use a backward index scan, the following conditions must be true:

- The index includes the columns in the ORDER BY clause in the same order that they appear in the ORDER BY clause.
- Each column in the sequence must have the opposite sequence (ASC or DESC) of the ORDER BY clause.

Example: Suppose that an index exists on the ACCT_STAT table. The index is defined by the following columns: ACCT_NUM, STATUS_DATE, STATUS_TIME. All of the columns in the index are in ascending order. Now, consider the following SELECT statements:

```
SELECT STATUS_DATE, STATUS
  FROM ACCT_STAT
  WHERE ACCT_NUM = :HV
  ORDER BY STATUS_DATE DESC, STATUS_TIME DESC;
```

```
SELECT STATUS_DATE, STATUS
  FROM ACCT_STAT
  WHERE ACCT_NUM = :HV
  ORDER BY STATUS_DATE ASC, STATUS_TIME ASC;
```

By using a backward index scan, DB2 can use the same index for both statements.

Not all sorts are inefficient. For example, if the index that provides ordering is not an efficient one and many rows qualify, it is possible that using another access path to retrieve and then sort the data could be more efficient than the inefficient, ordering index.

Indexes that are created to avoid sorts can sometimes be non-selective. If these indexes require data access and if the cluster ratio is poor, these indexes are unlikely to be chosen. Accessing many rows by using a poorly clustered index is often less efficient than accessing rows by using a table space scan and sort. Both table space scan and sort benefit from sequential access.

Costs of indexes

Before you begin creating indexes, consider carefully their costs:

- Indexes require storage space. Padded indexes require more space than nonpadded indexes for long index keys. For short index keys, nonpadded indexes can take more space.
- Each index requires an index space and a data set, or as many data sets as the number of data partitions if the index is partitioned, and operating system restrictions exist on the number of open data sets.
- Indexes must be changed to reflect every insert or delete operation on the base table. If an update operation updates a column that is in the index, then the index must also be changed. The time required by these operations increases accordingly.

- Indexes can be built automatically when loading data, but this takes time. They must be recovered or rebuilt if the underlying table space is recovered, which might also be time-consuming.

Recommendation: In reviewing the access paths described in "Index access paths," consider indexes as part of your database design, See Part 2, "Designing a database: advanced topics," on page 23 for details about database design in general. For a query with a performance problem, ask yourself:
- Would adding a column to an index allow the query to use index-only access?
- Do you need a new index?
- Is your choice of clustering index correct?

Index access paths

DB2 uses the following index access paths:
- "Matching index scan (MATCHCOLS>0)"
- "Index screening" on page 951
- "Nonmatching index scan (ACCESSTYPE=I and MATCHCOLS=0)" on page 951
- "IN-list index scan (ACCESSTYPE=N)" on page 951
- "Multiple index access (ACCESSTYPE is M, MX, MI, or MU)" on page 952
- "One-fetch access (ACCESSTYPE=I1)" on page 953
- "Index-only access (INDEXONLY=Y)" on page 954
- "Equal unique index (MATCHCOLS=number of index columns)" on page 954

Matching index scan (MATCHCOLS>0)

In a *matching index scan*, predicates are specified on either the leading or all of the index key columns. These predicates provide *filtering*; only specific index pages and data pages need to be accessed. If the degree of filtering is high, the matching index scan is efficient.

In the general case, the rules for determining the number of matching columns are simple, although there are a few exceptions.
- Look at the index columns from leading to trailing. For each index column, search for an indexable boolean term predicate on that column. (See "Properties of predicates" on page 749 for a definition of boolean term.) If such a predicate is found, then it can be used as a matching predicate.

  Column MATCHCOLS in a plan table shows how many of the index columns are matched by predicates.
- If no matching predicate is found for a column, the search for matching predicates stops.
- If a matching predicate is a range predicate, then there can be no more matching columns. For example, in the matching index scan example that follows, the range predicate `C2>1` prevents the search for additional matching columns.
- For star joins, a missing key predicate does not cause termination of matching columns that are to be used on the fact table index.

The exceptional cases are:
- At most one IN-list predicate can be a matching predicate on an index.
- For MX accesses and index access with list prefetch, IN-list predicates cannot be used as matching predicates.
- Join predicates cannot qualify as matching predicates when doing a merge join (METHOD=2). For example, T1.C1=T2.C1 cannot be a matching predicate when doing a merge join, although any local predicates, such as C1='5' can be used.

Join predicates can be used as matching predicates on the inner table of a nested loop join or hybrid join.

Matching index scan example: Assume there is an index on T(C1,C2,C3,C4):

```
SELECT * FROM T
  WHERE C1=1 AND C2>1
    AND C3=1;
```

Two matching columns occur in this example. The first one comes from the predicate C1=1, and the second one comes from C2>1. The range predicate on C2 prevents C3 from becoming a matching column.

Index screening

In *index screening*, predicates are specified on index key columns but are not part of the matching columns. Those predicates improve the index access by reducing the number of rows that qualify while searching the index. For example, with an index on T(C1,C2,C3,C4) in the following SQL statement, C3>0 and C4=2 are index screening predicates.

```
SELECT * FROM T
  WHERE C1 = 1
    AND C3 > 0 AND C4 = 2
    AND C5 = 8;
```

The predicates can be applied on the index, but they are not matching predicates. C5=8 is not an index screening predicate, and it must be evaluated when data is retrieved. The value of MATCHCOLS in the plan table is 1.

EXPLAIN does not directly tell when an index is screened; however, if MATCHCOLS is less than the number of index key columns, it indicates that index screening is possible.

Nonmatching index scan (ACCESSTYPE=I and MATCHCOLS=0)

In a *nonmatching index scan* no matching columns are in the index. Hence, all the index keys must be examined.

Because a nonmatching index usually provides no filtering, only a few cases provide an efficient access path. The following situations are examples:

- When index screening predicates exist

  In that case, not all of the data pages are accessed.
- When the clause OPTIMIZE FOR n ROWS is used

  That clause can sometimes favor a nonmatching index, especially if the index gives the ordering of the ORDER BY clause or GROUP BY clause.
- When more than one table exists in a nonsegmented table space

  In that case, a table space scan reads irrelevant rows. By accessing the rows through the nonmatching index, fewer rows are read.

IN-list index scan (ACCESSTYPE=N)

An *IN-list index scan* is a special case of the matching index scan, in which a single indexable IN predicate is used as a matching equal predicate.

You can regard the IN-list index scan as a series of matching index scans with the values in the IN predicate being used for each matching index scan. The following example has an index on (C1,C2,C3,C4) and might use an IN-list index scan:

```
SELECT * FROM T
  WHERE C1=1 AND C2 IN (1,2,3)
    AND C3>0 AND C4<100;
```

The plan table shows MATCHCOLS = 3 and ACCESSTYPE = N. The IN-list scan is performed as the following three matching index scans:

```
(C1=1,C2=1,C3>0), (C1=1,C2=2,C3>0), (C1=1,C2=3,C3>0)
```

Parallelism is supported for queries that involve IN-list index access. These queries used to run sequentially in previous releases of DB2, although parallelism could have been used when the IN-list access was for the inner table of a parallel group. Now, in environments in which parallelism is enabled, you can see a reduction in elapsed time for queries that involve IN-list index access for the outer table of a parallel group.

Multiple index access (ACCESSTYPE is M, MX, MI, or MU)

Multiple index access uses more than one index to access a table. It is a good access path when:

- No single index provides efficient access.
- A combination of index accesses provides efficient access.

RID lists are constructed for each of the indexes involved. The unions or intersections of the RID lists produce a final list of qualified RIDs that is used to retrieve the result rows, using list prefetch. You can consider multiple index access as an extension to list prefetch with more complex RID retrieval operations in its first phase. The complex operators are union and intersection.

DB2 chooses multiple index access for the following query:

```
SELECT * FROM EMP
   WHERE (AGE = 34) OR
         (AGE = 40 AND JOB = 'MANAGER');
```

For this query:

- EMP is a table with columns EMPNO, EMPNAME, DEPT, JOB, AGE, and SAL.
- EMPX1 is an index on EMP with key column AGE.
- EMPX2 is an index on EMP with key column JOB.

The plan table contains a sequence of rows describing the access. For this query, ACCESSTYPE uses the following values:

Value	Meaning
M	Start of multiple index access processing
MX	Indexes are to be scanned for later union or intersection
MI	An intersection (AND) is performed
MU	A union (OR) is performed

The following steps relate to the previous query and the values shown for the plan table in Table 170 on page 953:

1. Index EMPX1, with matching predicate AGE= 34, provides a set of candidates for the result of the query. The value of MIXOPSEQ is 1.
2. Index EMPX1, with matching predicate AGE = 40, also provides a set of candidates for the result of the query. The value of MIXOPSEQ is 2.
3. The first intersection (AND) is done, and the value of MIXOPSEQ is 3. This MI removes the two previous candidate lists (produced by MIXOPSEQs 2 and 3) by intersecting them to form an intermediate candidate list, IR1, which is not shown in PLAN_TABLE.

4. Index EMPX2, with matching predicate JOB='MANAGER', also provides a set of candidates for the result of the query. The value of MIXOPSEQ is 4.
5. The last step, where the value MIXOPSEQ is 5, is a union (OR) of the two remaining candidate lists, which are IR1 and the candidate list produced by MIXOPSEQ 1. This final union gives the result for the query.

Table 170. Plan table output for a query that uses multiple indexes. Depending on the filter factors of the predicates, the access steps can appear in a different order.

PLAN-NO	TNAME	ACCESS-TYPE	MATCH-COLS	ACCESS-NAME	PREFETCH	MIXOP-SEQ
1	EMP	M	0		L	0
1	EMP	MX	1	EMPX1		1
1	EMP	MX	1	EMPX1		2
1	EMP	MI	0			3
1	EMP	MX	1	EMPX2		4
1	EMP	MU	0			5

In this example, the steps in the multiple index access follow the physical sequence of the predicates in the query. This is not always the case. The multiple index steps are arranged in an order that uses RID pool storage most efficiently and for the least amount of time.

One-fetch access (ACCESSTYPE=I1)

One-fetch index access requires retrieving only one row. It is the best possible access path and is chosen whenever it is available. It applies to a statement with a MIN or MAX aggregate function: the order of the index allows a single row to give the result of the function.

One-fetch index access is a possible access path when:

- The query includes only one table.
- The query includes only one aggregate function (either MIN or MAX).
- Either no predicate or all predicates are matching predicates for the index.
- The query includes no GROUP BY clause.
- Aggregate functions are on:
  - The first index column if no predicates exist
  - The last matching column of the index if the last matching predicate is a range type
  - The next index column (after the last matching column) if all matching predicates are equal type

Queries using one-fetch index access: Assuming that an index exists on T(C1,C2,C3), the following queries use one-fetch index scan:

```
SELECT MIN(C1) FROM T;
SELECT MIN(C1) FROM T WHERE C1>5;
SELECT MIN(C1) FROM T WHERE C1>5 AND C1<10;
SELECT MIN(C2) FROM T WHERE C1=5;
SELECT MAX(C1) FROM T;
SELECT MAX(C2) FROM T WHERE C1=5 AND C2<10;
SELECT MAX(C2) FROM T WHERE C1=5 AND C2>5 AND C2<10;
SELECT MAX(C2) FROM T WHERE C1=5 AND C2 BETWEEN 5 AND 10;
```

Index-only access (INDEXONLY=Y)

With *index-only access*, the access path does not require any data pages because the access information is available in the index. Conversely, when an SQL statement requests a column that is not in the index, updates any column in the table, or deletes a row, DB2 has to access the associated data pages. Because the index is almost always smaller than the table itself, an index-only access path usually processes the data efficiently.

With an index on T(C1,C2), the following queries can use index-only access:

```
SELECT C1, C2 FROM T WHERE C1 > 0;
SELECT C1, C2 FROM T;
SELECT COUNT(*) FROM T WHERE C1 = 1;
```

Equal unique index (MATCHCOLS=number of index columns)

An index that is fully matched and unique, and in which all matching predicates are equal-predicates, is called an *equal unique index* case. This case guarantees that only one row is retrieved. If there is no one-fetch index access available, this is considered the most efficient access over all other indexes that are not equal unique. (The uniqueness of an index is determined by whether or not it was defined as unique.)

Sometimes DB2 can determine that an index that is not fully matching is actually an equal unique index case. Assume the following case:

```
Unique Index1: (C1, C2)
Unique Index2: (C2, C1, C3)

SELECT C3 FROM T
  WHERE C1 = 1 AND C2 = 5;
```

Index1 is a fully matching equal unique index. However, Index2 is also an equal unique index even though it is not fully matching. Index2 is the better choice because, in addition to being equal and unique, it also provides index-only access.

UPDATE using an index

If no index key columns are updated, you can use an index while performing an UPDATE operation.

To use a matching index scan to update an index in which its key columns are being updated, the following conditions must be met:

- Each updated key column must have a corresponding predicate of the form ″index_key_column = constant″ or ″index_key_column IS NULL″.
- If a view is involved, WITH CHECK OPTION must not be specified.

For updates that do not involve dynamic scrollable cursors, DB2 can use list prefetch, multiple index access, or IN-list access. With list prefetch or multiple index access, any index or indexes can be used in an UPDATE operation. Of course, to be chosen, those access paths must provide efficient access to the data.

A positioned update that uses a dynamic scrollable cursor cannot use an access path with list prefetch, or multiple index access. This means that indexes that do not meet the preceding criteria cannot be used to locate the rows to be updated.

Interpreting access to two or more tables (join)

A *join* operation retrieves rows from more than one table and combines them. The operation specifies at least two tables, but they need not be distinct.

This section begins with "Definitions and examples of join operations" and continues with descriptions of the methods of joining that can be indicated in a plan table:

- "Nested loop join (METHOD=1)" on page 957
- "Merge scan join (METHOD=2)" on page 959
- "Hybrid join (METHOD=4)" on page 961
- "Star join (JOIN_TYPE='S')" on page 963

Definitions and examples of join operations

This section contains definitions and examples that are related to join operations.

Definitions: A *composite table* represents the result of accessing one or more tables in a query. If a query contains a single table, only one composite table exists. If one or more joins are involved, an *outer composite table* consists of the intermediate result rows from the previous join step. This intermediate result may, or may not, be materialized into a work file.

The *new table* (or *inner table*) in a join operation is the table that is newly accessed in the step.

A join operation can involve more than two tables. In these cases, the operation is carried out in a series of steps. For non-star joins, each step joins only two tables.

Example: Figure 103 shows a two-step join operation.

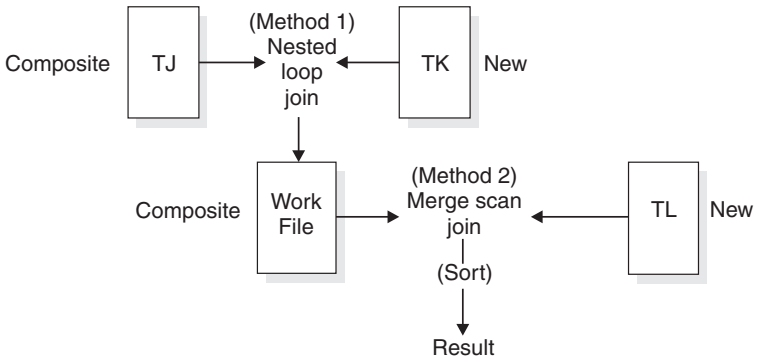


Figure 103. Two-step join operation

DB2 performs the following steps to complete the join operation:

1. Accesses the first table (METHOD=0), named TJ (TNAME), which becomes the composite table in step 2.
2. Joins the new table TK to TJ, forming a new composite table.
3. Sorts the new table TL (SORTN_JOIN=Y) and the composite table (SORTC_JOIN=Y), and then joins the two sorted tables.
4. Sorts the final composite table (TNAME is blank) into the desired order (SORTC_ORDERBY=Y).

Figure 103

Table 171 and Table 172 show a subset of columns in a plan table for this join operation.

Table 171. Subset of columns for a two-step join operation

METHOD	TNAME	ACCESS-TYPE	MATCH-COLS	ACCESS-NAME	INDEX-ONLY	TSLOCK-MODE
0	TJ	I	1	TJX1	N	IS
1	TK	I	1	TKX1	N	IS
2	TL	I	0	TLX1	Y	S
3			0		N	

Table 172. Subset of columns for a two-step join operation

SORTN UNIQ	SORTN JOIN	SORTN ORDERBY	SORTN GROUPBY	SORTC UNIQ	SORTC JOIN	SORTC ORDERBY	SORTC GROUPBY
N	N	N	N	N	N	N	N
N	N	N	N	N	N	N	N
N	Y	N	N	N	Y	N	N
N	N	N	N	N	N	Y	N

Definitions: A join operation typically matches a row of one table with a row of another on the basis of a *join condition*. For example, the condition might specify that the value in column A of one table equals the value of column X in the other table (WHERE T1.A = T2.X).

Two kinds of joins differ in what they do with rows in one table that do not match on the join condition with any row in the other table:

- An *inner join* discards rows of either table that do not match any row of the other table.
- An *outer join* keeps unmatched rows of one or the other table, or of both. A row in the composite table that results from an unmatched row is filled out with null values. As Table 173 shows, outer joins are distinguished by which unmatched rows they keep.

Table 173. Join types and kept unmatched rows

Outer join type	Included unmatched rows
Left outer join	The composite (outer) table
Right outer join	The new (inner) table
Full outer join	Both tables

Example: Suppose that you issue the following statement to explain an outer join:

```
EXPLAIN PLAN SET QUERYNO = 10 FOR
SELECT PROJECT, COALESCE(PROJECTS.PROD#, PRODNUM) AS PRODNUM,
       PRODUCT, PART, UNITS
  FROM PROJECTS LEFT JOIN
      (SELECT PART,
         COALESCE(PARTS.PROD#, PRODUCTS.PROD#) AS PRODNUM,
         PRODUCTS.PRODUCT
         FROM PARTS FULL OUTER JOIN PRODUCTS
             ON PARTS.PROD# = PRODUCTS.PROD#) AS TEMP
  ON PROJECTS.PROD# = PRODNUM
```

Table 174 shows a subset of the plan table for the outer join.

Table 174. Plan table output for an example with outer joins

QUERYNO	QBLOCKNO	PLANNO	TNAME	JOIN_TYPE
10	1	1	PROJECTS	
10	1	2	TEMP	L
10	2	1	PRODUCTS	
10	2	2	PARTS	F

Column JOIN_TYPE identifies the type of outer join with one of these values:
- F for FULL OUTER JOIN
- L for LEFT OUTER JOIN
- Blank for INNER JOIN or no join

At execution, DB2 converts every right outer join to a left outer join; thus JOIN_TYPE never identifies a right outer join specifically.

Materialization with outer join: Sometimes DB2 has to materialize a result table when an outer join is used in conjunction with other joins, views, or nested table expressions. You can tell when this happens by looking at the TABLE_TYPE and TNAME columns of the plan table. When materialization occurs, TABLE_TYPE contains a W, and TNAME shows the name of the materialized table as DSNWFQB(*xx*), where *xx* is the number of the query block (QBLOCKNO) that produced the work file.

Nested loop join (METHOD=1)

This section describes nested loop join, which is common join method. Figure 104 illustrates a nested loop join.

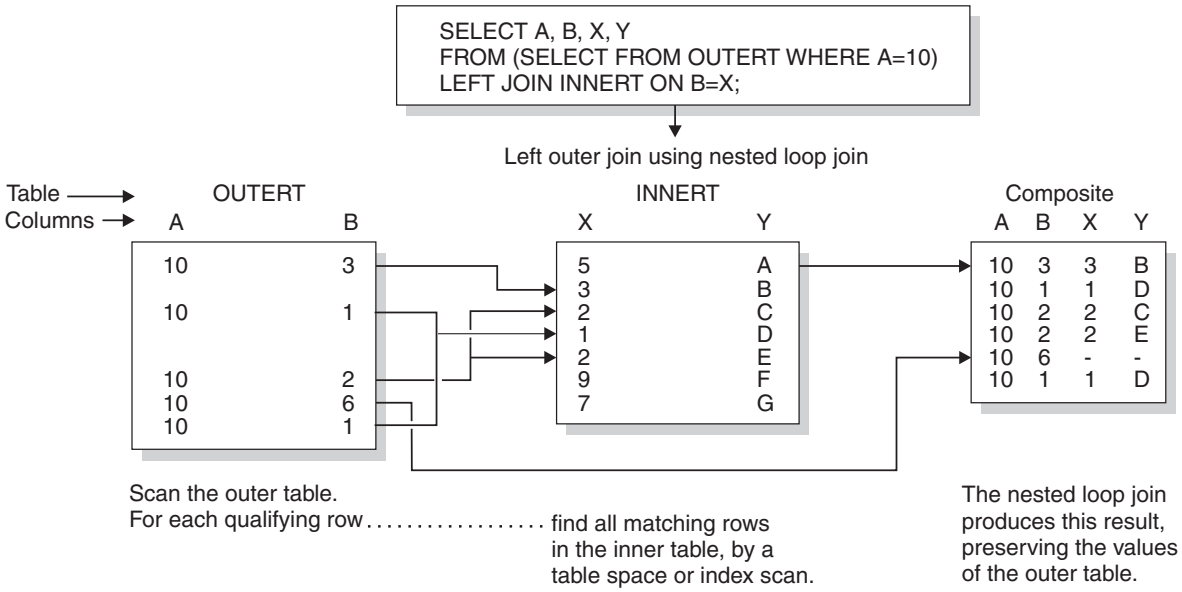


Figure 104. Nested loop join for a left outer join

Method of joining

DB2 scans the composite (outer) table. For each row in that table that qualifies (by satisfying the predicates on that table), DB2 searches for matching rows of the new

(inner) table. It concatenates any it finds with the current row of the composite table. If no rows match the current row, then:
- For an inner join, DB2 discards the current row.
- For an outer join, DB2 concatenates a row of null values.

Stage 1 and stage 2 predicates eliminate unqualified rows during the join. (For an explanation of those types of predicate, see "Stage 1 and stage 2 predicates" on page 751.) DB2 can scan either table using any of the available access methods, including table space scan.

Performance considerations

The nested loop join repetitively scans the inner table. That is, DB2 scans the outer table once, and scans the inner table as many times as the number of qualifying rows in the outer table. Therefore, the nested loop join is usually the most efficient join method when the values of the join column passed to the inner table are in sequence and the index on the join column of the inner table is clustered, or the number of rows retrieved in the inner table through the index is small.

When nested loop join is used

Nested loop join is often used if:
- The outer table is small.
- Predicates with small filter factors reduce the number of qualifying rows in the outer table.
- An efficient, highly clustered index exists on the join columns of the inner table.
- The number of data pages accessed in the inner table is small.
- No join columns exist. Hybrid and sort merge joins require join columns; nested loop joins do not.

Example: left outer join: Figure 104 on page 957 illustrates a nested loop for a left outer join. The outer join preserves the unmatched row in OUTERT with values A=10 and B=6. The same join method for an inner join differs only in discarding that row.

Example: one-row table priority: For a case like the following example, with a unique index on T1.C2, DB2 detects that T1 has only one row that satisfies the search condition. DB2 makes T1 the first table in a nested loop join.

```
SELECT * FROM T1, T2
  WHERE  T1.C1 = T2.C1 AND
         T1.C2 = 5;
```

Example: Cartesian join with small tables first: A *Cartesian join* is a form of nested loop join in which there are no join predicates between the two tables. DB2 usually avoids a Cartesian join, but sometimes it is the most efficient method, as in the following example. The query uses three tables: T1 has 2 rows, T2 has 3 rows, and T3 has 10 million rows.

```
SELECT * FROM T1, T2, T3
  WHERE  T1.C1 = T3.C1 AND
         T2.C2 = T3.C2 AND
         T3.C3 = 5;
```

Join predicates are between T1 and T3 and between T2 and T3. There is no join predicate between T1 and T2.

Assume that 5 million rows of T3 have the value C3=5. Processing time is large if T3 is the outer table of the join and tables T1 and T2 are accessed for each of 5 million rows.

However if all rows from T1 and T2 are joined, without a join predicate, the 5 million rows are accessed only six times, once for each row in the Cartesian join of T1 and T2. It is difficult to say which access path is the most efficient. DB2 evaluates the different options and could decide to access the tables in the sequence T1, T2, T3.

Sorting the composite table: Your plan table could show a nested loop join that includes a sort on the composite table. DB2 might sort the composite table (the outer table in Figure 104) if the following conditions exist:

- The join columns in the composite table and the new table are not in the same sequence.
- The join column of the composite table has no index.
- The index is poorly clustered.

Nested loop join with a sorted composite table has the following performance advantages:

- Uses sequential detection efficiently to prefetch data pages of the new table, reducing the number of synchronous I/O operations and the elapsed time.
- Avoids repetitive full probes of the inner table index by using the index look-aside.

Merge scan join (METHOD=2)

Merge scan join is also known as *merge join* or *sort merge join*. For this method, there must be one or more predicates of the form TABLE1.COL1=TABLE2.COL2, where the two columns have the same data type and length attribute.

Method of joining

Figure 105 illustrates a merge scan join.

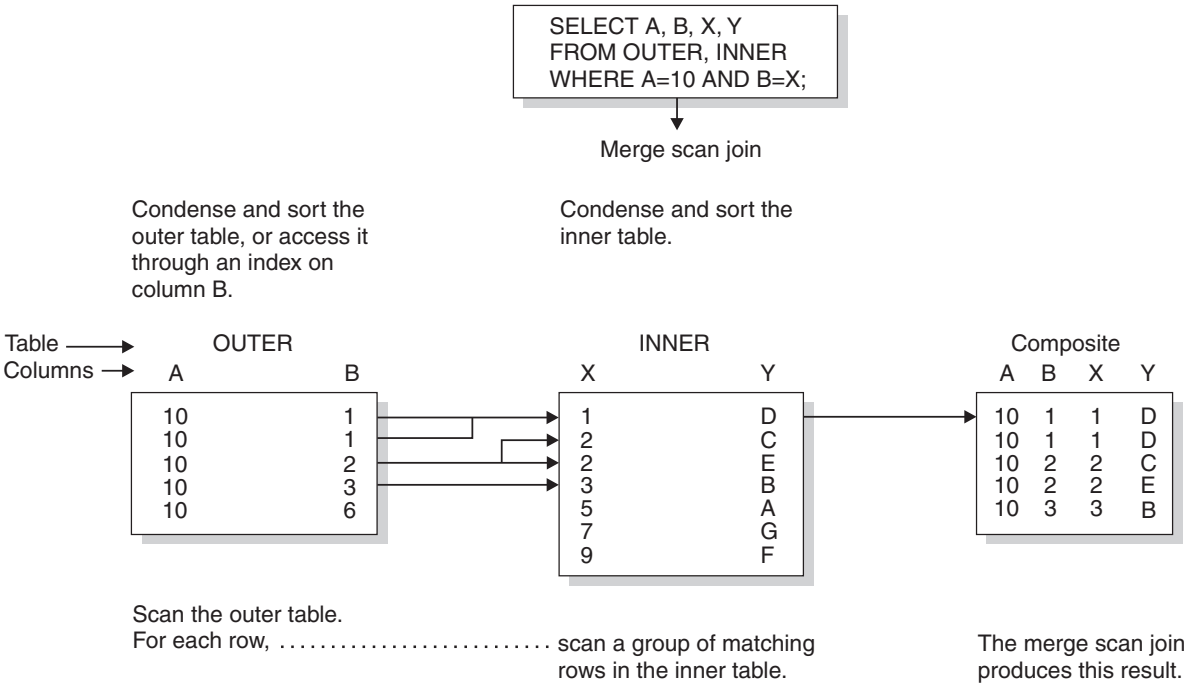


Figure 105. Merge scan join

DB2 scans both tables in the order of the join columns. If no efficient indexes on the join columns provide the order, DB2 might sort the outer table, the inner table, or both. The inner table is put into a work file; the outer table is put into a work file only if it must be sorted. When a row of the outer table matches a row of the inner table, DB2 returns the combined rows.

DB2 then reads another row of the inner table that might match the same row of the outer table and continues reading rows of the inner table as long as there is a match. When there is no longer a match, DB2 reads another row of the outer table.

- If that row has the same value in the join column, DB2 reads again the matching group of records from the inner table. Thus, a group of duplicate records in the inner table is scanned as many times as there are matching records in the outer table.
- If the outer row has a new value in the join column, DB2 searches ahead in the inner table. It can find any of the following rows:
  - Unmatched rows in the inner table, with lower values in the join column.
  - A new matching inner row. DB2 then starts the process again.
  - An inner row with a higher value of the join column. Now the row of the outer table is unmatched. DB2 searches ahead in the outer table, and can find any of the following rows:
    - Unmatched rows in the outer table.
    - A new matching outer row. DB2 then starts the process again.
    - An outer row with a higher value of the join column. Now the row of the inner table is unmatched, and DB2 resumes searching the inner table.

If DB2 finds an unmatched row:

For an inner join, DB2 discards the row.

For a left outer join, DB2 discards the row if it comes from the inner table and keeps it if it comes from the outer table.

For a full outer join, DB2 keeps the row.

When DB2 keeps an unmatched row from a table, it concatenates a set of null values as if that matched from the other table. A merge scan join must be used for a full outer join.

Performance considerations

A full outer join by this method uses all predicates in the ON clause to match the two tables and reads every row at the time of the join. Inner and left outer joins use only stage 1 predicates in the ON clause to match the tables. If your tables match on more than one column, it is generally more efficient to put all the predicates for the matches in the ON clause, rather than to leave some of them in the WHERE clause.

For an inner join, DB2 can derive extra predicates for the inner table at bind time and apply them to the sorted outer table to be used at run time. The predicates can reduce the size of the work file needed for the inner table.

If DB2 has used an efficient index on the join columns, to retrieve the rows of the inner table, those rows are already in sequence. DB2 puts the data directly into the work file without sorting the inner table, which reduces the elapsed time.

When merge scan join is used

A merge scan join is often used if:

- The qualifying rows of the inner and outer table are large, and the join predicate does not provide much filtering; that is, in a many-to-many join.
- The tables are large and have no indexes with matching columns.
- Few columns are selected on inner tables. This is the case when a DB2 sort is used. The fewer the columns to be sorted, the more efficient the sort is.

Hybrid join (METHOD=4)

The method applies only to an inner join and requires an index on the join column of the inner table. Figure 106 illustrates a hybrid join.

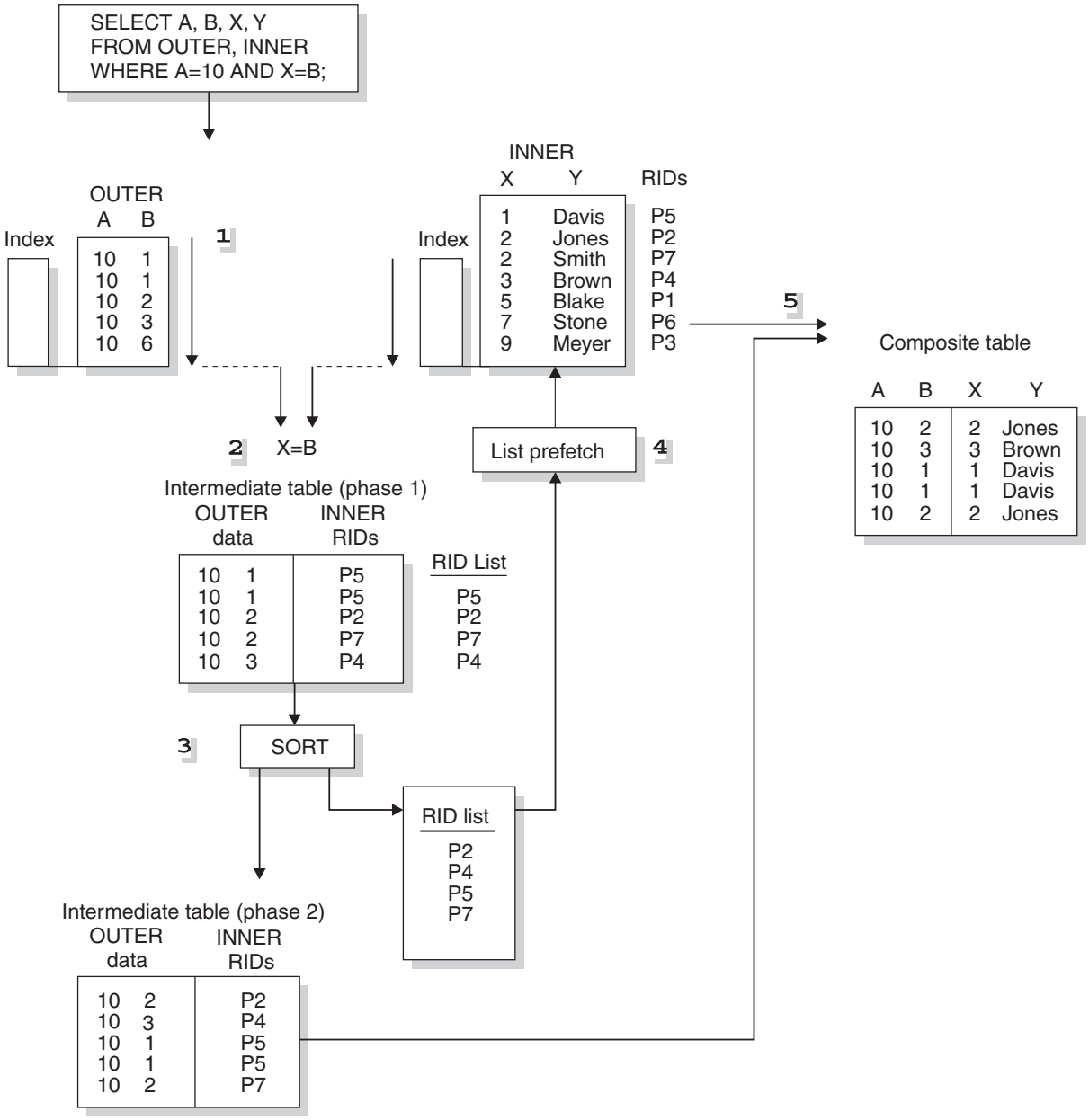


Figure 106. Hybrid join (SORTN_JOIN='Y')

Method of joining

The method requires obtaining RIDs in the order needed to use list prefetch. The steps are shown in Figure 106 on page 961. In that example, both the outer table (OUTER) and the inner table (INNER) have indexes on the join columns.

DB2 performs the following steps:

1. Scans the outer table (OUTER).
2. Joins the outer table with RIDs from the index on the inner table. The result is the phase 1 intermediate table. The index of the inner table is scanned for every row of the outer table.
3. Sorts the data in the outer table and the RIDs, creating a sorted RID list and the phase 2 intermediate table. The sort is indicated by a value of Y in column SORTN_JOIN of the plan table. If the index on the inner table is a well-clustered index, DB2 can skip this sort; the value in SORTN_JOIN is then N.
4. Retrieves the data from the inner table, using list prefetch.
5. Concatenates the data from the inner table and the phase 2 intermediate table to create the final composite table.

Possible results from EXPLAIN for hybrid join

Table 181 shows possible EXPLAIN results from a hybrid join and an explanation of each column value.

Table 181. Explanation of EXPLAIN results for a hybrid join

Column value	Explanation
METHOD='4'	A hybrid join was used.
SORTC_JOIN='Y'	The composite table was sorted.
SORTN_JOIN='Y'	The intermediate table was sorted in the order of inner table RIDs. A non-clustered index accessed the inner table RIDs.
SORTN_JOIN='N'	The intermediate table RIDs were not sorted. A clustered index retrieved the inner table RIDs, and the RIDs were already well ordered.
PREFETCH='L'	Pages were read using list prefetch.

Performance considerations

Hybrid join uses list prefetch more efficiently than nested loop join, especially if there are indexes on the join predicate with low cluster ratios. It also processes duplicates more efficiently because the inner table is scanned only once for each set of duplicate values in the join column of the outer table.

If the index on the inner table is highly clustered, there is no need to sort the intermediate table (SORTN_JOIN=N). The intermediate table is placed in a table in memory rather than in a work file.

When hybrid join is used

Hybrid join is often used if:

- A nonclustered index or indexes are used on the join columns of the inner table.
- The outer table has duplicate qualifying rows.

Star join (JOIN_TYPE='S')

Star join is a special join technique that DB2 uses to efficiently join tables that form a star schema. A star schema is a logical database design that is included in decision support applications. A star schema is composed of a fact table and a number of dimension tables that are connected to it. A dimension table contains several values that are given an ID, which is used in the fact table instead of all the values.

You can think of the fact table, which is much larger than the dimension tables, as being in the center surrounded by dimension tables; the result resembles a star formation. Figure 107 illustrates the star formation.

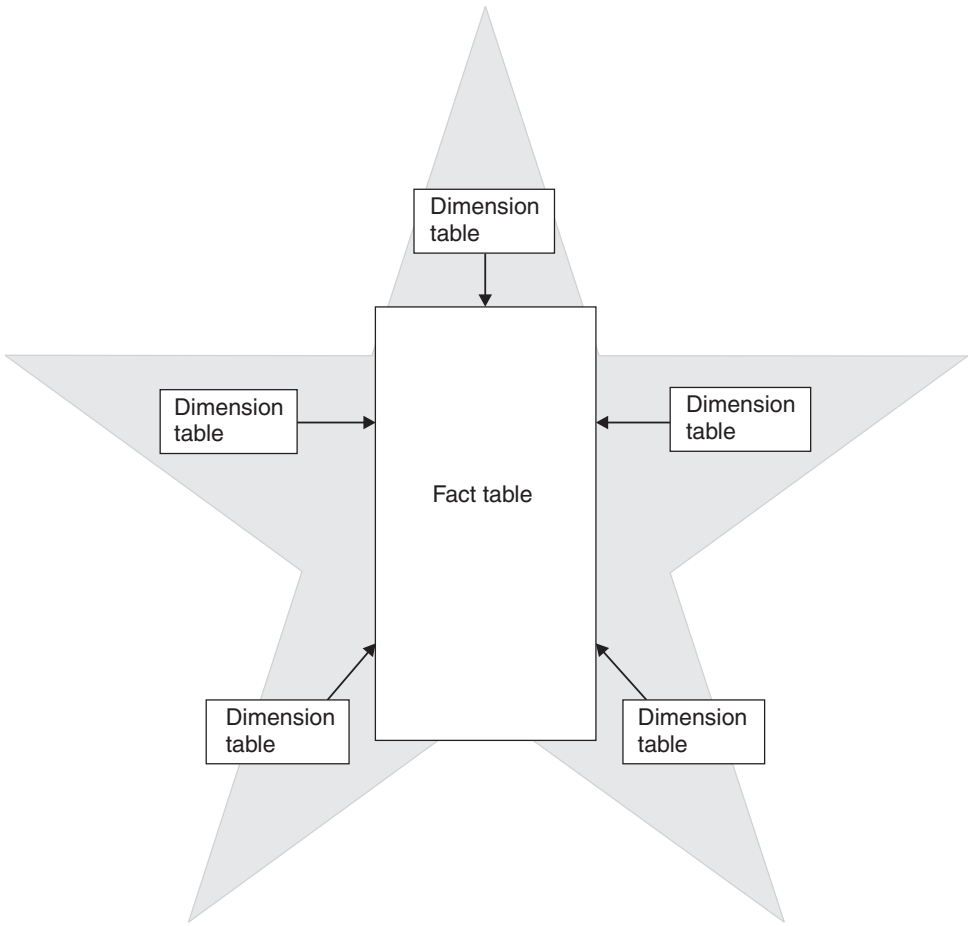


Figure 107. Star schema with a fact table and dimension tables

Unlike the steps in the other join methods (nested loop join, merge scan join, and hybrid join) in which only two tables are joined in each step, a step in the star join method can involve three or more tables. Dimension tables are joined to the fact table via a multi-column index that is defined on the fact table. Therefore, having a well-defined, multi-column index on the fact table is critical for efficient star join processing.

Example of a star schema

For an example of a star schema consider the following scenario. A star schema is composed of a fact table for sales, with dimension tables connected to it for time, products, and geographic locations. The time table has an ID for each month, its

quarter, and the year. The product table has an ID for each product item and its class and its inventory. The geographic location table has an ID for each city and its country.

In this scenario, the sales table contains three columns with IDs from the dimension tables for time, product, and location instead of three columns for time, three columns for products, and two columns for location. Thus, the size of the fact table is greatly reduced. In addition, if you needed to change an item, you would do it once in a dimension table instead of several times for each instance of the item in the fact table.

You can create even more complex star schemas by normalizing a dimension table into several tables. The normalized dimension table is called a *snowflake*. Only one of the tables in the snowflake joins directly wiht the fact table.

When star join is used

To access the data in a star schema, you often write SELECT statements that include join operations between the fact table and the dimension tables, but no join operations between dimension tables. DB2 uses star join processing as the join type for the query if the following conditions are true:

- The query references at least two dimensions.
- All join predicates are between the fact table and the dimension tables, or within tables of the same snowflake. If a snowflake is connected to the fact table, only one table in the snowflake (the central dimension table) can be joined to the fact table.
- All join predicates between the fact table and dimension tables are equi-join predicates.
- All join predicates between the fact table and dimension tables are Boolean term predicates. For more information, see "Boolean term (BT) predicates" on page 752.
- There are no instances of predicates that consist of a local predicate on a dimension table and a local predicate on a different table that are connected with an OR logical operator.
- No correlated subqueries cross dimensions.
- No single fact table column is joined to columns of different dimension tables in join predicates. For example, fact table column `F1` cannot be joined to column `D1` of dimension table T1 and also joined to column `D2` of dimension table T2.
- After DB2 simplifies join operations, no outer join operations exist. For more information, see "When DB2 simplifies join operations" on page 770.
- The data type and length of both sides of a join predicate are the same.
- The value of subsystem parameter STARJOIN is 1, or the cardinality of the fact table to the largest dimension table meets the requirements specified by the value of the subsystem parameter. The values of STARJOIN and cardinality requirements are:

  -1 Star join is disabled. This is the default.

  1 Star join is enabled. The one table with the largest cardinality is the fact table. However, if there is more than one table with this cardinality, star join is not enabled.

  0 Star join is enabled if the cardinality of the fact table is at least 25 times the cardinality of the largest dimension that is a base table that is joined to the fact table.

n Star join is enabled if the cardinality of the fact table is at least *n* times the cardinality of the largest dimension that is a base table that is joined to the fact table, where 2≤*n*≤32768.

You can set the subsystem parameter STARJOIN by using the STAR JOIN QUERIES field on the DSNTIP8 installation panel.

- The number of tables in the star schema query block, including the fact table, dimensions tables, and snowflake tables, meet the requirements that are specified by the value of subsystem parameter SJTABLES. The value of SJTABLES is considered only if the subsystem parameter STARJOIN qualifies the query for star join. The values of SJTABLES are:

  1, 2, or 3 Star join is always considered.

  4 to 255 Star join is considered if the query block has at least the specified number of tables. If star join is enabled, 10 is the default value.

  256 and greater Star join will never be considered.

  Star join, which can reduce bind time significantly, does not provide optimal performance in all cases. Performance of star join depends on a number of factors such as the available indexes on the fact table, the cluster ratio of the indexes, and the selectivity of rows through local and join predicates. Follow these general guidelines for setting the value of SJTABLES:

  – If you have queries that reference less than 10 tables in a star schema database and you want to make the star join method applicable to all qualified queries, set the value of SJTABLES to the minimum number of tables used in queries that you want to be considered for star join.

    Example: Suppose that you query a star schema database that has one fact table and three dimension tables. You should set SJTABLES to 4.

  – If you want to use star join for relatively large queries that reference a star schema database but are not necessarily suitable for star join, use the default. The star join method will be considered for all qualified queries that have 10 or more tables.

  – If you have queries that reference a star schema database but, in general, do not want to use star join, consider setting SJTABLES to a higher number, such as 15, if you want to drastically cut the bind time for large queries and avoid a potential bind time SQL return code -101 for large qualified queries.

  For recommendations on indexes for star schemas, see "Creating indexes for efficient star-join processing" on page 796.

Examples: query with three dimension tables: Suppose that you have a store in San Jose and want information about sales of audio equipment from that store in 2000. For this example, you want to join the following tables:

- A fact table for SALES (S)
- A dimension table for TIME (T) with columns for an ID, month, quarter, and year
- A dimension table for geographic LOCATION (L) with columns for an ID, city, region, and country
- A dimension table for PRODUCT (P) with columns for an ID, product item, class, and inventory

You could write the following query to join the tables:

```
SELECT *
  FROM SALES S, TIME T, PRODUCT P, LOCATION L
  WHERE S.TIME = T.ID      AND
        S.PRODUCT = P.ID   AND
        S.LOCATION = L.ID  AND
        T.YEAR = 2000      AND
        P.CLASS = 'SAN JOSE';
```

You would use the following index:

```
CREATE INDEX  XSALES_TPL ON SALES (TIME, PRODUCT, LOCATION);
```

You EXPLAIN output looks like Table 182.

Table 182. Plan table output for a star join example with TIME, PRODUCT, and LOCATION

QUERYNO	QBLOCKNO	METHOD	TNAME	JOIN TYPE	SORTN JOIN	ACCESS TYPE
1	1	0	TIME	S	Y	R
1	1	1	PRODUCT	S	Y	R
1	1	1	LOCATION	S	Y	R
1	1	1	SALES	S		I

All snowflakes are processed before the central part of the star join, as individual query blocks, and are materialized into work files. There is a work file for each snowflake. The EXPLAIN output identifies these work files by naming them DSN_DIM_TBLX(*nn*), where *nn* indicates the corresponding QBLOCKNO for the snowflake.

This next example shows the plan for a star join that contains two snowflakes. Suppose that two new tables MANUFACTURER (M) and COUNTRY (C) are added to the tables in the previous example to break dimension tables PRODUCT (P) and LOCATION (L) into snowflakes:

- The PRODUCT table has a new column MID that represents the manufacturer.
- Table MANUFACTURER (M) has columns for MID and name to contain manufacturer information.
- The LOCATION table has a new column CID that represents the country.
- Table COUNTRY (C) has columns for CID and name to contain country information.

You could write the following query to join all the tables:

```
SELECT *
  FROM SALES S, TIME T, PRODUCT P, MANUFACTURER M,
       LOCATION L, COUNTRY C
  WHERE S.TIME = T.ID      AND
        S.PRODUCT = P.ID   AND
        P.MID = M.MID      AND
        S.LOCATION = L.ID  AND
        L.CID = C.CID      AND
        T.YEAR = 2000      AND
        M.NAME = 'some_company';
```

The joined table pairs (PRODUCT, MANUFACTURER) and (LOCATION, COUNTRY) are snowflakes. The EXPLAIN output of this query looks like Table 183 on page 967.

Table 183. Plan table output for a star join example with snowflakes

QUERYNO	QBLOCKNO	METHOD	TNAME	JOIN TYPE	SORTN JOIN	ACCESS TYPE
1	1	0	TIME	S	Y	R
1	1	1	DSN_DIM_TBLX(02)	S	Y	R
1	1	1	SALES	S		I
1	1	1	DSN_DIM_TBLX(03)		Y	T
1	2	0	PRODUCT			R
1	2	1	MANUFACTURER			I
1	3	0	LOCATION			R
1	3	4	COUNTRY			I

Note: This query consists of three query blocks:

- QBLOCKNO=1: The main star join block
- QBLOCKNO=2: A snowflake (PRODUCT, MANUFACTURER) that is materialized into work file DSN_DIM_TBLX(02)
- QBLOCKNO=3: A snowflake (LOCATION, COUNTRY) that is materialized into work file DSN_DIM_TBLX(03)

The joins in the snowflakes are processed first, and each snowflake is materialized into a work file. Therefore, when the main star join block (QBLOCKNO=1) is processed, it contains four tables: SALES (the fact table), TIME (a base dimension table), and the two snowflake work files.

In this example, in the main star join block, the star join method is used for the first three tables (as indicated by S in the JOIN TYPE column of the plan table) and the remaining work file is joined by the nested loop join with sparse index access on the work file (as indicated by T in the ACCESSTYPE column for DSN_DIM_TBLX(3)).

Dedicated virtual memory pool for star join operations

You can create a dedicated virtual memory pool for star join operations. When the virtual memory pool is enabled for star joins, DB2 caches data from work files that are used by star join queries. A virtual memory pool dedicated to star join operations has the following advantages:

- Immediate data availability. During a star join operation, work files might be scanned many times. If the work-file data is cached in the dedicated virtual memory pool, that data is immediately available for join operations.
- Reduced buffer pool contention. Because the dedicated virtual memory pool caches data separately from the work-file buffer pool, contention with the buffer pool is reduced. Reduced contention improves performance particularly when sort operations are performed concurrently.

To determine the size of the virtual memory pool, perform the following steps:

1. Determine the value of A. Estimate the number of star join queries that run concurrently.
2. Determine the value of B. Estimate the average number of work files that a star join query uses. In typical cases, with highly normalized star schemas, the average number is about three to six work files.
3. Determine the value of C. Estimate the number of work-file rows, the maximum length of the key, and the total of the maximum length of the

relevant columns. Multiply these three values together to find the size of the data caching space for the work file, or the value of C.

4. Multiply (A) * (B) * (C) to determine the size of the pool in MB.

The default virtual memory pool size is 20 MB. To set the pool size, use the SJMXPOOL parameter on the DSNTIP8 installation panel.

Example: The following example shows how to determine the size of the virtual memory pool. Suppose that you issue the following star join query, where SALES is the fact table:

```
SELECT C.COUNTRY, P.PRDNAME, SUM(F.SPRICE)
  FROM SALES F, TIME T, PROD P, LOC L, SCOUN C
  WHERE F.TID = T.TID AND
        F.PID = P.PID AND
        F.LID = L.LID AND
        L.CID = C.CID AND
        P.PCODE IN (4, 7, 21, 22, 53)
  GROUP BY .COUNTRY, P.PRDNAME;
```

The EXPLAIN output of this query looks like Table 184.

Table 184. EXPLAIN output for a star join query

QBLOCKNO	PLANNO	TNAME	METHOD	JOIN_ TYPE	ACCESS TYPE	ACCESS NAME
1	1	TIME	0	S	R	
1	2	PROD	1	S	T	
1	3	SALES	1	S	I	XSALES
1	4	DSN_DIM_TBLX(02)	1		T	
1	5		3			
2	1	LOC	0		R	
2	2	SCOUN	4		I	XSCOUN

For this query, two work files can be cached in memory. These work files, PROD and DSN_DIM_TBLX(02), are indicated by ACCESSTYPE=T.

To determine the size of the dedicated virtual memory pool, perform the following steps:

1. Determine the value of A. Estimate the number of star join queries that run concurrently.

   In this example, based on the type of operation, up to 12 star join queries are expected run concurrently. Therefore, A = 12.

2. Determine the value of B. Estimate the average number of work files that a star join query uses.

   In this example, the star join query uses two work files, PROD and DSN_DIM_TBLX(02). Therefore B = 2.

3. Determine the value of C. Estimate the number of work-file rows, the maximum length of the key, and the total of the maximum length of the relevant columns. Multiply these three values together to find the size of the data caching space for the work file, or the value of C.

   Both PROD and DSN_DIM_TBLX(02) are used to determine the value of C.

   Recommendation: Average the values for a representative sample of work files, and round the value up to determine an estimate for a value of C.

- The number of work-file rows depends on the number of rows that match the predicate. For PROD, 87 rows are stored in the work file because 87 rows match the IN-list predicate. No selective predicate is used for DSN_DIM_TBLX(02), so the entire result of the join is stored in the work file. The work file for DSN_DIM_TBLX(02) holds 2800 rows.
- The maximum length of the key depends on the data type definition of the table's key column. For PID, the key column for PROD, the maximum length is 4. DSN_DIM_TBLX(02) is a work file that results from the join of LOC and SCOUN. The key column that is used in the join is LID from the LOC table. The maximum length of LID is 4.
- The maximum data length depends on the maximum length of the key column and the maximum length of the column that is selected as part of the star join. Add to the maximum data length 1 byte for nullable columns, 2 bytes for varying length columns, and 3 bytes for nullable and varying length columns.

  For the PROD work file, the maximum data length is the maximum length of PID, which is 4, plus the maximum length of PRDNAME, which is 24. Therefore, the maximum data length for the PROD work file is 28. For the DSN_DIM_TBLX(02) workfile, the maximum data length is the maximum length of LID, which is 4, plus the maximum length of COUNTRY, which is 36. Therefore, the maximum data length for the DSN_DIM_TBLX(02) work file is 40.

For PROD, C = (87) * (4 + 28) = 2784 bytes. For DSN_DIM_TBLX(02), C = (2800) * (4 + 40) = 123200 bytes.

The average of these two estimated values for C is approximately 62 KB. Because the number of rows in each work file can vary depending on the selection criteria in the predicate, the value of C should be rounded up to the nearest multiple of 100 KB. Therefore C = 100 KB.

4. Multiply (A) * (B) * (C) to determine the size of the pool in MB.

   The size of the pool is determined by multiplying (12) * (2) * (100KB) = 2.4 MB.

Interpreting data prefetch

Prefetch is a mechanism for reading a set of pages, usually 32, into the buffer pool with only one asynchronous I/O operation. Prefetch can allow substantial savings in both processor cycles and I/O costs. To achieve those savings, monitor the use of prefetch.

A plan table can indicate the use of three kinds of prefetch:

- "Sequential prefetch (PREFETCH=S)"
- "Dynamic prefetch (PREFETCH=D)" on page 970
- "List prefetch (PREFETCH=L)" on page 970

Additionally, you can choose not to use prefetch.

If DB2 does not choose prefetch at bind time, it can sometimes use it at execution time nevertheless. The method is described in "Sequential detection at execution time" on page 972.

Sequential prefetch (PREFETCH=S)

Sequential prefetch reads a sequential set of pages. The maximum number of pages read by a request issued from your application program is determined by the size of the buffer pool used. For each buffer pool size (4 KB, 8 KB, 16 KB, and 32 KB),

Table 185 shows the number pages read by prefetch for each asynchronous I/O.

Table 185. The number of pages read by prefetch, by buffer pool size

Buffer pool size	Number of buffers	Pages read by prefetch (for each asynchronous I/O)
4 KB	<=223 buffers	8 pages
	224-999 buffers	16 pages
	1000+ buffers	32 pages
8 KB	<=112 buffers	4 pages
	113-499 buffers	8 pages
	500+ buffers	16 pages
16 KB	<=56 buffers	2 pages
	57-249 buffers	4 pages
	250+ buffers	8 pages
32 KB	<=16 buffers	0 pages (prefetch disabled)
	17-99 buffers	2 pages
	100+ buffers	4 pages

For certain utilities (LOAD, REORG, RECOVER), the prefetch quantity can be twice as much.

When sequential prefetch is used: Sequential prefetch is generally used for a table space scan.

For an index scan that accesses eight or more consecutive data pages, DB2 requests sequential prefetch at bind time. The index must have a cluster ratio of 80% or higher. Both data pages and index pages are prefetched.

Dynamic prefetch (PREFETCH=D)

Dynamic prefetch can reduce paging and improve performance over sequential prefetch for some data access that involves data that is not on consecutive pages. When DB2 expects that *dynamic prefetch* will be used, DB2 sets PREFETCH=D. At runtime, dynamic prefetch might or might not actually be used. However, DB2 expects dynamic prefetch and optimizes for that behavior.

When dynamic prefetch is used: Dynamic prefetch is used in prefetch situations when the pages that DB2 will access are distributed in a nonconsecutive manner. If the pages are distributed in a sufficiently consecutive manner, sequential prefetch is used instead.

List prefetch (PREFETCH=L)

List prefetch reads a set of data pages determined by a list of RIDs taken from an index. The data pages need not be contiguous. The maximum number of pages that can be retrieved in a single list prefetch is 32 (64 for utilities).

List prefetch can be used in conjunction with either single or multiple index access.

The access method

List prefetch uses the following three steps:

1. RID retrieval: A list of RIDs for needed data pages is found by matching index scans of one or more indexes.
2. RID sort: The list of RIDs is sorted in ascending order by page number.
3. Data retrieval: The needed data pages are prefetched in order using the sorted RID list.

List prefetch does not preserve the data ordering given by the index. Because the RIDs are sorted in page number order before accessing the data, the data is not retrieved in order by any column. If the data must be ordered for an ORDER BY clause or any other reason, it requires an additional sort.

In a hybrid join, if the index is highly clustered, the page numbers might not be sorted before accessing the data.

List prefetch can be used with most matching predicates for an index scan. IN-list predicates are the exception; they cannot be the matching predicates when list prefetch is used.

When list prefetch is used

List prefetch is used:

- Usually with a single index that has a cluster ratio lower than 80%
- Sometimes on indexes with a high cluster ratio, if the estimated amount of data to be accessed is too small to make sequential prefetch efficient, but large enough to require more than one regular read
- Always to access data by multiple index access
- Always to access data from the inner table during a hybrid join
- Usually for updatable cursors when the index contains columns that might be updated.

Bind time and execution time thresholds

DB2 does not consider list prefetch if the estimated number of RIDs to be processed would take more than 50% of the RID pool when the query is executed. You can change the size of the RID pool in the field RID POOL SIZE on installation panel DSNTIPC. The maximum size of a RID pool is 10 000 MB. The maximum size of a single RID list is approximately 26 million RIDs. For information about calculating RID pool size, see “Increasing RID pool size” on page 683.

During execution, DB2 ends list prefetching if more than 25% of the rows in the table (with a minimum of 6524) must be accessed. Record IFCID 0125 in the performance trace, mapped by macro DSNDQW01, indicates whether list prefetch ended.

When list prefetch ends, the query continues processing by a method that depends on the current access path.

- For access through a single index or through the union of RID lists from two indexes, processing continues by a table space scan.
- For index access before forming an intersection of RID lists, processing continues with the next step of multiple index access. If no step remains and no RID list has been accumulated, processing continues by a table space scan.

When DB2 forms an intersection of RID lists, if any list has 32 or fewer RIDs, intersection stops and the list of 32 or fewer RIDs is used to access the data.

Sequential detection at execution time

If DB2 does not choose prefetch at bind time, it can sometimes use prefetch at execution time nevertheless. The method is called *sequential detection*.

When sequential detection is used

DB2 can use sequential detection for both index leaf pages and data pages. It is most commonly used on the inner table of a nested loop join, if the data is accessed sequentially.

If a table is accessed repeatedly using the same statement (for example, DELETE in a do-while loop), the data or index leaf pages of the table can be accessed sequentially. This is common in a batch processing environment. Sequential detection can then be used if access is through:

- SELECT or FETCH statements
- UPDATE and DELETE statements
- INSERT statements when existing data pages are accessed sequentially

DB2 can use sequential detection if it did not choose sequential prefetch at bind time because of an inaccurate estimate of the number of pages to be accessed.

Sequential detection is not used for an SQL statement that is subject to referential constraints.

How to tell whether sequential detection was used

A plan table does not indicate sequential detection, which is not determined until run time. You can determine whether sequential detection was used from record IFCID 0003 in the accounting trace or record IFCID 0006 in the performance trace.

How to tell if sequential detection might be used

The pattern of accessing a page is tracked when the application scans DB2 data through an index. Tracking is done to detect situations where the access pattern that develops is sequential or nearly sequential.

The most recent eight pages are tracked. A page is considered page-sequential if it is within P/2 advancing pages of the current page, where P is the prefetch quantity. P is usually 32.

If a page is page-sequential, DB2 determines further if data access is sequential or nearly sequential. Data access is declared sequential if more than 4 out of the last eight pages are page-sequential; this is also true for index-only access. The tracking is continuous, allowing access to slip into and out of data access sequential.

When data access is first declared sequential, which is called *initial data access* sequential, three page ranges are calculated as follows:

- Let A be the page being requested. RUN1 is defined as the page range of length P/2 pages starting at A.
- Let B be page A + P/2. RUN2 is defined as the page range of length P/2 pages starting at B.
- Let C be page B + P/2. RUN3 is defined as the page range of length P pages starting at C.

For example, assume that page A is 10. Figure 108 on page 973 illustrates the page ranges that DB2 calculates.

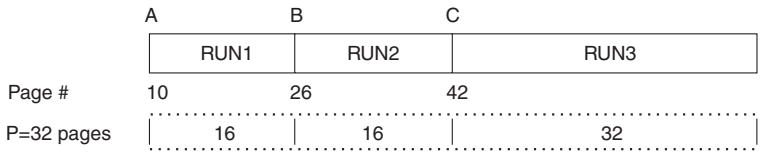


Figure 108. Initial page ranges to determine when to use prefetch

For initial data access sequential, prefetch is requested starting at page A for P pages (RUN1 and RUN2). The prefetch quantity is always P pages.

For subsequent page requests where the page is 1) page sequential and 2) data access sequential is still in effect, prefetch is requested as follows:

- If the desired page is in RUN1, no prefetch is triggered because it was already triggered when data access sequential was first declared.
- If the desired page is in RUN2, prefetch for RUN3 is triggered and RUN2 becomes RUN1, RUN3 becomes RUN2, and RUN3 becomes the page range starting at C+P for a length of P pages.

If a data access pattern develops such that data access sequential is no longer in effect and, thereafter, a new pattern develops that is sequential, then initial data access sequential is declared again and handled accordingly.

Because, at bind time, the number of pages to be accessed can only be estimated, sequential detection acts as a safety net and is employed when the data is being accessed sequentially.

In extreme situations, when certain buffer pool thresholds are reached, sequential prefetch can be disabled. See “Buffer pool thresholds” on page 667 for a description of these thresholds.

Determining sort activity

DB2 can use two general types of sorts that DB2 can use when accessing data. One is a sort of data rows; the other is a sort of row identifiers (RIDs) in a RID list.

Sorts of data

After you run EXPLAIN, DB2 sorts are indicated in PLAN_TABLE. The sorts can be either sorts of the composite table or the new table. If a single row of PLAN_TABLE has a ’Y’ in more than one of the sort composite columns, then one sort accomplishes two things. (DB2 will not perform two sorts when two ’Y’s are in the same row.) For instance, if both SORTC_ORDERBY and SORTC_UNIQ are ’Y’ in one row of PLAN_TABLE, then a single sort puts the rows in order and removes any duplicate rows as well.

The only reason DB2 sorts the new table is for join processing, which is indicated by SORTN_JOIN.

Sorts for group by and order by

These sorts are indicated by SORTC_ORDERBY, and SORTC_GROUPBY in PLAN_TABLE. If there is both a GROUP BY clause and an ORDER BY clause, and if every item in the ORDER-BY list is in the GROUP-BY list, then only one sort is performed, which is marked as SORTC_ORDERBY.

The performance of the sort by the GROUP BY clause is improved when the query accesses a single table and when the GROUP BY column has no index.

Sorts to remove duplicates

This type of sort is used to process a query with SELECT DISTINCT, with a set function such as COUNT(DISTINCT COL1), or to remove duplicates in UNION processing. It is indicated by SORTC_UNIQ in PLAN_TABLE.

Sorts used in join processing

Before joining two tables, it is often necessary to first sort either one or both of them. For hybrid join (METHOD 4) and nested loop join (METHOD 1), the composite table can be sorted to make the join more efficient. For merge join (METHOD 2), both the composite table and new table need to be sorted unless an index is used for accessing these tables that gives the correct order already. The sorts needed for join processing are indicated by SORTN_JOIN and SORTC_JOIN in the PLAN_TABLE.

Sorts needed for subquery processing

When a noncorrelated IN or NOT IN subquery is present in the query, the results of the subquery are sorted and put into a work file for later reference by the parent query. The results of the subquery are sorted because this allows the parent query to be more efficient when processing the IN or NOT IN predicate. Duplicates are not needed in the work file, and are removed. Noncorrelated subqueries used with =ANY or =ALL, or NOT=ANY or NOT=ALL also need the same type of sort as IN or NOT IN subqueries. When a sort for a noncorrelated subquery is performed, you see both SORTC_ORDERBY and SORTC_UNIQUE in PLAN_TABLE. This is because DB2 removes the duplicates and performs the sort.

SORTN_GROUPBY, SORTN_ORDERBY, and SORTN_UNIQ are not currently used by DB2.

Sorts of RIDs

To perform list prefetch, DB2 sorts RIDs into ascending page number order. This sort is very fast and is done totally in memory. A RID sort is usually not indicated in the PLAN_TABLE, but a RID sort normally is performed whenever list prefetch is used. The only exception to this rule is when a hybrid join is performed and a single, highly clustered index is used on the inner table. In this case SORTN_JOIN is 'N', indicating that the RID list for the inner table was not sorted.

The effect of sorts on OPEN CURSOR

The type of sort processing required by the cursor affects the amount of time it can take for DB2 to process the OPEN CURSOR statement. This section outlines the effect of sorts and parallelism on OPEN CURSOR.

Without parallelism:

- If no sorts are required, then OPEN CURSOR does not access any data. It is at the first fetch that data is returned.
- If a sort is required, then the OPEN CURSOR causes the materialized result table to be produced. Control returns to the application after the result table is materialized. If a cursor that requires a sort is closed and reopened, the sort is performed again.
- If there is a RID sort, but no data sort, then it is not until the first row is fetched that the RID list is built from the index and the first data record is returned. Subsequent fetches access the RID pool to access the next data record.

With parallelism:

- At OPEN CURSOR, parallelism is asynchronously started, regardless of whether a sort is required. Control returns to the application immediately after the parallelism work is started.
- If there is a RID sort, but no data sort, then parallelism is not started until the first fetch. This works the same way as with no parallelism.

Processing for views and nested table expressions

This section describes how DB2 processes views and nested table expressions. A nested table expression (which is called *table expression* in this description) is the specification of a subquery in the FROM clause of an SQL SELECT statement. The processing of table expressions is similar to a view. Two methods are used to satisfy your queries that reference views or table expressions:

- "Merge"
- "Materialization" on page 976

You can determine the methods that are used by executing EXPLAIN for the statement that contains the view or nested table expression. In addition, you can use EXPLAIN to determine when UNION operators are used and how DB2 might eliminate unnecessary subselects to improve the performance of a query.

Merge

The merge process is more efficient than materialization, as described in "Performance of merge versus materialization" on page 981. In the merge process, the statement that references the view or table expression is combined with the fullselect that defined the view or table expression. This combination creates a logically equivalent statement. This equivalent statement is executed against the database.

Example: Consider the following statements, one of which defines a view, the other of which references the view:

```
View-defining statement:                View referencing statement:

CREATE VIEW VIEW1 (VC1,VC21,VC32) AS     SELECT VC1,VC21
  SELECT C1,C2,C3 FROM T1                  FROM VIEW1
    WHERE C1 > C3;                         WHERE VC1 IN (A,B,C);
```

The fullselect of the view-defining statement can be merged with the view-referencing statement to yield the following logically equivalent statement:

```
Merged statement:

SELECT C1,C2 FROM T1
  WHERE C1 > C3 AND C1 IN (A,B,C);
```

Example: The following statements show another example of when a view and table expression can be merged:

```
SELECT * FROM V1 X
  LEFT JOIN
    (SELECT * FROM T2) Y ON X.C1=Y.C1
       LEFT JOIN T3 Z ON X.C1=Z.C1;
```

```
Merged statement:

SELECT * FROM V1 X
  LEFT JOIN
    T2 ON X.C1 = T2.C1
      LEFT JOIN T3 Z ON X.C1 = Z.C1;
```

Materialization

Views and table expressions cannot always be merged.

Example: Look at the following statements:

```
View defining statement:                View referencing statement:

CREATE VIEW VIEW1 (VC1,VC2) AS          SELECT MAX(VC1)
  SELECT SUM(C1),C2 FROM T1               FROM VIEW1;
    GROUP BY C2;
```

Column VC1 occurs as the argument of a aggregate function in the view referencing statement. The values of VC1, as defined by the view-defining fullselect, are the result of applying the aggregate function SUM(C1) to groups after grouping the base table T1 by column C2. No equivalent single SQL SELECT statement can be executed against the base table T1 to achieve the intended result. There is no way to specify that aggregate functions should be applied successively.

Two steps of materialization

In the previous example, DB2 performs materialization of the view or table expression, which is a two step process.

1. The fullselect that defines the view or table expression is executed against the database, and the results are placed in a temporary copy of a result table.
2. The statement that references the view or table expression is then executed against the temporary copy of the result table to obtain the intended result.

Whether materialization is needed depends upon the attributes of the referencing statement, or logically equivalent referencing statement from a prior merge, and the attributes of the fullselect that defines the view or table expression.

When views or table expressions are materialized

In general, DB2 uses materialization to satisfy a reference to a view or table expression when there is aggregate processing (grouping, aggregate functions, distinct), indicated by the defining fullselect, in conjunction with either aggregate processing indicated by the statement referencing the view or table expression, or by the view or table expression participating in a join. For views and table expressions that are defined with UNION or UNION ALL, DB2 can often distribute aggregate processing, joins, and qualified predicates to avoid materialization. For more information, see "Using EXPLAIN to determine UNION activity and query rewrite" on page 979.

Table 186 indicates some cases in which materialization occurs. DB2 can also use materialization in statements that contain multiple outer joins, outer joins that combine with inner joins, or merges that cause a join of greater than 15 tables.

Table 186. Cases when DB2 performs view or table expression materialization. The "X" indicates a case of materialization. Notes follow the table.

	View definition or table expression uses...[2]					
SELECT FROM view or table expression uses...[1]	GROUP BY	DISTINCT	Aggregate function	Aggregate function DISTINCT	UNION	UNION ALL(4)
Joins (3)	X	X	X	X	X	
GROUP BY	X	X	X	X	X	
DISTINCT		X		X	X	

Table 186. Cases when DB2 performs view or table expression materialization (continued). The "X" indicates a case of materialization. Notes follow the table.

	View definition or table expression uses...[2]					
SELECT FROM view or table expression uses...[1]	GROUP BY	DISTINCT	Aggregate function	Aggregate function DISTINCT	UNION	UNION ALL(4)
Aggregate function	X	X	X	X	X	X
Aggregate function DISTINCT	X	X	X	X	X	
SELECT subset of view or table expression columns		X			X	

Notes to Table 186 on page 976:

1. If the view is referenced as the target of an INSERT, UPDATE, or DELETE, then view merge is used to satisfy the view reference. Only updatable views can be the target in these statements. See Chapter 5 of *DB2 SQL Reference* for information about which views are read-only (not updatable).

   An SQL statement can reference a particular view multiple times where some of the references can be merged and some must be materialized.
2. If a SELECT list contains a host variable in a table expression, then materialization occurs. For example:

   ```
   SELECT C1 FROM
      (SELECT :HV1 AS C1 FROM T1) X;
   ```

   If a view or nested table expression is defined to contain a user-defined function, and if that user-defined function is defined as NOT DETERMINISTIC or EXTERNAL ACTION, then the view or nested table expression is always materialized.
3. Additional details about materialization with outer joins:
   - If a WHERE clause exists in a view or table expression, and it does not contain a column, materialization occurs.

     Example:

     ```
     SELECT X.C1 FROM
        (SELECT C1 FROM T1
          WHERE 1=1) X LEFT JOIN T2 Y
                       ON X.C1=Y.C1;
     ```
   - If the outer join is a full outer join and the SELECT list of the view or nested table expression does not contain a standalone column for the column that is used in the outer join ON clause, then materialization occurs.

     Example:

     ```
     SELECT X.C1 FROM
        (SELECT C1+10 AS C2 FROM T1) X FULL JOIN T2 Y
                       ON X.C2=Y.C2;
     ```
   - If there is no column in a SELECT list of a view or nested table expression, materialization occurs.

     Example:

     ```
     SELECT X.C1 FROM
        (SELECT 1+2+:HV1. AS C1 FROM T1) X LEFT JOIN T2 Y
                       ON X.C1=Y.C1;
     ```

- If the SELECT list of a view or nested table expression contains a CASE expression, and the result of the CASE expression is referenced in the outer query block, then materialization occurs.

  Example:

  ```
  SELECT X.C1  FROM
         T1 X LEFT JOIN
          (SELECT CASE C2 WHEN 5 THEN 10 ELSE 20 END AS YC1 FROM T2) Y
                   ON X.C1 = Y.YC1;
  ```

4. DB2 cannot avoid materialization for UNION ALL in all cases. Some of the situations in which materialization occurs includes:
   - When the view is the operand in an outer join for which nulls are used for non-matching values, materialization occurs. This situation happens when the view is either operand in a full outer join, the right operand in a left outer join, or the left operand in a right outer join.
   - If the number of tables would exceed 225 after distribution, then distribution will not occur, and the result will be materialized.

Using EXPLAIN to determine when materialization occurs

For each reference to a view or table expression that is materialized, rows describing the access path for both steps of the materialization process appear in the PLAN_TABLE. These rows describe the access path used to formulate the temporary result indicated by the view's defining fullselect, and they describe the access to the temporary result as indicated by the referencing statement. The defining fullselect can also refer to views or table expressions that need to be materialized.

When DB2 chooses materialization, TNAME contains the name of the view or table expression, and TABLE_TYPE contains a W. A value of Q in TABLE_TYPE for the name of a view or nested table expression indicates that the materialization was virtual and not actual. (Materialization can be virtual when the view or nested table expression definition contains a UNION ALL that is not distributed.) When DB2 chooses merge, EXPLAIN data for the merged statement appears in PLAN_TABLE; only the names of the base tables on which the view or table expression is defined appear.

Example: Consider the following statements, which define a view and reference the view:

```
View defining statement:

CREATE VIEW V1DIS (SALARY, WORKDEPT) as
  (SELECT DISTINCT SALARY, WORKDEPT FROM DSN8810.EMP)

View referencing statement:

SELECT * FROM DSN8810.DEPT
  WHERE DEPTNO IN (SELECT WORKDEPT FROM V1DIS)
```

Table 187 shows a subset of columns in a plan table for the query.

Table 187. Plan table output for an example with view materialization

QBLOCKNO	PLANNO	QBLOCK_ TYPE	TNAME	TABLE_ TYPE	METHOD
1	1	SELECT	DEPT	T	0
2	1	NOCOSUB	V1DIS	W	0
2	2	NOCOSUB		?	3

Table 187. Plan table output for an example with view materialization (continued)

QBLOCKNO	PLANNO	QBLOCK_ TYPE	TNAME	TABLE_ TYPE	METHOD
3	1	NOCOSUB	EMP	T	0
3	2	NOCOSUB		?	3

Notice how TNAME contains the name of the view and TABLE_TYPE contains W to indicate that DB2 chooses materialization for the reference to the view because of the use of SELECT DISTINCT in the view definition.

Example: Consider the following statements, which define a view and reference the view:

```
View defining statement:

CREATE VIEW V1NODIS (SALARY, WORKDEPT) as
  (SELECT SALARY, WORKDEPT FROM DSN8810.EMP)

View referencing statement:

SELECT * FROM DSN8810.DEPT
   WHERE DEPTNO IN (SELECT WORKDEPT FROM V1NODIS)
```

If the VIEW was defined without DISTINCT, DB2 would choose merge instead of materialization. In the sample output, the name of the view does not appear in the plan table, but the table name on which the view is based does appear.

Table 188 shows a sample plan table for the query.

Table 188. Plan table output for an example with view merge

QBLOCKNO	PLANNO	QBLOCK_ TYPE	TNAME	TABLE_ TYPE	METHOD
1	1	SELECT	DEPT	T	0
2	1	NOCOSUB	EMP	T	0
2	2	NOCOSUB		?	3

For an example of when a view definition contains a UNION ALL and DB2 can distribute joins and aggregations and avoid materialization, see "Using EXPLAIN to determine UNION activity and query rewrite." When DB2 avoids materialization in such cases, TABLE_TYPE contains a Q to indicate that DB2 uses an intermediate result that is not materialized, and TNAME shows the name of this intermediate result as DSNWFQB(*xx*), where *xx* is the number of the query block that produced the result.

Using EXPLAIN to determine UNION activity and query rewrite

For each reference to a view or table expression that is defined with UNION or UNION ALL operators, DB2 tries to rewrite the query into a logically equivalent statement with improved performance by:

- Distributing qualified predicates, joins, and aggregations across the subselects of UNION ALL. Such distribution helps to avoid materialization. No distribution is performed for UNION.
- Eliminating unnecessary subselects of the view or table expression. For DB2 to eliminate subselects, the referencing query and the view or table definition must have predicates that are based on common columns.

The QBLOCK_TYPE column in the plan table indicates union activity. For a UNION ALL, the column contains 'UNIONA'. For UNION, the column contains 'UNION'. When QBLOCK_TYPE='UNION', the METHOD column on the same row is set to 3 and the SORTC_UNIQ column is set to 'Y' to indicate that a sort is necessary to remove duplicates. As with other views and table expressions, the plan table also shows when DB2 uses materialization instead of merge.

Example: Consider the following statements, which define a view, reference the view, and show how DB2 rewrites the referencing statement:

```
View defining statement: View is created on three tables that contain weekly data

CREATE VIEW V1 (CUSTNO, CHARGES, DATE) as
  SELECT CUSTNO, CHARGES, DATE
  FROM WEEK1
  WHERE DATE BETWEEN '01/01/2000' And '01/07/2000'
UNION ALL
 SELECT CUSTNO, CHARGES, DATE
  FROM WEEK2
  WHERE DATE BETWEEN '01/08/2000' And '01/14/2000'
UNION ALL
  SELECT CUSTNO, CHARGES, DATE
  FROM WEEK3
  WHERE DATE BETWEEN '01/15/2000' And '01/21/2000';

View referencing statement: For each customer in California, find the average
charges during the first and third Friday of January 2000

SELECT V1.CUSTNO, AVG(V1.CHARGES)
  FROM CUST, V1
  WHERE CUST.CUSTNO=V1.CUSTNO
    AND CUST.STATE='CA'
    AND DATE IN ('01/07/2000','01/21/2000')
  GROUP BY V1.CUSTNO;

Rewritten statement (assuming that CHARGES is defined as NOT NULL):

SELECT CUSTNO_U, SUM(SUM_U)/SUM(CNT_U)
  FROM
  ( SELECT WEEK1.CUSTNO, SUM(CHARGES), COUNT(CHARGES)
      FROM CUST, WEEK1
      Where CUST.CUSTNO=WEEK1.CUSTNO AND CUST.STATE='CA'
            AND DATE BETWEEN '01/01/2000' And '01/07/2000'
            AND DATE IN ('01/07/2000','01/21/2000')
      GROUP BY WEEK1.CUSTNO
    UNION ALL
    SELECT WEEK3.CUSNTO, SUM(CHARGES), COUNT(CHARGES)
      FROM CUST,WEEK3
      WHERE CUST.CUSTNO=WEEK3 AND CUST.STATE='CA'
            AND DATE BETWEEN '01/15/2000' And '01/21/2000'
            AND DATE IN ('01/07/2000','01/21/2000')
      GROUP BY WEEK3.CUSTNO
  ) AS X(CUSTNO_U,SUM_U,CNT_U)
GROUP BY CUSNTO_U;
```

Table 189 shows a subset of columns in a plan table for the query.

Table 189. Plan table output for an example with a view with UNION ALLs

QBLOCKNO	PLANNO	TNAME	TABLE_TYPE	METHOD	QBLOCK_ TYPE	PARENT_ QBLOCKNO
1	1	DSNWFQB(02)	Q	0		0
1	2		?	3		0
2	1		?	0	UNIONA	1
3	1	CUST	T	0		2

Table 189. Plan table output for an example with a view with UNION ALLs (continued)

QBLOCKNO	PLANNO	TNAME	TABLE_TYPE	METHOD	QBLOCK_ TYPE	PARENT_ QBLOCKNO
3	2	WEEK1	T	1		2
4	1	CUST	T	0		2
4	2	WEEK3	T	2		2

Notice how DB2 eliminates the second subselect of the view definition from the rewritten query and how the plan table indicates this removal by showing a UNION ALL for only the first and third subselect in the view definition. The Q in the TABLE_TYPE column indicates that DB2 does not materialize the view.

Performance of merge versus materialization

Merge performs better than materialization. For materialization, DB2 uses a table space scan to access the materialized temporary result. DB2 materializes a view or table expression only if it cannot merge.

Materialization is a two-step process with the first step resulting in the formation of a temporary result. The smaller the temporary result, the more efficient is the second step. To reduce the size of the temporary result, DB2 attempts to evaluate certain predicates from the WHERE clause of the referencing statement at the first step of the process rather than at the second step. Only certain types of predicates qualify. First, the predicate must be a simple Boolean term predicate. Second, it must have one of the forms shown in Table 190.

Table 190. Predicate candidates for first-step evaluation

Predicate	Example
COL op literal	V1.C1 > hv1
COL IS (NOT) NULL	V1.C1 IS NOT NULL
COL (NOT) BETWEEN literal AND literal	V1.C1 BETWEEN 1 AND 10
COL (NOT) LIKE constant (ESCAPE constant)	V1.C2 LIKE 'p\%%' ESCAPE '\'
COL IN (list)	VI.C2 IN (a,b,c)

Note: Where "op" is =, <>, >, <, <=, or >=, and literal is either a host variable, constant, or special register. The literals in the BETWEEN predicate need not be identical.

Implied predicates generated through predicate transitive closure are also considered for first step evaluation.

Estimating a statement's cost

You can use EXPLAIN to populate a statement table, *owner*.DSN_STATEMNT_TABLE, at the same time as your PLAN_TABLE is being populated. DB2 provides cost estimates, in service units and in milliseconds, for SELECT, INSERT, UPDATE, and DELETE statements, both static and dynamic. The estimates do not take into account several factors, including cost adjustments that are caused by parallel processing, or the use of triggers or user-defined functions.

Use the information provided in the statement table to:

- Help you determine if a statement is not going to perform within range of your service-level agreements and to tune accordingly.

  DB2 puts its cost estimate into one of two *cost categories*: category A or category B. Estimates that go into cost category A are the ones for which DB2 has

adequate information to make an estimate. That estimate is not likely to be 100% accurate, but is likely to be more accurate than any estimate that is in cost category B.

DB2 puts estimates into cost category B when it is forced to use default values for its estimates, such as when no statistics are available, or because host variables are used in a query. See the description of the REASON column in Table 191 on page 983 for more information about how DB2 determines into which cost category an estimate goes.

- Give a system programmer a basis for entering service-unit values by which to govern dynamic statements.

  Information about using predictive governing is in "Predictive governing" on page 724.

This section describes the following tasks to obtain and use cost estimate information from EXPLAIN:

1. "Creating a statement table"
2. "Populating and maintaining a statement table" on page 984
3. "Retrieving rows from a statement table" on page 984
4. "The implications of cost categories" on page 985

See Part 6 of *DB2 Application Programming and SQL Guide* for more information about how to change applications to handle the SQLCODES that are associated with predictive governing.

Creating a statement table

To collect information about the estimated cost of a statement, create a table called DSN_STATEMNT_TABLE to hold the results of EXPLAIN. A copy of the statements that are needed to create the table are in the DB2 sample library, under the member name DSNTESC.

Figure 109 shows the current format of a statement table.

```
CREATE TABLE DSN_STATEMNT_TABLE
  ( QUERYNO            INTEGER       NOT NULL WITH DEFAULT,
    APPLNAME           CHAR(8)       NOT NULL WITH DEFAULT,
    PROGNAME           VARCHAR(128)  NOT NULL WITH DEFAULT,
    COLLID             VARCHAR(128)  NOT NULL WITH DEFAULT,
    GROUP_MEMBER       CHAR(8)       NOT NULL WITH DEFAULT,
    EXPLAIN_TIME       TIMESTAMP     NOT NULL WITH DEFAULT,
    STMT_TYPE          CHAR(6)       NOT NULL WITH DEFAULT,
    COST_CATEGORY      CHAR(1)       NOT NULL WITH DEFAULT,
    PROCMS             INTEGER       NOT NULL WITH DEFAULT,
    PROCSU             INTEGER       NOT NULL WITH DEFAULT,
    REASON             VARCHAR(254)  NOT NULL WITH DEFAULT
    STMT_ENCODE        CHAR(1)       NOT NULL WITH DEFAULT);
```

Figure 109. Current format of DSN_STATEMNT_TABLE

Your statement table can use an older format in which the STMT_ENCODE column does not exist, PROGNAME has a data type of CHAR(8), and COLLID has a data type of CHAR(18). However, use the most current format because it gives you the most information. You can alter a statement table in the older format to a statement table in the current format.

Table 191 on page 983 shows the content of each column.

Table 191. Descriptions of columns in DSN_STATEMNT_TABLE

Column Name	Description
QUERYNO	A number that identifies the statement being explained. See the description of the QUERYNO column in Table 165 on page 932 for more information. If QUERYNO is not unique, the value of EXPLAIN_TIME is unique.
APPLNAME	The name of the application plan for the row, or blank. See the description of the APPLNAME column in Table 165 on page 932 for more information.
PROGNAME	The name of the program or package containing the statement being explained, or blank. See the description of the PROGNAME column in Table 165 on page 932 for more information.
COLLID	The collection ID for the package. Applies only to an embedded EXPLAIN statement executed from a package or to a statement that is explained when binding a package. Blank if not applicable. The value DSNDYNAMICSQLCACHE indicates that the row is for a cached statement.
GROUP_MEMBER	The member name of the DB2 that executed EXPLAIN, or blank. See the description of the GROUP_MEMBER column in Table 165 on page 932 for more information.
EXPLAIN_TIME	The time at which the statement is processed. This time is the same as the BIND_TIME column in PLAN_TABLE.
STMT_TYPE	The type of statement being explained. Possible values are: **SELECT** SELECT **INSERT** INSERT **UPDATE** UPDATE **DELETE** DELETE **SELUPD** SELECT with FOR UPDATE OF **DELCUR** DELETE WHERE CURRENT OF CURSOR **UPDCUR** UPDATE WHERE CURRENT OF CURSOR
COST_CATEGORY	Indicates if DB2 was forced to use default values when making its estimates. Possible values: **A** Indicates that DB2 had enough information to make a cost estimate without using default values. **B** Indicates that some condition exists for which DB2 was forced to use default values. See the values in REASON to determine why DB2 was unable to put this estimate in cost category A.
PROCMS	The estimated processor cost, in milliseconds, for the SQL statement. The estimate is rounded up to the next integer value. The maximum value for this cost is 2147483647 milliseconds, which is equivalent to approximately 24.8 days. If the estimated value exceeds this maximum, the maximum value is reported.
PROCSU	The estimated processor cost, in service units, for the SQL statement. The estimate is rounded up to the next integer value. The maximum value for this cost is 2147483647 service units. If the estimated value exceeds this maximum, the maximum value is reported.

Table 191. Descriptions of columns in DSN_STATEMNT_TABLE (continued)

Column Name	Description	
REASON	A string that indicates the reasons for putting an estimate into cost category B.	
	HAVING CLAUSE	A subselect in the SQL statement contains a HAVING clause.
	HOST VARIABLES	The statement uses host variables, parameter markers, or special registers.
	REFERENTIAL CONSTRAINTS	Referential constraints of the type CASCADE or SET NULL exist on the target table of a DELETE statement.
	TABLE CARDINALITY	The cardinality statistics are missing for one or more of the tables that are used in the statement. Or, the statement required the materialization of views or nested table expressions.
	TRIGGERS	Triggers are defined on the target table of an INSERT, UPDATE, or DELETE statement.
	UDF	The statement uses user-defined functions.
STMT_ENCODE	Encoding scheme of the statement. If the statement repesents a single CCSID set, the possible values are: **A** ASCII **E** EBCDIC **U** Unicode If the statement has multiple CCSID sets, the value is M.	

Populating and maintaining a statement table

You populate a statement table at the same time as you populate the corresponding plan table. For more information, see "Populating and maintaining a plan table" on page 936.

Just as with the plan table, DB2 just adds rows to the statement table; it does not automatically delete rows. INSERT triggers are not activated unless you insert rows yourself using and SQL INSERT statement.

To clear the table of obsolete rows, use DELETE, just as you would for deleting rows from any table. You can also use DROP TABLE to drop a statement table completely.

Retrieving rows from a statement table

To retrieve all rows in a statement table, you can use a query like the following statement, which retrieves all rows about the statement that is represented by query number 13:

```
SELECT * FROM JOE.DSN_STATEMNT_TABLE
   WHERE QUERYNO = 13;
```

The QUERYNO, APPLNAME, PROGNAME, COLLID, and EXPLAIN_TIME columns contain the same values as corresponding columns of PLAN_TABLE for a given plan. You can use these columns to join the plan table and statement table:

```
SELECT A.*, PROCMS, COST_CATEGORY
 FROM JOE.PLAN_TABLE A, JOE.DSN_STATEMNT_TABLE B
   WHERE A.APPLNAME = 'APPL1' AND
   A.APPLNAME = B.APPLNAME AND
```

```
#                    A.QUERYNO  = B.QUERYNO AND   A.PROGNAME = B.PROGNAME AND
                     A.COLLID   = B.COLLID AND
                     A.BIND_TIME = B.EXPLAIN_TIME
                ORDER BY A.QUERYNO, A.QBLOCKNO, A.PLANNO, A.MIXOPSEQ;
```

The implications of cost categories

Cost categories are DB2's way of differentiating estimates for which adequate information is available from those for which it is not. You probably wouldn't want to spend a lot of time tuning a query based on estimates that are returned in cost category B, because the actual cost could be radically different based on such things as what value is in a host variable, or how many levels of nested triggers and user-defined functions exist.

Similarly, if system administrators use these estimates as input into the resource limit specification table for governing (either predictive or reactive), they probably would want to give much greater latitude for statements in cost category B than for those in cost category A.

Because of the uncertainty involved, category B statements are also good candidates for reactive governing.

What goes into cost category B? DB2 puts a statement's estimate into cost category B when any of the following conditions exist:

- The statement has UDFs.
- Triggers are defined for the target table:
  - The statement is INSERT, and insert triggers are defined on the target table.
  - The statement is UPDATE, and update triggers are defined on the target table.
  - The statement is DELETE, and delete triggers are defined on the target table.
- The target table of a delete statement has referential constraints defined on it as the parent table, and the delete rules are either CASCADE or SET NULL.
- The WHERE clause predicate has one of the following forms:
  - COL op literal, and the literal is a host variable, parameter marker, or special register. The operator can be >, >=, <, <=, LIKE, or NOT LIKE.
  - COL BETWEEN literal AND literal where either literal is a host variable, parameter marker, or special register.
  - LIKE with an escape clause that contains a host variable.
- The cardinality statistics are missing for one or more tables that are used in the statement.
- A subselect in the SQL statement contains a HAVING clause.

What goes into cost category A? DB2 puts everything that doesn't fall into category B into category A.

Chapter 35. Parallel operations and query performance

When DB2 plans to access data from a table or index in a partitioned table space, it can initiate multiple parallel operations. The response time for data or processor-intensive queries can be significantly reduced.

Query I/O parallelism manages concurrent I/O requests for a single query, fetching pages into the buffer pool in parallel. This processing can significantly improve the performance of I/O-bound queries. I/O parallelism is used only when one of the other parallelism modes cannot be used.

Query CP parallelism enables true multitasking within a query. A large query can be broken into multiple smaller queries. These smaller queries run simultaneously on multiple processors accessing data in parallel. This reduces the elapsed time for a query.

To expand even farther the processing capacity available for processor-intensive queries, DB2 can split a large query across different DB2 members in a data sharing group. This is known as Sysplex query parallelism. For more information about Sysplex query parallelism, see Chapter 6 of *DB2 Data Sharing: Planning and Administration*.

DB2 can use parallel operations for processing:
- Static and dynamic queries
- Local and remote data access
- Queries using single table scans and multi-table joins
- Access through an index, by table space scan or by list prefetch
- Sort operations

Parallel operations usually involve at least one table in a partitioned table space. Scans of large partitioned table spaces have the greatest performance improvements where both I/O and central processor (CP) operations can be carried out in parallel.

Parallelism for partitioned and nonpartitioned table spaces: Both partitioned and nonpartitioned table spaces can take advantage of query parallelism. Parallelism is now enabled to include non-clustering indexes. Thus, table access can be run in parallel when the application is bound with DEGREE (ANY) and the table is accessed through a non-clustering index.

This chapter contains the following topics:

- "Comparing the methods of parallelism" on page 988
- "Partitioning for optimal parallel performance" on page 990
- "Enabling parallel processing" on page 993
- "When parallelism is not used" on page 994
- "Interpreting EXPLAIN output" on page 995
- "Monitoring parallel operations" on page 997
- "Tuning parallel processing" on page 1000
- "Disabling query parallelism" on page 1001

© Copyright IBM Corp. 1982, 2008

Comparing the methods of parallelism

The figures in this section show how the parallel methods compare with sequential prefetch and with each other. All three techniques assume access to a table space with three partitions, P1, P2, and P3. The notations P1, P2, and P3 are partitions of a table space. R1, R2, R3, and so on, are requests for sequential prefetch. The combination P2R1, for example, means the first request from partition 2.

Figure 110 shows **sequential processing**. With sequential processing, DB2 takes the 3 partitions in order, completing partition 1 before starting to process partition 2, and completing 2 before starting 3. Sequential prefetch allows overlap of CP processing with I/O operations, but I/O operations do not overlap with each other. In the example in Figure 110, a prefetch request takes longer than the time to process it. The processor is frequently waiting for I/O.

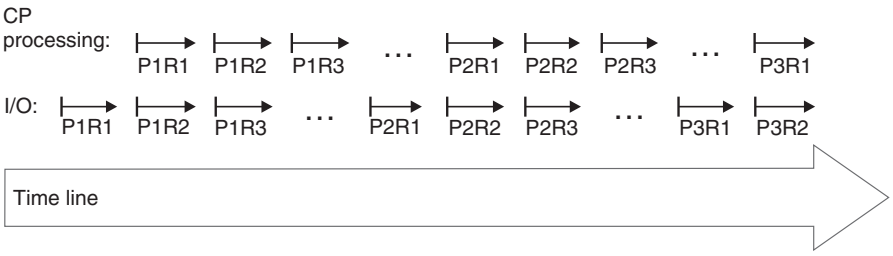


Figure 110. CP and I/O processing techniques. Sequential processing.

Figure 111 shows **parallel I/O operations**. With parallel I/O, DB2 prefetches data from the 3 partitions at one time. The processor processes the first request from each partition, then the second request from each partition, and so on. The processor is not waiting for I/O, but there is still only one processing task.

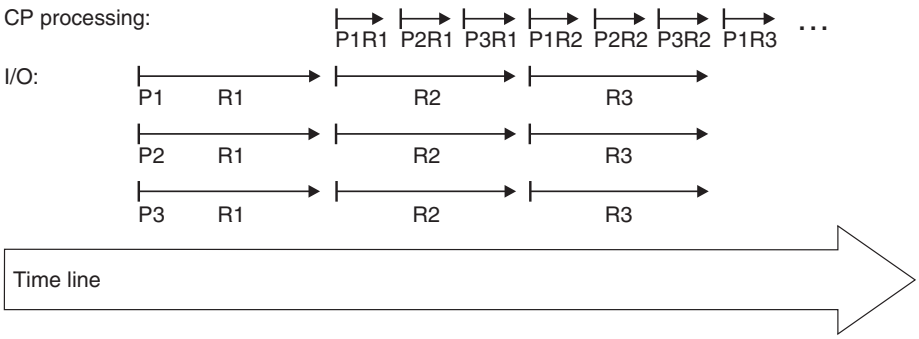


Figure 111. CP and I/O processing techniques. Parallel I/O processing.

Figure 112 on page 989 shows **parallel CP processing**. With CP parallelism, DB2 can use multiple parallel tasks to process the query. Three tasks working concurrently can greatly reduce the overall elapsed time for data-intensive and processor-intensive queries. The same principle applies for **Sysplex query parallelism**, except that the work can cross the boundaries of a single CPC.

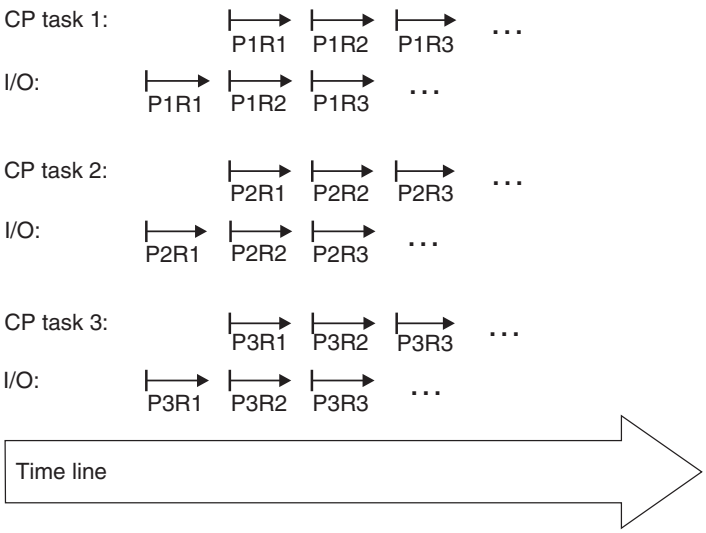


Figure 112. CP and I/O processing techniques. Query processing using CP parallelism. The tasks can be contained within a single CPC or can be spread out among the members of a data sharing group.

Queries that are most likely to take advantage of parallel operations: Queries that can take advantage of parallel processing are:

- Those in which DB2 spends most of the time fetching pages—an I/O-intensive query

  A typical I/O-intensive query is something like the following query, assuming that a table space scan is used on many pages:

  ```
  SELECT COUNT(*) FROM ACCOUNTS
   WHERE BALANCE > 0 AND
   DAYS_OVERDUE > 30;
  ```

- Those in which DB2 spends a lot of processor time and also, perhaps, I/O time, to process rows. Those include:
  - *Queries with intensive data scans and high selectivity.* Those queries involve large volumes of data to be scanned but relatively few rows that meet the search criteria.
  - *Queries containing aggregate functions.* Column functions (such as MIN, MAX, SUM, AVG, and COUNT) usually involve large amounts of data to be scanned but return only a single aggregate result.
  - *Queries accessing long data rows.* Those queries access tables with long data rows, and the ratio of rows per page is very low (one row per page, for example).
  - *Queries requiring large amounts of central processor time.* Those queries might be read-only queries that are complex, data-intensive, or that involve a sort.

    A typical processor-intensive query is something like:

    ```
    SELECT MAX(QTY_ON_HAND) AS MAX_ON_HAND,
      AVG(PRICE) AS AVG_PRICE,
      AVG(DISCOUNTED_PRICE) AS DISC_PRICE,
      SUM(TAX) AS SUM_TAX,
      SUM(QTY_SOLD) AS SUM_QTY_SOLD,
      SUM(QTY_ON_HAND - QTY_BROKEN) AS QTY_GOOD,
      AVG(DISCOUNT) AS AVG_DISCOUNT,
      ORDERSTATUS,
      COUNT(*) AS COUNT_ORDERS
    FROM   ORDER_TABLE
    ```

```
WHERE SHIPPER = 'OVERNIGHT' AND
      SHIP_DATE < DATE('1996-01-01')
GROUP BY ORDERSTATUS
ORDER BY ORDERSTATUS;
```

Terminology: When the term **task** is used with information about parallel processing, the context should be considered. For parallel query CP processing or Sysplex query parallelism, a task is an actual z/OS execution unit used to process a query. For parallel I/O processing, a task simply refers to the processing of one of the concurrent I/O streams.

A **parallel group** is the term used to name a particular set of parallel operations (parallel tasks or parallel I/O operations). A query can have more than one parallel group, but each parallel group within the query is identified by its own unique ID number.

The **degree of parallelism** is the number of parallel tasks or I/O operations that DB2 determines can be used for the operations on the parallel group. The maximum of parallel operations that DB2 can generate is 254. However, for most queries and DB2 environments, DB2 chooses a lower number. You might need to limit the maximum number further because more parallel operations consume processor, real storage, and I/O resources. If resource consumption in high in your parallelism environment, use the MAX DEGREE field on installation panel DSNTIP4 to explicitly limit the maximum number of parallel operations that DB2 generates, as explain in "Enabling parallel processing" on page 993.

In a parallel group, an **originating task** is the TCB (SRB for distributed requests) that coordinates the work of all the **parallel tasks**. Parallel tasks are executable units composed of special SRBs, which are called **preemptable** SRBs.

With preemptable SRBs, the z/OS dispatcher can interrupt a task at any time to run other work at the same or higher dispatching priority. For non-distributed parallel work, parallel tasks run under a type of preemptable SRB called a **client** SRB, which lets the parallel task inherit the importance of the originating address space. For distributed requests, the parallel tasks run under a preemptable SRB called an **enclave** SRB. Enclave SRBs are described more fully in "Using z/OS workload management to set performance objectives" on page 739.

Partitioning for optimal parallel performance

This section includes some general considerations for how to partition data for the best performance when using parallel processing. Bear in mind that DB2 does not always choose parallelism, even if you partition the data.

This exercise assumes the following:

- You have narrowed the focus to a few, critical queries that are running sequentially. It is best to include a mix of I/O-intensive and processor-intensive queries into this initial set. You know how long those queries take now and what your performance objectives for those queries are. Although tuning for one set of queries might not work for all queries, overall performance and throughput can be improved.
- You are optimizing for query-at-a-time operations, and you want a query to make use of all the processor and I/O resources available to it.

  When running many queries at the same time, you will probably have to increase the number of partitions and the amount of processing power to achieve similar elapsed times.

This section guides you through the following analyses:

1. Determining the nature of the query (what balance of processing and I/O resources it needs)
2. Determining how many partitions the table space should have to meet your performance objective, number based on the nature of the query and on the processor and I/O configuration at your site

Determining if a query is I/O- or processor-intensive

To determine if your sequential queries are I/O or processor-intensive, examine the DB2 accounting reports:

- If the "other read I/O time" is close to the total query elapsed time, then the query is I/O-intensive. "Other read I/O time" is the time that DB2 is waiting for pages to be read in to the buffer pools.
- If "CPU time" is close to the total query elapsed time, then the query is processor-intensive.
- If the processor time is somewhere between 30 and 70 percent of the elapsed time, then the query is pretty well-balanced.

Determining the number of partitions

This section is intended to give you some general guidance on determining the number of partitions. Again, you must take into account the I/O subsystem, the nature of the queries you run, and, if necessary, plan for the data to grow. If your physical and logical design are not closely tied together, thus allowing you to specify any number of partitions, it does no harm to specify more partitions than you need immediately. However, start with a reasonable number of partitions because you can always add more partitions later with the ALTER TABLESPACE statement.

Consider also the operational complexity of managing many partitions. This complexity may not be as much of an issue at sites that use tools, such as the DB2 Automated Utilities Generator and job schedulers.

In general, the number of partitions falls in a range between the number of CPs and the maximum number of I/O paths to the data. When determining the number of partitions that use a mixed set of processor- and I/O-intensive queries, always choose the largest number of partitions in the range you determine.

- **For processor-intensive queries**, specify, at a minimum, a number that is equal to the number of CPs in the system that you want to use for parallelism, whether you have a single CPC or multiple CPCs in a data sharing group. If the query is processor-intensive, it can use all CPs available in the system. If you plan to use Sysplex query parallelism, then choose a number that is close to the total number of CPs (including partial allocation of CPs) that you plan to allocate for decision support processing across the data sharing group. Do not include processing resources that are dedicated to other, higher priority, work. For more information about Sysplex query parallelism, see Chapter 6 of *DB2 Data Sharing: Planning and Administration*.
- **For I/O-intensive queries,** calculate the ratio of elapsed time to processor time. Multiply this ratio by the number of processors allocated for decision support processing. Round up this number to determine how many partitions you can use to the best advantage, assuming that these partitions can be on separate devices and have adequate paths to the data. This calculation also assumes that you have adequate processing power to handle the increase in partitions. (This might not be much of an issue with an extremely I/O-intensive query.)

By partitioning the amount indicated previosuly, the query is brought into balance by reducing the I/O wait time. If the number of partitions is less than the number of CPs available on your system, increase this number close to the number of CPs available. By doing so, other queries that read this same table, but that are more processor-intensive, can take advantage of the additional processing power.

Example: Suppose that you have a 10-way CPC and the calculated number of partitions is five. Instead of limiting the table space to five partitions, use 10, to equal the number of CPs in the CPC.

Example configurations for an I/O-intensive query: If the I/O cost of your queries is about twice as much as the processing cost, the optimal number of partitions when run on a 10-way processor is 20 (2 * number of processors). Figure 113 shows an I/O configuration that minimizes the elapsed time and allows the CPC to run at 100% busy. It assumes the suggested guideline of four devices per control unit and four channels per control unit.[8]

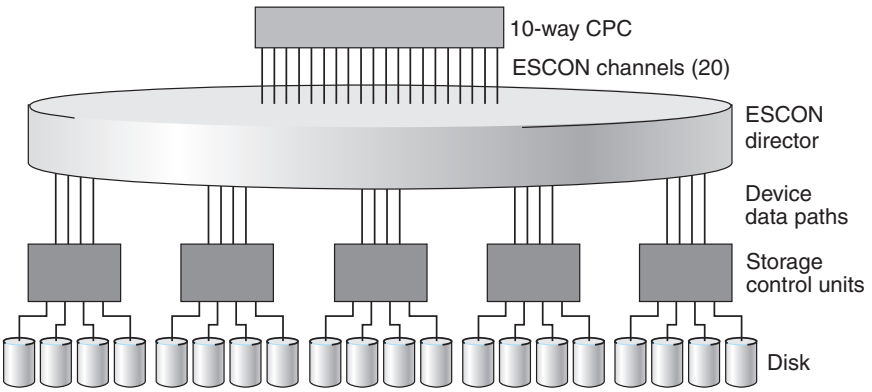


Figure 113. I/O configuration that maximizes performance for an I/O-intensive query

Working with a table space that is already partitioned

Assume that a table space already has 10 partitions and a particular query uses CP parallelism on a 10-way CPC. When you add "other read I/O wait time" (from accounting class 3) and processing time (from accounting class 2) you determine that I/O cost is three times more than the processing cost. In this case, the optimal number of partitions is 30 (three times more I/O paths). However, if you can run on a data sharing group and you add another DB2 to the group that is running on a 10-way CPC, the I/O configuration that minimizes the elapsed time and allows both CPCs to run at 100% would be 60 partitions.

Making the partitions the same size

The degree of parallelism is influenced by the size of the largest physical partition. In most cases, DB2 divides the table space into logical pieces, called *work ranges* to differentiate these from physical pieces, based on the size of the largest physical partition of a given table. Suppose that a table consists of 10 000 pages and 10 physical partitions, the largest of which is 5000 pages. DB2 is most likely to create only two work ranges, and the degree of parallelism would be 2. If the same table has evenly sized partitions of 1000 pages each and the query is I/O-intensive, then

8. A lower-cost configuration could use as few as two to three channels per control unit shared among all controllers using an ESCON director. However, using four paths minimizes contention and provides the best performance. Paths might also need to be taken offline for service.

ten logical work ranges might be created. This example would result in a degree of parallelism of 10 and reduced elapsed time.

DB2 tries to create equal work ranges by dividing the total cost of running the work by the logical partition cost. This division often has some left over work. In this case, DB2 creates an additional task to handle the extra work, rather than making all the work ranges larger, which would reduce the degree of parallelism.

To rebalance partitions that have become skewed, reorganize the table space, specifying the REBALANCE keyword on the REORG utility statement.

Working with partitioned indexes

Similar to partitioned table spaces, the degree of parallelism for accessing partitioned indexes depends on the nature of the query and on the processor and I/O configuration at your site. For an I/O-intensive query, the degree of a parallelism for access to a partitioned index depends on the number of index partitions that are referenced, whereas the degree of parallelism for access to a nonpartitioned index depends on the number of CPs in the system. For a processor-intensive query, the degree of parallelism for both partitioned and nonpartitioned indexes is influenced by the number of CPs in the system.

Enabling parallel processing

Queries can only take advantage of parallelism if you enable parallel processing. Use the following actions to enable parallel processing:

- For **static SQL**, specify DEGREE(ANY) on BIND or REBIND. This bind option affects static SQL only and does not enable parallelism for dynamic statements.
- For **dynamic SQL**, set the CURRENT DEGREE special register to 'ANY'. Setting the special register affects dynamic statements only. It will have no effect on your static SQL statements. You should also make sure that parallelism is not disabled for your plan, package, or authorization ID in the RLST. You can set the special register with the following SQL statement:

  ```
  SET CURRENT DEGREE='ANY';
  ```

  You can also change the special register default from 1 to ANY for the entire DB2 subsystem by modifying the CURRENT DEGREE field on installation panel DSNTIP4.
- If you bind with isolation CS, choose also the option CURRENTDATA(NO), if possible. This option can improve performance in general, but it also ensures that DB2 will consider parallelism for ambiguous cursors. If you bind with CURRENDATA(YES) and DB2 cannot tell if the cursor is read-only, DB2 does not consider parallelism. When a cursor is read-only, it is recommended that you specify the FOR FETCH ONLY or FOR READ ONLY clause on the DECLARE CURSOR statement to explicitly indicate that the cursor is read-only.
- The virtual buffer pool parallel sequential threshold (VPPSEQT) value must be large enough to provide adequate buffer pool space for parallel processing. For more information about VPPSEQT, see “Buffer pool thresholds” on page 667.

If you enable parallel processing when DB2 estimates a given query’s I/O and central processor cost is high, multiple parallel tasks can be activated if DB2 estimates that elapsed time can be reduced by doing so.

Recommendation: For parallel sorts, allocate sufficient work files to maintain performance.

Special requirements for CP parallelism: DB2 must be running on a central processor complex that contains two or more tightly coupled processors (sometimes called central processors, or CPs). If only one CP is online when the query is bound, DB2 considers only parallel I/O operations.

DB2 also considers only parallel I/O operations if you declare a cursor WITH HOLD and bind with isolation RR or RS. For more restrictions on parallelism, see Table 192.

For complex queries, run the query in parallel within a member of a data sharing group. With Sysplex query parallelism, use the power of the data sharing group to process individual complex queries on many members of the data sharing group. For more information about how you can use the power of the data sharing group to run complex queries, see Chapter 6 of *DB2 Data Sharing: Planning and Administration*.

Limiting the degree of parallelism: If you want to limit the maximum number of parallel tasks that DB2 generates, you can use the MAX DEGREE field on installation panel DSNTIP4. Changing MAX DEGREE, however, is not the way to turn parallelism off. You use the DEGREE bind parameter or CURRENT DEGREE special register to turn parallelism off.

When parallelism is not used

Parallelism is not used for all queries; for some access paths, it doesn't make sense to incur parallelism overhead. For example, if you are selecting from a temporary table, parallelism is not used. Check Table 192 to determine whether your query uses any of the access paths that do not allow parallelism.

Table 192. Checklist of parallel modes and query restrictions

	Is parallelism allowed?			
If query uses this...	**I/O**	**CP**	**Sysplex**	**Comments**
Access via RID list (list prefetch and multiple index access)	Yes	Yes	No	Indicated by an "L" in the PREFETCH column of PLAN_TABLE, or an M, MX, MI, or MQ in the ACCESSTYPE column of PLAN_TABLE.
Queries that return LOB values	Yes	Yes	No	
Merge scan join on more than one column	Yes	Yes	Yes	
Queries that qualify for direct row access	No	No	No	Indicated by D in the PRIMARY_ACCESS_TYPE column of PLAN_TABLE
Materialized views or materialized nested table expressions at reference time	No	No	No	
EXISTS within WHERE predicate	No	No	No	
Security label column on table	Yes	Yes	No	

DB2 avoids certain hybrid joins when parallelism is enabled: To ensure that you can take advantage of parallelism, DB2 does not pick one type of hybrid join

(SORTN_JOIN=Y) when the plan or package is bound with CURRENT DEGREE=ANY or if the CURRENT DEGREE special register is set to 'ANY'.

Interpreting EXPLAIN output

To understand how DB2 plans to use parallelism, examine your PLAN_TABLE output. (Details on all columns in PLAN_TABLE are described in Table 165 on page 932.) This section describes a method for examining PLAN_TABLE columns for parallelism and gives several examples.

A method for examining PLAN_TABLE columns for parallelism

The steps for interpreting the output for parallelism are as follows:

1. **Determine if DB2 plans to use parallelism:**

   For each query block (QBLOCKNO) in a query (QUERYNO), a non-null value in ACCESS_DEGREE or JOIN_DEGREE indicates that some degree of parallelism is planned.

2. **Identify the parallel groups in the query:**

   All steps (PLANNO) with the same value for ACCESS_PGROUP_ID, JOIN_PGROUP_ID, SORTN_PGROUP_ID, or SORTC_PGROUP_ID indicate that a set of operations are in the same parallel group. Usually, the set of operations involves various types of join methods and sort operations. Parallel group IDs can appear in the same row of PLAN_TABLE output, or in different rows, depending on the operation being performed. The examples in "PLAN_TABLE examples showing parallelism" help clarify this concept.

3. **Identify the parallelism mode:**

   The column PARALLELISM_MODE tells you the kind of parallelism that is planned: I for query I/O, C for query CP, and X for Sysplex query. Within a query block, you cannot have a mixture of "I" and "C" parallel modes. However, a statement that uses more than one query block, such as a UNION, can have "I" for one query block and "C" for another. You can have a mixture of "C" and "X" modes in a query block, but not in the same parallel group.

   If the statement was bound while this DB2 is a member of a data sharing group, the PARALLELISM_MODE column can contain "X" even if only this one DB2 member is active. This lets DB2 take advantage of extra processing power that might be available at execution time. If other members are not available at execution time, then DB2 runs the query within the single DB2 member.

PLAN_TABLE examples showing parallelism

For these examples, the other values would not change whether the PARALLELISM_MODE is I, C, or X.

- **Example 1: single table access**

  Assume that DB2 decides at bind time to initiate three concurrent requests to retrieve data from table T1. Part of PLAN_TABLE appears as shown in Table 193 on page 996. If DB2 decides not to use parallel operations for a step, ACCESS_DEGREE and ACCESS_PGROUP_ID contain null values.

Table 193. Part of PLAN_TABLE for single table access

TNAME	METHOD	ACCESS_ DEGREE	ACCESS_ PGROUP_ ID	JOIN_ DEGREE	JOIN_ PGROUP_ ID	SORTC_ PGROUP_ ID	SORTN_ PGROUP_ ID
T1	0	3	1	(null)	(null)	(null)	(null)

- **Example 2: nested loop join**

  Consider a query that results in a series of nested loop joins for three tables, T1, T2 and T3. T1 is the outermost table, and T3 is the innermost table. DB2 decides at bind time to initiate three concurrent requests to retrieve data from each of the three tables. Each request accesses part of T1 and all of T2 and T3. For the nested loop join method with sort, all the retrievals are in the same parallel group except for star join with ACCESSTYPE=T (sparse index). Part of PLAN_TABLE appears as shown in Table 194:

Table 194. Part of PLAN_TABLE for a nested loop join

TNAME	METHOD	ACCESS_ DEGREE	ACCESS_ PGROUP_ ID	JOIN_ DEGREE	JOIN_ PGROUP_ ID	SORTC_ PGROUP_ ID	SORTN_ PGROUP_ ID
T1	0	3	1	(null)	(null)	(null)	(null)
T2	1	3	1	3	1	(null)	(null)
T3	1	3	1	3	1	(null)	(null)

- **Example 3: merge scan join**

  Consider a query that causes a merge scan join between two tables, T1 and T2. DB2 decides at bind time to initiate three concurrent requests for T1 and six concurrent requests for T2. The scan and sort of T1 occurs in one parallel group. The scan and sort of T2 occurs in another parallel group. Furthermore, the merging phase can potentially be done in parallel. Here, a third parallel group is used to initiate three concurrent requests on each intermediate sorted table. Part of PLAN_TABLE appears as shown in Table 195:

Table 195. Part of PLAN_TABLE for a merge scan join

TNAME	METHOD	ACCESS_ DEGREE	ACCESS_ PGROUP_ ID	JOIN_ DEGREE	JOIN_ PGROUP_ ID	SORTC_ PGROUP_ ID	SORTN_ PGROUP_ ID
T1	0	3	d	(null)	(null)	d	(null)
T2	2	6	2	3	3	d	d

In a multi-table join, DB2 might also execute the sort for a composite that involves more than one table in a parallel task. DB2 uses a cost basis model to determine whether to use parallel sort in all cases. When DB2 decides to use parallel sort, SORTC_PGROUP_ID and SORTN_PGROUP_ID indicate the parallel group identifier. Consider a query that joins three tables, T1, T2, and T3, and uses a merge scan join between T1 and T2, and then between the composite and T3. If DB2 decides, based on the cost model, that all sorts in this query are to be performed in parallel, part of PLAN_TABLE appears as shown in Table 196 on page 997:

Table 196. Part of PLAN_TABLE for a multi-table, merge scan join

TNAME	METHOD	ACCESS_ DEGREE	ACCESS_ PGROUP_ ID	JOIN_ DEGREE	JOIN_ PGROUP_ ID	SORTC_ PGROUP_ ID	SORTN_ PGROUP_ ID
T1	0	3	1	(null)	(null)	(null)	(null)
T2	2	6	2	6	3	1	2
T3	2	6	4	6	5	3	4

- **Example 4: hybrid join**

  Consider a query that results in a hybrid join between two tables, T1 and T2. Furthermore, T1 needs to be sorted; as a result, in PLAN_TABLE the T2 row has SORTC_JOIN=Y. DB2 decides at bind time to initiate three concurrent requests for T1 and six concurrent requests for T2. Parallel operations are used for a join through a clustered index of T2.

  Because T2's RIDs can be retrieved by initiating concurrent requests on the clustered index, the joining phase is a parallel step. The retrieval of T2's RIDs and T2's rows are in the same parallel group. Part of PLAN_TABLE appears as shown in Table 197:

Table 197. Part of PLAN_TABLE for a hybrid join

TNAME	METHOD	ACCESS_ DEGREE	ACCESS_ PGROUP_ ID	JOIN_ DEGREE	JOIN_ PGROUP_ ID	SORTC_ PGROUP_ ID	SORTN_ PGROUP_ ID
T1	0	3	1	(null)	(null)	(null)	(null)
T2	4	6	2	6	2	1	(null)

Monitoring parallel operations

The number of parallel operations or tasks used to access data is initially determined at bind time, and later adjusted when the query is executed.

Bind time: At bind time, DB2 collects partition statistics from the catalog, estimates the processor cycles for the costs of processing the partitions, and determines the optimal number of parallel tasks to achieve minimum elapsed time.

When a planned degree exceeds the number of online CPs, the query might not be completely processor-bound. Instead it might be approaching the number of partitions because it is I/O-bound. In general, the more I/O-bound a query is, the closer the degree of parallelism is to the number of partitions.

In general, the more processor-bound a query is, the closer the degree of parallelism is to the number of online CPs, and it can even exceed the number of CPs by one.

Example: Suppose that you have a processor-intensive query on a 10-partition table, and that this query is running on a 6-way CPC. The degree of parallelism can be up to 7 in this case.

To help DB2 determine the optimal degree of parallelism, use the utility RUNSTATS to keep your statistics current.

PLAN_TABLE shows the planned degree of parallelism in the columns ACCESS_DEGREE and JOIN_DEGREE.

Execution time: For each parallel group, parallelism (either CP or I/O) can execute at a reduced degree or degrade to sequential operations for the following reasons:

- Amount of virtual buffer pool space available
- Host variable values
- Availability of the hardware sort assist facility
- Ambiguous cursors
- A change in the number or configuration of online processors
- The join technique that DB2 uses (I/O parallelism not supported when DB2 uses the star join technique)

At execution time, a plan using Sysplex query parallelism can use CP parallelism. All parallelism modes can degenerate to a sequential plan. No other changes are possible.

Using DISPLAY BUFFERPOOL

You can use the output from DISPLAY BUFFERPOOL DETAIL report to see how well the buffer pool is able to satisfy parallel operations.

```
DSNB440I = PARALLEL ACTIVITY -
           PARALLEL REQUEST =      282  DEGRADED PARALLEL=        5
```

The PARALLEL REQUEST field in this example shows that DB2 was negotiating buffer pool resource for 282 parallel groups. Of those 282 groups, only 5 were degraded because of a lack of buffer pool resource. A large number in the DEGRADED PARALLEL field could indicate that there are not enough buffers that can be used for parallel processing.

Using DISPLAY THREAD

DISPLAY THREAD displays parallel tasks. Whereas previously you would only see information about the originating task, now you can see information about the parallel tasks associated with that originating task. The status field contains PT for parallel tasks. All parallel tasks are displayed immediately after their corresponding originating thread.

See Chapter 2 of *DB2 Command Reference* for information about the syntax of the command DISPLAY THREAD.

Using DB2 trace

The statistics trace indicates when parallel groups do not run to the planned degree or run sequentially. These are possible indicators that there are queries that are not achieving the best possible response times. Use the accounting trace to ensure that your parallel queries are meeting their response time goals. If there appears to be a problem with a parallel query, then use the performance trace to do further analysis.

Accounting trace

By default, DB2 rolls task accounting into an accounting record for the originating task. OMEGAMON also summarizes all accounting records generated for a parallel query and presents them as one logical accounting record. OMEGAMON presents the times for the originating tasks separately from the accumulated times for all the parallel tasks.

As shown in Figure 114 on page 999 CPU TIME-AGENT is the time for the originating tasks, while CPU TIME-PAR.TASKS (**A**) is the accumulated processing time for the parallel tasks.

```
TIMES/EVENTS  APPL (CLASS 1)   DB2 (CLASS 2)    CLASS 3 SUSP.   ELAPSED TIME
------------  --------------  --------------    --------------  ------------
ELAPSED TIME       32.578741       32.312218    LOCK/LATCH         25.461371
 NONNESTED         28.820003       30.225885    SYNCHRON. I/O       0.142382
 STORED PROC        3.758738        2.086333     DATABASE I/O       0.116320
 UDF                0.000000        0.000000     LOG WRTE I/O       0.026062
 TRIGGER            0.000000        0.000000    OTHER READ I/O   3:00.404769
                                                OTHER WRTE I/O      0.000000
CPU TIME         1:29.695300     1:29.644026    SER.TASK SWTCH      0.000000
 AGENT              0.225153        0.178128     UPDATE COMMIT      0.000000
  NONNESTED         0.132351        0.088834     OPEN/CLOSE         0.000000
  STORED PRC        0.092802        0.089294     SYSLGRNG REC       0.000000
  UDF               0.000000        0.000000     EXT/DEL/DEF        0.000000
  TRIGGER           0.000000        0.000000     OTHER SERVICE      0.000000
 PAR.TASKS      A 1:29.470147     1:29.465898    ARC.LOG(QUIES)      0.000000

⋮
...            QUERY PARALLEL.     TOTAL
               ---------------  --------
               MAXIMUM MEMBERS         1
               MAXIMUM DEGREE         10
               GROUPS EXECUTED         1
                RAN AS PLANNED  B      1
                RAN REDUCED     C      0
                ONE DB2 COOR=N         0
                ONE DB2 ISOLAT         0
                ONE DB2 DCL TTABLE     0
                SEQ - CURSOR    D      0
                SEQ - NO ESA    E      0
                SEQ - NO BUF    F      0
                SEQ - ENCL.SER.        0
               MEMB SKIPPED(%)         0
               DISABLED BY RLF  G   NO
               REFORM PARAL-CONFIG  H  0
               REFORM PARAL-NO BUF     0
```

Figure 114. Partial accounting trace, query parallelism

As the report shows, the values for CPU TIME and I/O WAIT TIME are larger than the elapsed time. The processor and suspension time can be greater than the elapsed time because these two times are accumulated from multiple parallel tasks. The elapsed time would be less than the processor and suspension time if these two times are accumulated sequentially.

If you have baseline accounting data for the same thread run without parallelism, the elapsed times and processor times should not be significantly larger when that query is run in parallel. If it is significantly larger, or if response time is poor, you need to examine the accounting data for the individual tasks. Use the OMEGAMON Record Trace for the IFCID 0003 records of the thread you want to examine. Use the performance trace if you need more information to determine the cause of the response time problem.

Performance trace

The performance trace can give you information about tasks within a group. To determine the actual number of parallel tasks used, refer to field QW0221AD in IFCID 0221, as mapped by macro DSNDQW03. The 0221 record also gives you information about the key ranges used to partition the data.

IFCID 0222 contains the elapsed time information for each parallel task and each parallel group in each SQL query. OMEGAMON presents this information in its SQL Activity trace.

If your queries are running sequentially or at a reduced degree because of a lack of buffer pool resources, the QW0221C field of IFCID 0221 indicates which buffer pool is constrained.

Tuning parallel processing

Much of the information in this section applies also to Sysplex query parallelism. See Chapter 6 of *DB2 Data Sharing: Planning and Administration* for more information.

If there are many parallel groups that do not run at the planned degree (see **B** in Figure 114 on page 999), check the following factors:

- Buffer pool availability

  Depending on buffer pool availability, DB2 could reduce the degree of parallelism (see **C** in Figure 114 on page 999) or revert to a sequential plan before executing the parallel group (**F** in the figure).

  To determine which buffer pool is short on storage, see section QW0221C in IFCID 0221. You can use the ALTER BUFFERPOOL command to increase the buffer pool space available for parallel operations by modifying the following parameters:
  - VPSIZE, the size of the virtual buffer pool
  - VPSEQT, the sequential steal threshold
  - VPPSEQT, the parallel sequential threshold
  - VPXPSEQT, the assisting parallel sequential threshold, used only for Sysplex query parallelism.

  If the buffer pool is busy with parallel operations, the sequential prefetch quantity might also be reduced.

  The parallel sequential threshold also has an impact on **work file processing** for parallel queries. DB2 assumes that you have all your work files of the same size (4KB or 32KB) in the same buffer pool and makes run time decisions based on a single buffer pool. A lack of buffer pool resources for the work files can lead to a reduced degree of parallelism or cause the query to run sequentially.

  If increasing the parallel thresholds does not help solve the problem of reduced degree, you can increase the total buffer pool size (VPSIZE). Use information from the statistics trace to determine the amount of buffer space you need. Use the following formula:

  ```
  (QBSTJIS ⁄ QBSTPQF) × 32 = buffer increase value
  ```

  QBSTJIS is the total number of requested prefetch I/O streams that were denied because of a storage shortage in the buffer pool. (There is one I/O stream per parallel task.) QBSTPQF is the total number of times that DB2 could not allocate enough buffer pages to allow a parallel group to run to the planned degree.

  As an example, assume QBSTJIS is 100 000 and QBSTPQF is 2500:

  ```
  (100000 ⁄ 2500) × 32 = 1280
  ```

  Use ALTER BUFFERPOOL to increase the current VPSIZE by 2560 buffers to alleviate the degree degradation problem. Use the DISPLAY BUFFERPOOL command to see the current VPSIZE.

- Physical contention

  As much as possible, put data partitions on separate physical devices to minimize contention. Try not to use more partitions than there are internal paths in the controller.

- Run time host variables

A host variable can determine the qualifying partitions of a table for a given query. In such cases, DB2 defers the determination of the planned degree of parallelism until run time, when the host variable value is known.

- Updatable cursor

  At run time, DB2 might determine that an ambiguous cursor is updatable. This appears in **D** in the accounting report.

- Proper hardware and software support

  If you do not have the hardware sort facility at run time, and a sort merge join is needed, you see a value in **E**.

- A change in the configuration of online processors

  If there are fewer online processors at run time than at bind time, DB2 reformulates the parallel degree to take best advantage of the current processing power. This reformulation is indicated by a value in **H** in the accounting report.

Locking considerations for repeatable read applications: For CP parallelism, locks are obtained independently by each task. Be aware that this situation can possibly increase the total number of locks taken for applications that:

- Use an isolation level of repeatable read
- Use CP parallelism
- Repeatedly access the table space using a lock mode of IS without issuing COMMITs

Recommendation: As is recommended for all repeatable-read applications, issue frequent COMMITs to release the lock resources that are held. Repeatable read or read stability isolation cannot be used with Sysplex query parallelism.

Disabling query parallelism

To disable parallel operations, do any of the following actions:

- For static SQL, rebind to change the option DEGREE(ANY) to DEGREE(1). You can do this by using the DB2I panels, the DSN subcommands, or the DSNH CLIST. The default is DEGREE(1).
- For dynamic SQL, execute the following SQL statement:

  ```
  SET CURRENT DEGREE = '1';
  ```

  The default value for CURRENT DEGREE is 1 unless your installation has changed the default for the CURRENT DEGREE special register.

- Set the parallel sequential threshold (VPPSEQT) to 0.
- Add a row to your resource limit facility's specification table (RLST) for your plan, package, or authorization ID with the RLFFUNC value set to "3" to disable I/O parallelism, "4" to disable CP parallelism, or "5" to disable Sysplex query parallelism. To disable all types of parallelism, you need a row for all three types (assuming that Sysplex query parallelism is enabled on your system.) In a system with a very high processor utilization rate (that is, greater than 98 percent), I/O parallelism might be a better choice because of the increase in processor overhead with CP parallelism. In this case, you could disable CP parallelism for your dynamic queries by putting a "4" in the resource limit specification table for the plan or package.

  If you have a Sysplex, you might want to use a "5" to disable Sysplex query parallelism, depending on how high processor utilization is in the members of the data sharing group.

To determine if parallelism has been disabled by a value in your resource limit specification table (RLST), look for a non-zero value in field QXRLFDPA in IFCID 0002 or 0003 (shown in **G** in Figure 114 on page 999). The QW0022RP field in IFCID 0022 indicates whether this particular statement was disabled. For more information about how the resource limit facility governs modes of parallelism, see "Descriptions of the RLST columns" on page 719.

Chapter 36. Tuning and monitoring in a distributed environment

This chapter describes some ways you can tune your systems and applications that use DRDA or DB2 private protocol for distributed data access.

This chapter contains the following sections:

- "Remote access types: DRDA and private protocol"
- "Tuning distributed applications" on page 1004
- "Monitoring DB2 in a distributed environment" on page 1013
- "Monitoring distributed processing with RMF" on page 1017

Remote access types: DRDA and private protocol

DB2 supports two different types of remote access between the requesting relational database management system (DBMS) and the serving relational database management system. The two types of access are *DRDA access* and *DB2 private protocol access*. When three-part named objects are referenced (or aliases for three-part name objects are referenced), DB2 chooses between the two connection types based on the bind option that you choose (or the default protocol set at your site).

Recommendation: Use DRDA for new applications, and migrate existing private protocol applications to DRDA. No enhancements are planned for private protocol.

Characteristics of DRDA

With DRDA, the application can remotely bind packages and can execute packages of static or dynamic SQL that have previously been bound at that location. DRDA has the following characteristics and benefits:

- With DRDA access, an application can access data at any server that supports DRDA, not just a DB2 server on a z/OS operating system.
- DRDA supports all SQL features, including user-defined functions, LOBs, and stored procedures.
- DRDA can avoid multiple binds and minimize the number of binds that are required.
- DRDA supports multiple-row FETCH.

DRDA is the preferred method for remote access with DB2.

Characteristics of DB2 private protocol

Private protocol is an older method for remote access. It can be used only between DB2 subsystems and only over a SNA network. Private protocol has not been enhanced to support many new SQL features. Because of these limitations, it is highly recommended that you migrate to DRDA protocol for communicating between DB2 subsystems.

© Copyright IBM Corp. 1982, 2008

Tuning distributed applications

A query sent to a remote system can sometimes take longer to execute than the same query, accessing tables of the same size, on the local DB2 subsystem. The principal reasons for this potential increase in execution time are:

- The time required to send messages across the network
- Overhead processing, including startup and communication subsystem session management

Some aspects of overhead processing, for instance, network processing, are not under DB2 control. (Suggestions for tuning your network are in Part 3 of *DB2 Installation Guide*.)

Monitoring and tuning performance in a distributed environment is a complex task that requires knowledge of several products. Some guidelines follow for improving the performance of distributed applications. The guidelines are divided into the following areas:

- "Application and requesting systems"
- "Serving system" on page 1012

Application and requesting systems

Minimizing the number of messages sent between the requester and the server is a primary way to improve performance. This section describes the following topics:

- "BIND options"
- "SQL statement options"
- "Block fetching result sets" on page 1005
- "Optimizing for very large results sets for DRDA" on page 1010
- "Optimizing for small results sets for DRDA" on page 1011

BIND options

If appropriate for your applications, consider the following bind options to improve performance:

- Use the DEFER(PREPARE) bind option, which can reduce the number of messages that must be sent back and forth across the network. For more information on using the DEFER(PREPARE) option, see Part 4 of *DB2 Application Programming and SQL Guide*.
- Bind application plans and packages with ISOLATION(CS) to reduce contention and message overhead.

SQL statement options

If appropriate for your applications, consider using the following strategies to improve the performance of SQL statements:

Commit frequently
Commit frequently to avoid holding resources at the server.

Minimize SQL statements
Avoid using several SQL statements when one well-tuned SQL statement can retrieve the desired results. Alternatively, put your SQL statements in a stored procedure, issue your SQL statements at the server through the stored procedure, and return the result. Using a stored procedure creates only one send and receive operation (for the CALL statement) instead of a potential send and receive operation for each SQL statement.

Depending on how many SQL statements are in your application, using stored procedures can significantly decrease your elapsed time and may

decrease your processor costs. For more information on how to use stored procedures, see Part 6 of *DB2 Application Programming and SQL Guide*.

Use the RELEASE statement and the DISCONNECT(EXPLICIT) bind option
The RELEASE statement minimizes the network traffic that is needed to release a remote connection at commit time. For example, if the application has connections to several different servers, specify the RELEASE statement when the application has completed processing for each server. The RELEASE statement does not close cursors, release any resources, or prevent further use of the connection until the COMMIT is issued. It just makes the processing at COMMIT time more efficient.

The bind option DISCONNECT(EXPLICIT) destroys all remote connections for which RELEASE was specified.

Use the COMMIT ON RETURN YES clause
Consider using the COMMIT ON RETURN YES clause of the CREATE PROCEDURE statement to indicate that DB2 should issue an implicit COMMIT on behalf of the stored procedure upon return from the CALL statement. Using the clause can reduce the length of time locks are held and can reduce network traffic. With COMMIT ON RETURN YES, any updates made by the client before calling the stored procedure are committed with the stored procedure changes. See Part 6 of *DB2 Application Programming and SQL Guide* for more information.

Set the CURRENT RULES special register to DB2
When requesting LOB data, set the CURRENT RULES special register to DB2 instead of to STD before performing a CONNECT. A value of DB2, which is the default, can offer performance advantages. When a DB2 UDB for z/OS server receives an OPEN request for a cursor, the server uses the value in the CURRENT RULES special register to determine whether the application intends to switch between LOB values and LOB locator values when fetching different rows in the cursor. If you specify a value of DB2 for CURRENT RULES, the application indicates that the first FETCH request will specify the format for each LOB column in the answer set and that the format will not change in a subsequent FETCH request. However, if you set the value of CURRENT RULES to STD, the application intends to fetch a LOB column into either a LOB locator host variable or a LOB host variable.

Although a value of STD for CURRENT RULES gives you more programming flexibility when you retrieve LOB data, you can get better performance if you use a value of DB2. With the STD option, the server will not block the cursor, while with the DB2 option it may block the cursor where it is possible to do so. For more information, see "LOB data and its effect on block fetch for DRDA" on page 1007.

Block fetching result sets

Block fetch can significantly decrease the number of messages sent across the network. Block fetch is used only with cursors that will not update or delete data. With block fetch, DB2 groups the rows that are retrieved by an SQL query into as large a "block" of rows as will fit in a message buffer. DB2 then transmits the block over the network, without requiring a separate message for each row.

DB2 can use two different types of block fetch:
- Limited block fetch
- Continuous block fetch

Both types of block fetch are used for DRDA and private protocol, but the implementation of continuous block fetch for DRDA is slightly different than that for private protocol.

Continuous block fetch: In terms of response times, continuous block fetch is most efficient for larger result sets because fewer messages are transmitted and because overlapped processing is performed at the requester and the server. However, continuous block fetch uses more networking resources than limited block fetch. When networking resources are critical, use limited block fetch to run applications.

The requester can use both forms of blocking at the same time and with different servers.

If an application is doing read-only processing and can use continuous block fetch, the sequence goes like this:

1. The requester sends a message to open a cursor and begins fetching the block of rows at the server.
2. The server sends back a block of rows and the requester begins processing the first row.
3. The server continues to send blocks of rows to the requester, without further prompting. The requester processes the second and later rows as usual, but fetches them from a buffer on the requester's system.

For private protocol, continuous block fetch uses one conversation for each open cursor. Having a dedicated conversation for each cursor allows the server to continue sending until all the rows are returned.

For DRDA, only one conversation is used, and it must be made available to the other SQL statements that are in the application. Thus, the server usually sends back a subset of all the rows. The number of rows that the server sends depends on the following factors:

- The size of each row
- The number of extra blocks that are requested by the requesting system compared to the number of extra blocks that the server will return

  For a DB2 UDB for z/OS requester, the EXTRA BLOCKS REQ field on installation panel DSNTIP5 determines the maximum number of extra blocks requested. For a DB2 UDB for z/OS server, the EXTRA BLOCKS SRV field on installation panel DSNTIP5 determines the maximum number of extra blocks allowed.

  Example: Suppose that the requester asks for 100 extra query blocks and that the server allows only 50. The server returns no more than 50 extra query blocks. The server might choose to return fewer than 50 extra query blocks for any number of reasons that DRDA allows.
- Whether continuous block fetch is enabled, and the number of extra rows that the server can return if it regulates that number.

  To enable continuous block fetch for DRDA and to regulate the number of extra rows sent by a DB2 UDB for z/OS server, you must use the OPTIMIZE FOR n ROWS clause on your SELECT statement. See "Optimizing for very large results sets for DRDA" on page 1010 for more information.

If you want to use continuous block fetch for DRDA, have the application fetch all the rows of the cursor before doing any other SQL. Fetching all the rows first

prevents the requester from having to buffer the data, which can consume a lot of storage. Choose carefully which applications should use continuous block fetch for DRDA.

Limited block fetch: Limited block fetch guarantees the transfer of a minimum amount of data in response to each request from the requesting system. With limited block fetch, a single conversation is used to transfer messages and data between the requester and server for multiple cursors. Processing at the requester and server is synchronous. The requester sends a request to the server, which causes the server to send a response back to the requester. The server must then wait for another request to tell it what should be done next.

Block fetch with scrollable cursors for DRDA: When a DB2 UDB for z/OS requester uses a scrollable cursor to retrieve data from a DB2 UDB for z/OS server, the following conditions are true:

- The requester never requests more than 64 rows in a query block, even if more rows fit in the query block. In addition, the requester never requests extra query blocks. This is true even if the setting of field EXTRA BLOCKS REQ in the DISTRIBUTED DATA FACILITY PANEL 2 installation panel on the requester allows extra query blocks to be requested.
- The requester discards rows of the result table if the application does not use those rows.

  Example: If the application fetches row *n* and then fetches row *n*+2, the requester discards row *n*+1.

  The application gets better performance for a blocked scrollable cursor if it mostly scrolls forward, fetches most of the rows in a query block, and avoids frequent switching between FETCH ABSOLUTE statements with negative and positive values.
- If the scrollable cursor does not use block fetch, the server returns one row for each FETCH statement.

LOB data and its effect on block fetch for DRDA: For a non-scrollable blocked cursor, the server sends all the non-LOB data columns for a block of rows in one message, including LOB locator values. As each row is fetched by the application, the requester obtains the non-LOB data columns directly from the query block. If the row contains non-null and non-zero length LOB values, those values are retrieved from the server at that time. This behavior limits the impact to the network by pacing the amount of data that is returned at any one time. If all LOB data columns are retrieved into LOB locator host variables or if the row does not contain any non-null or non-zero length LOB columns, then the whole row can be retrieved directly from the query block.

For a scrollable blocked cursor, the LOB data columns are returned at the same time as the non-LOB data columns. When the application fetches a row that is in the block, a separate message is not required to get the LOB columns.

Ensuring block fetch:

General-use Programming Interface

To use either limited or continuous block fetch, DB2 must determine that the cursor is not used for updating or deleting. The easiest way to indicate that the cursor does not modify data is to add the FOR FETCH ONLY or FOR READ ONLY clause to the query in the DECLARE CURSOR statement as in the following example:

```
EXEC SQL
  DECLARE THISEMP CURSOR FOR
    SELECT EMPNO, LASTNAME, WORKDEPT, JOB
    FROM DSN8810.EMP
    WHERE WORKDEPT = 'D11'
    FOR FETCH ONLY
END-EXEC.
```

If you do not use FOR FETCH ONLY or FOR READ ONLY, DB2 still uses block fetch for the query if the following conditions are true:

- The cursor is a non-scrollable cursor, and the result table of the cursor is read-only. This applies to static and dynamic cursors except for read-only views. (See Chapter 5 of *DB2 SQL Reference* for information about declaring a cursor as read-only.)
- The cursor is a scrollable cursor that is declared as INSENSITIVE, and the result table of the cursor is read-only.
- The cursor is a scrollable cursor that is declared as SENSITIVE, the result table of the cursor is read-only, and the value of bind option CURRENTDATA is NO.
- The result table of the cursor is not read-only, but the cursor is ambiguous, and the value of bind option CURRENTDATA is NO. A cursor is ambiguous when:
  - It is not defined with the clauses FOR FETCH ONLY, FOR READ ONLY, or FOR UPDATE OF.
  - It is not defined on a read-only result table.
  - It is not the target of a WHERE CURRENT clause on an SQL UPDATE or DELETE statement.
  - It is in a plan or package that contains the SQL statements PREPARE or EXECUTE IMMEDIATE.

DB2 triggers block fetch for static SQL only when it can detect that no updates or deletes are in the application. For dynamic statements, because DB2 cannot detect what follows in the program, the decision to use block fetch is based on the declaration of the cursor.

DB2 does not use continuous block fetch if the following conditions are true:

- The cursor is referred to in the statement DELETE WHERE CURRENT OF elsewhere in the program.
- The cursor statement appears that it can be updated at the requesting system. (DB2 does not check whether the cursor references a view at the server that cannot be updated.)

The following three tables summarize the conditions under which a DB2 server uses block fetch.

Table 198 shows the conditions for a non-scrollable cursor.

Table 198. Effect of CURRENTDATA and cursor type on block fetch for a non-scrollable cursor

Isolation level	CURRENTDATA	Cursor type	Block fetch
CS, RR, or RS	Yes	Read-only	Yes
		Updatable	No
		Ambiguous	No
	No	Read-only	Yes
		Updatable	No
		Ambiguous	Yes
UR	Yes	Read-only	Yes
	No	Read-only	Yes

Table 199 shows the conditions for a scrollable cursor that is not used to retrieve a stored procedure result set.

Table 199. Effect of CURRENTDATA and isolation level on block fetch for a scrollable cursor that is not used for a stored procedure result set

Isolation level	Cursor sensitivity	CURRENTDATA	Cursor type	Block fetch
CS, RR, or RS	INSENSITIVE	Yes	Read-only	Yes
		No	Read-only	Yes
	SENSITIVE	Yes	Read-only	No
			Updatable	No
			Ambiguous	No
		No	Read-only	Yes
			Updatable	No
			Ambiguous	Yes
UR	INSENSITIVE	Yes	Read-only	Yes
		No	Read-only	Yes
	SENSITIVE	Yes	Read-only	Yes
		No	Read-only	Yes

Table 200 shows the conditions for a scrollable cursor that is used to retrieve a stored procedure result set.

Table 200. Effect of CURRENTDATA and isolation level on block fetch for a scrollable cursor that is used for a stored procedure result set

Isolation level	Cursor sensitivity	CURRENTDATA	Cursor type	Block fetch
CS, RR, or RS	INSENSITIVE	Yes	Read-only	Yes
		No	Read-only	Yes
	SENSITIVE	Yes	Read-only	No
		No	Read-only	Yes

Table 200. Effect of CURRENTDATA and isolation level on block fetch for a scrollable cursor that is used for a stored procedure result set (continued)

Isolation level	Cursor sensitivity	CURRENTDATA	Cursor type	Block fetch
UR	INSENSITIVE	Yes	Read-only	Yes
		No	Read-only	Yes
	SENSITIVE	Yes	Read-only	Yes
		No	Read-only	Yes

End of General-use Programming Interface

Optimizing for very large results sets for DRDA

Enabling a DB2 client to request multiple query blocks on each transmission can reduce network activity and improve performance significantly for applications that use DRDA access to download large amounts of data. You can specify a large value of *n* in the OPTIMIZE FOR *n* ROWS clause of a SELECT statement to increase the number of DRDA query blocks that a DB2 server returns in each network transmission for a non-scrollable cursor. If *n* is greater than the number of rows that fit in a DRDA query block, OPTIMIZE FOR *n* ROWS lets the DRDA client request multiple blocks of query data on each network transmission instead of requesting a new block when the first block is full. This use of OPTIMIZE FOR *n* ROWS is intended only for applications in which the application opens a cursor and downloads great amounts of data. The OPTIMIZE FOR *n* ROWS clause has no effect on scrollable cursors.

Recommendation: Because the application SQL uses only one conversation, do not try to do other SQL work until the entire answer set is processed. If the application issues another SQL statement before the previous statement's answer set has been received, DDF must buffer them in its address space. You can buffer up to 10 MB in this way.

Because specifying a large number of network blocks can saturate the network, limit the number of blocks according to what your network can handle. You can limit the number of blocks used for these large download operations. When the client supports extra query blocks, DB2 chooses the **smallest** of the following values when determining the number of query blocks to send:

- The number of blocks into which the number of rows (*n*) on the OPTIMIZE clause will fit. For example, assume you specify 10000 rows for *n*, and the size of each row that is returned is approximately 100 bytes. If the block size used is 32 KB (32768 bytes), the calculation is as follows:

  ```
  (10000 * 100) / 32768 = 31 blocks
  ```

- The DB2 server value for the EXTRA BLOCKS SRV field on installation panel DSNTIP5. The maximum value that you can specify is 100.
- The client's extra query block limit, which is obtained from the DRDA MAXBLKEXT parameter received from the client. When DB2 UDB for z/OS acts as a DRDA client, you set this parameter at installation time with the EXTRA BLOCKS REQ field on installation panel DSNTIP5. The maximum value that you can specify is 100. DB2 Connect sets the MAXBLKEXT parameter to –1 (unlimited).

If the client does not support extra query blocks, the DB2 server on z/OS automatically reduces the value of *n* to match the number of rows that fit within a DRDA query block.

Recommendation for cursors that are defined WITH HOLD: Do not set a large number of query blocks for cursors that are defined WITH HOLD. If the application commits while there are still a lot of blocks in the network, DB2 buffers the blocks in the requester's memory (the *ssnm*DIST address space if the requester is a DB2 UDB for z/OS) before the commit can be sent to the server.

For examples of performance problems that can occur from not using OPTIMIZE FOR *n* ROWS when downloading large amounts of data, see Part 4 of *DB2 Application Programming and SQL Guide*.

Optimizing for small results sets for DRDA

When a client does not need all the rows from a potentially large result set, preventing the DB2 server from returning all the rows for a query can reduce network activity and improve performance significantly for DRDA applications. You can use either the OPTIMIZE FOR *n* ROWS clause or the FETCH FIRST *n* ROWS ONLY clause of a SELECT statement to limit the number of rows returned to a client program.

Using OPTIMIZE FOR *n* ROWS: When you specify OPTIMIZE FOR *n* ROWS and *n* is less than the number of rows that fit in the DRDA query block (default size on z/OS is 32 KB), the DB2 server prefetches and returns only as many rows as fit into the query block. For example, if the client application is interested in seeing only one screen of data, specify OPTIMIZE FOR *n* ROWS, choosing a small number for *n*, such as 3 or 4. The OPTIMIZE FOR *n* ROWS clause has no effect on scrollable cursors.

Using FETCH FIRST *n* ROWS ONLY: The FETCH FIRST *n* ROWS ONLY clause does not affect network blocking. If FETCH FIRST *n* ROWS ONLY is specified and OPTIMIZE FOR *n* ROWS is not specified, DB2 uses the FETCH FIRST value to optimize the access path. However, DRDA does not consider this value when it determines network blocking.

When both the FETCH FIRST *n* ROWS ONLY clause and the OPTIMIZE FOR *n* ROWS clause are specified, the value for the OPTIMIZE FOR *n* ROWS clause is used for access path selection.

Example: Suppose that you submit the following SELECT statement:

```
SELECT * FROM EMP
FETCH FIRST 5 ROWS ONLY
OPTIMIZE FOR 20 ROWS;
```

The OPTIMIZE FOR value of 20 rows is used for network blocking and access path selection.

When you use FETCH FIRST *n* ROWS ONLY, DB2 might use a *fast implicit close.* Fast implicit close means that during a distributed query, the DB2 server automatically closes the cursor when it prefetches the *n*th row if FETCH FIRST *n* ROWS ONLY is specified or when there are no more rows to return. Fast implicit close can improve performance because it can save an additional network transmission between the client and the server.

DB2 uses fast implicit close when the following conditions are true:
- The query uses limited block fetch.
- The query retrieves no LOBs.
- The cursor is not a scrollable cursor.
- Either of the following conditions is true:

- The cursor is declared WITH HOLD, and the package or plan that contains the cursor is bound with the KEEPDYNAMIC(YES) option.
- The cursor is declared WITH HOLD and the DRDA client passes the QRYCLSIMP parameter set to SERVER MUST CLOSE, SERVER DECIDES, or SERVER MUST NOT CLOSE.
- The cursor is not defined WITH HOLD.

When you use FETCH FIRST *n* ROWS ONLY and DB2 does a fast implicit close, the DB2 server closes the cursor after it prefetches *n* rows, or when there are no more rows.

Data encryption security options

Data encryption security options provide added security for the security-sensitive data that an application requests from the system. However, the encryption options can also have a negative impact on performance. The following encryption options have a larger performance cost than other options:

- Encrypted user ID and encrypted security-sensitive data
- Encrypted user ID, encrypted password, and encrypted security-sensitive data

Recommendation: To maximize performance of requester systems, use the minimum level of security that is required by the sensitivity of the data.

Serving system

For access using DB2 private protocol, the serving system is the DB2 system on which the SQL is dynamically executed. For access using DRDA, the serving system is the system on which your remotely bound package executes.

If you are executing a package on a remote DBMS, then improving performance on the server depends on the nature of the server. If the remote DBMS on which the package executes is another DB2, then the information in Chapter 34, "Using EXPLAIN to improve SQL performance," on page 927 is appropriate for access path considerations.

Other considerations that could affect performance on a DB2 server are:

- The maximum number of database access threads that the server allows to be allocated concurrently. (This is the MAX REMOTE ACTIVE field on installation panel DSNTIPE.) A request can be queued while waiting for an available thread. Making sure that requesters commit frequently can let threads be used by other requesters. See "Setting thread limits for database access threads" on page 735 for more information.
- The priority of database access threads on the remote system. A low priority could impede your application's distributed performance. See "Using z/OS workload management to set performance objectives" on page 739 for more information.
- For instructions on avoiding RACF calls at the server, see "Controlling requests from remote applications" on page 238, and more particularly "Do you manage inbound IDs through DB2 or RACF?" on page 243.

When DB2 is the server, it is a good idea to activate accounting trace class 7. This provides accounting information at the package level, which can be very useful in determining performance problems.

Monitoring DB2 in a distributed environment

DB2 provides several ways to monitor DB2 data and events in a distributed environment. You can use the DISPLAY command and the trace facility to obtain information. DB2 can also return server-elapsed time to certain types of client applications.

The DISPLAY command

The DB2 DISPLAY command gives you information about the status of threads, databases, tracing, allied subsystems, and applications. Several forms of the DISPLAY command are particularly helpful for monitoring DB2: DISPLAY THREAD, DISPLAY LOCATION, DISPLAY DDF DETAIL, DISPLAY DATABASE, and DISPLAY TRACE. For the detailed syntax of each command, refer to Chapter 2 of *DB2 Command Reference*. See also:

- "Monitoring threads" on page 379
- "The command DISPLAY LOCATION" on page 403
- "The command DISPLAY DDF" on page 402

Tracing distributed events

A number of IFCIDs, including IFCID 0001 (statistics) and IFCID 0003 (accounting), record distributed data and events.

If your applications update data at other sites, turn on the statistics class 4 trace and always keep it active. This statistics trace covers error situations surrounding in doubt threads; it provides a history of events that might impact data availability and data consistency.

DB2 accounting records are created separately at the requester and each server. Events are recorded in the accounting record at the location where they occur. When a thread becomes active, the accounting fields are reset. Later, when the thread becomes inactive or is terminated, the accounting record is created.

Figure 115 on page 1014 shows the relationship of the accounting class 1 and 2 times and the requester and server accounting records. Figure 116 on page 1015 and Figure 117 on page 1016 show the server and requester distributed data facility blocks from the OMEGAMON accounting long trace.

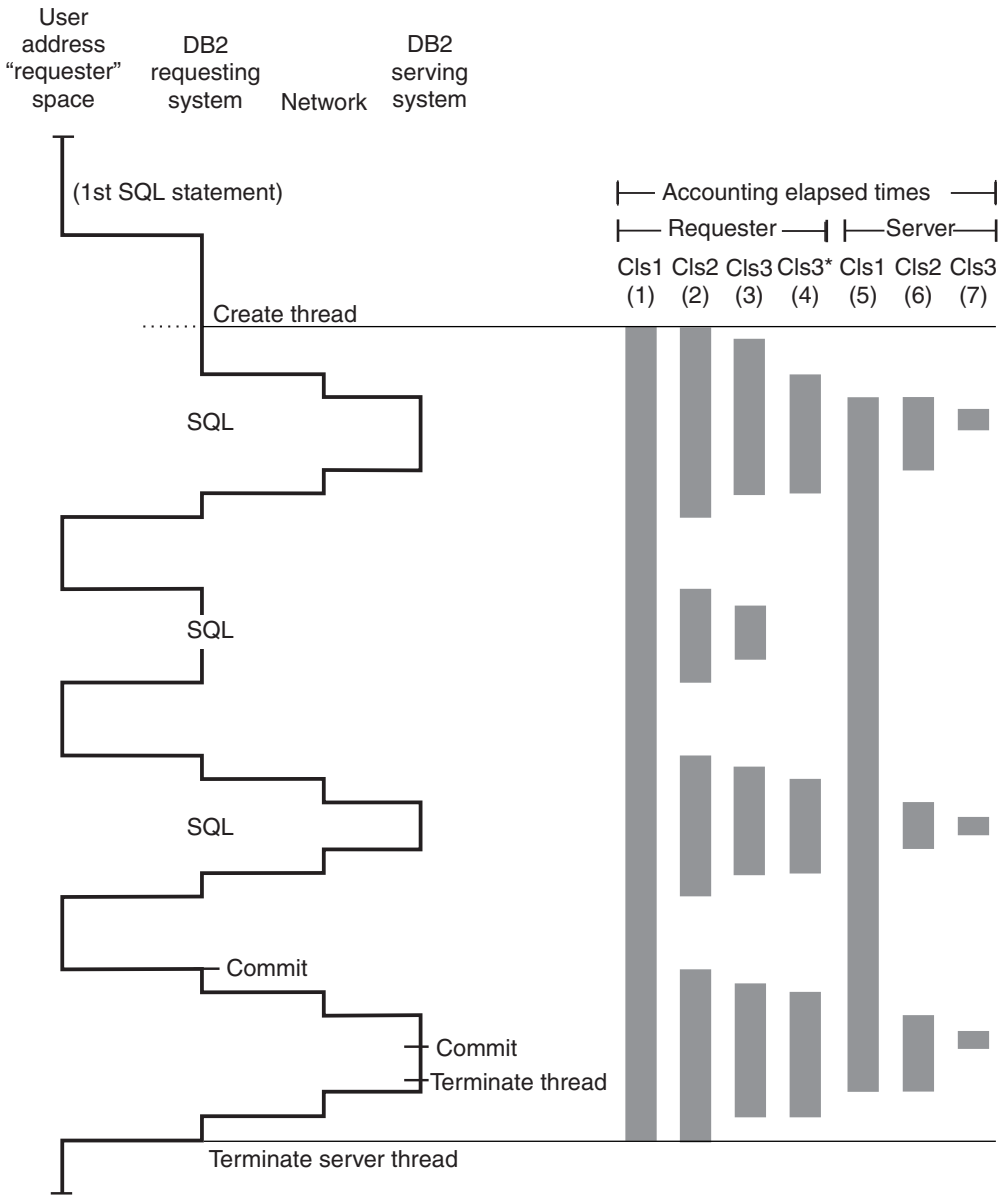


Figure 115. Elapsed times in a DDF environment as reported by OMEGAMON. These times are valid for access that uses either DRDA or private protocol (except as noted).

This figure is a simplified picture of the processes that go on in the serving system. It does not show block fetch statements and is only applicable to a single row retrieval.

The various elapsed times referred to in the header are:

- (1) - Requester Cls1

  This time is reported in the ELAPSED TIME field under the APPL (CLASS 1) column near the top of the OMEGAMON accounting long trace for the requesting DB2 subsystem. It represents the elapsed time from the creation of the allied distributed thread until the termination of the allied distributed thread.
- (2) - Requester Cls2

  This time is reported in the ELAPSED TIME field under the DB2 (CLASS 2) column near the top of the OMEGAMON accounting long trace for the

requesting DB2 subsystem. It represents the elapsed time from when the application passed the SQL statements to the local DB2 system until return. This is considered “In DB2” time.

- (3) - Requester Cls3

  This time is reported in the TOTAL CLASS 3 field under the CLASS 3 SUSP column near the top of the OMEGAMON accounting long trace for the requesting DB2 system. It represents the amount of time the requesting DB2 system spent suspended waiting for locks or I/O.

- (4) - Requester Cls3* (Requester wait time for activities not in DB2)

  This time is reported in the SERVICE TASK SWITCH, OTHER SERVICE field of the OMEGAMON accounting report for the requesting DB2 subsystem. It is typically time spent waiting for the network and server to process the request.

- (5) - Server Cls1

  This time is reported in the ELAPSED TIME field under the APPL (CLASS 1) column near the top of the OMEGAMON accounting long trace for the serving DB2 subsystem. It represents the elapsed time from the creation of the database access thread until the termination of the database access thread.

- (6) - Server Cls2

  This time is reported in the ELAPSED TIME field under the DB2 (CLASS 2) column near the top of the OMEGAMON accounting long trace of the serving DB2 subsystem. It represents the elapsed time to process the SQL statements and the commit at the server.

- (7) - Server Cls3

  This time is reported in the TOTAL CLASS 3 field under the CLASS 3 SUSP column near the top of the OMEGAMON accounting long trace for the serving DB2 subsystem. It represents the amount of time the serving DB2 system spent suspended waiting for locks or I/O.

The Class 2 processing time (the TCB time) at the requester does not include processing time at the server. To determine the total Class 2 processing time, add the Class 2 time at the requester to the Class 2 time at the server.

Likewise, add the getpage counts, prefetch counts, locking counts, and I/O counts of the requester to the equivalent counts at the server. For private protocol, SQL activity is counted at both the requester and server. For DRDA, SQL activity is counted only at the server.

```
---- DISTRIBUTED ACTIVITY ------------------------------------------------------------------------
SERVER                  : BOEBDB2SERV      SUCCESSFULLY ALLOC.CONV: C N/A      MSG.IN BUFFER  E :      0
PRODUCT ID              : DB2              CONVERSATION TERMINATED:   N/A
PRODUCT VERSION         : V8 R1 M0         MAX OPEN CONVERSATIONS :   N/A      PREPARE SENT     :      1
METHOD                  : DRDA PROTOCOL    CONT->LIM.BL.FTCH SWCH : D N/A      LASTAGN.SENT     :      0
REQUESTER ELAP.TIME     :   0.685629                                           MESSAGES SENT    :      3
SERVER ELAPSED TIME     :        N/A       COMMIT(2) RESP.RECV.   :     1      MESSAGES RECEIVED:      2
SERVER CPU TIME         :        N/A       BKOUT(2) R.R.          :     0      BYTES SENT       :   9416
DBAT WAITING TIME       :   0.026118       TRANSACT.SENT          :     1      BYTES RECEIVED   :   1497
COMMIT (2) SENT         :          1       COMMT(1)SENT           :     0      BLOCKS RECEIVED  :      0
BACKOUT(2) SENT         :          0       ROLLB(1)SENT           :     0      STMT BOUND AT SER: F  N/A
CONVERSATIONS INITIATED: A         1       SQL SENT               :     0
CONVERSATIONS QUEUED    : B        0       ROWS RECEIVED          :     1      FORGET RECEIVED  :      0
```

Figure 116. DDF block of a requester thread from a OMEGAMON accounting long trace

```
---- DISTRIBUTED ACTIVITY -------------------------------------------------------------------------
REQUESTER          : BOEBDB2REQU         ROLLBK(1) RECEIVED :     0      PREPARE RECEIVED   :     1
PRODUCT ID         : DB2                 SQL RECEIVED       :     0      LAST AGENT RECV.   :     1
PRODUCT VERSION    : V8 R1 M0            COMMIT(2) RESP.SENT:     1      THREADS INDOUBT    :     0
METHOD             : DRDA PROTOCOL       BACKOUT(2)RESP.SENT:     0      MESSAGES.IN BUFFER :     0
COMMIT(2) RECEIVED :              1      BACKOUT(2)PERFORMED:     0      ROWS SENT          :     0
BACKOUT(2) RECEIVED:              0      MESSAGES SENT      :     3      BLOCKS SENT        :     0
COMMIT(2) PERFORMED:              1      MESSAGES RECEIVED  :     5      CONVERSAT.INITIATED:     1
TRANSACTIONS RECV. :              1      BYTES SENT         :   643      FORGET SENT        :     0
COMMITS(1) RECEIVED :             0      BYTES RECEIVED     :  3507
```

Figure 117. DDF block of a server thread from a OMEGAMON accounting long trace

The accounting distributed fields for each serving or requesting location are collected from the viewpoint of this thread communicating with the other location identified. For example, SQL *sent* from the requester is SQL *received* at the server. Do not add together the distributed fields from the requester and the server.

Several fields in the distributed section merit specific attention. The number of conversations is reported in several fields:

- The number of conversation allocations is reported as CONVERSATIONS INITIATED (**A**).
- The number of conversation requests queued during allocation is reported as CONVERSATIONS QUEUED (**B**).
- The number of successful conversation allocations is reported as SUCCESSFULLY ALLOC.CONV (**C**).
- The number of times a switch was made from continuous block fetch to limited block fetch is reported as CONT->LIM.BL.FTCH (**D**). This is only applicable to access that uses DB2 private protocol.

You can use the difference between initiated allocations and successful allocations to identify a session resource constraint problem. If the number of conversations queued is high, or if the number of times a switch was made from continuous to limited block fetch is high, you might want to tune VTAM to increase the number of conversations. VTAM and network parameter definitions are important factors in the performance of DB2 distributed processing. For more information, see *VTAM for MVS/ESA Network Implementation Guide*.

Bytes sent, bytes received, messages sent, and messages received are recorded at both the requester and the server. They provide information on the volume of data transmitted. However, because of the way distributed SQL is processed for private protocol, more bytes may be reported as sent than are reported as received.

To determine the percentage of the rows transmitted by block fetch, compare the total number of rows sent to the number of rows sent in a block fetch buffer, which is reported as MSG.IN BUFFER (**E**). The number of rows sent is reported at the server, and the number of rows received is reported at the requester. Block fetch can significantly affect the number of rows sent across the network.

The number of SQL statements bound for remote access is the number of statements dynamically bound at the server for private protocol. This field is maintained at the requester and is reported as STMT BOUND AT SER (**F**).

Because of the manner in which distributed SQL is processed, the number of rows that are reported might differ slightly from the number of rows that are received. However, a significantly lower number of rows received may indicate that the application did not fetch the entire answer set. This is especially true for access that uses DB2 private protocol.

Reporting server-elapsed time

Client applications that access DB2 data using DRDA access can request that DB2 return the server-elapsed time. server-elapsed time allows remote clients to determine the actual amount of time it takes for DB2 to parse a remote request, process any SQL statements required to satisfy the request, and generate the reply. Because server-elapsed time does not include any of the network time used to receive the request or send the reply, client applications can use the information to quickly isolate poor response times to the network or to the DB2 server without you having to perform traces on the server. Only application clients using DB2 Connect Version 7 or later can request the server-elapsed time. When the System Monitor statement switch has been turned on, DB2 returns server-elapsed time information as a new element through the regular Snapshot Monitoring APIs.

Monitoring distributed processing with RMF

If you use RMF to monitor DDF work, understand how DDF is using the enclave SRBs described in "Using z/OS workload management to set performance objectives" on page 739. The information that is reported using RMF or an equivalent product in the SMF 72 records are the portions of the client's request that are covered by individual enclaves. The way DDF uses enclaves relates directly to whether the DDF thread can become inactive.

Duration of an enclave

"Using threads in INACTIVE MODE for DRDA-only connections" on page 736 describes the difference between threads that are always active and those that can be pooled. If the thread is always active, the duration of the thread is the duration of the enclave. If the thread can be pooled, the following conditions determine the duration of an enclave:

- If the associated package is bound with KEEPDYNAMIC(NO), or there are no open held cursors, or there are active declared temporary tables, the duration of the enclave is the period during which the thread is active.
- If the associated package is bound with KEEPDYNAMIC(YES), and no held cursors or active declared temporary tables exist, and only KEEPDYNAMIC(YES) keeps the thread from being pooled, the duration of the enclave is the period from the beginning to the end of the transaction.

While a thread is pooled, such as during think time, it is not using an enclave. Therefore, SMF 72 record does not report inactive periods.

ACTIVE MODE threads are treated as a single enclave from the time it is created until the time it is terminated. This means that the entire life of the database access thread is reported in the SMF 72 record, regardless of whether SQL work is actually being processed. Figure 118 on page 1018 contrasts the two types of threads.

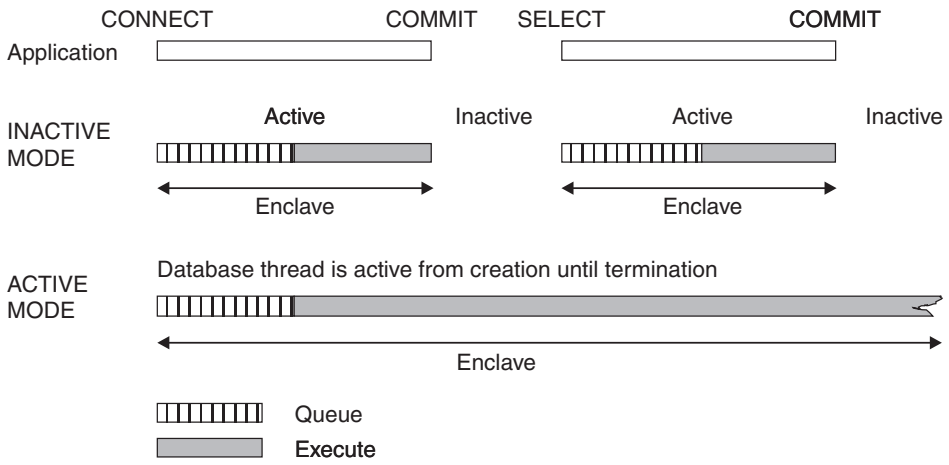


Figure 118. Contrasting ACTIVE MODE threads and POOLED MODE threads

Queue time: Note that the information that is reported back to RMF includes queue time. This particular queue time includes waiting for a new or existing thread to become available.

RMF records for enclaves

The two most frequently used SMF records are types 30 and 72. The type 30 record contains resource consumption at the address space level. You can pull out total enclave usage from the record, but you must use DB2 accounting traces to see resource consumption for a particular enclave.

Each enclave contributes its data to one type 72 record for the service class and to zero or one (0 or 1) type 72 records for the report class. You can use WLM classification rules to separate different enclaves into different service or report classes. Separating the enclaves in this way enables you to understand the DDF work better.

Chapter 37. Monitoring and tuning stored procedures and user-defined functions

Stored procedures that are created in Version 8 of DB2 and all user-defined functions must run in WLM-established address spaces. This chapter primarily presents performance tuning information for user-defined functions and stored procedures in WLM-established address spaces, which is the same for both.

This chapter contains these topics:

- "Controlling address space storage"
- "Assigning procedures and functions to WLM application environments" on page 1020
- "Providing DB2 cost information for accessing user-defined table functions" on page 1021
- "Monitoring stored procedures with the accounting trace" on page 1022
- "Accounting for nested activities" on page 1024

Stored procedures that were created in a release of DB2 prior to Version 8 can run in a DB2-established stored procedures address space. For information about a DB2-established address space and how it compares to WLM-established address spaces, see "Comparing the types of stored procedure address spaces" on page 1025.

Controlling address space storage

To maximize the number of procedures or functions that can run concurrently in a WLM-established stored procedures address space, use the following guidelines:

- Set REGION size for the address spaces to REGION=0 to obtain the largest possible amount of storage below the 16-MB line.
- Limit storage required by application programs below the 16-MB line by:
  - Linking editing programs above the line with AMODE(31) and RMODE(ANY) attributes
  - Using the RES and DATA(31) compiler options for COBOL programs
- Limit storage required by Language Environment by using these run-time options:
  - HEAP(,,ANY) to allocate program heap storage above the 16-MB line
  - STACK(,,ANY,) to allocate program stack storage above the 16-MB line
  - STORAGE(,,,4K) to reduce reserve storage area below the line to 4 KB
  - BELOWHEAP(4K,,) to reduce the heap storage below the line to 4 KB
  - LIBSTACK(4K,,) to reduce the library stack below the line to 4 KB
  - ALL31(ON) to indicate all programs contained in the stored procedure run with AMODE(31) and RMODE(ANY)

Each task control block that runs in a WLM-established stored procedures address space uses approximately 200 KB below the 16-MB line. DB2 needs this storage for stored procedures and user-defined functions because you can create both main programs and subprograms, and DB2 must create an environment for each.

© Copyright IBM Corp. 1982, 2008

A stored procedure can invoke only one utility in one address space at any given time because of the resource requirements of utilities. On the WLM Application-Environment panel, set NUMTCB to 1. See Figure 119 on page 1021. With NUMTCB=1 or NUMTCB being forced to 1, multiple WLM address spaces are created to run each concurrent utility request that comes from a stored procedure call.

Dynamically extending load libraries: Use partitioned data set extended (PDSEs) for load libraries containing stored procedures. Using PDSEs may eliminate your need to stop and start the stored procedures address space due to growth of the load libraries. If a load library grows from additions or replacements, the library may have to be extended.

If you use PDSEs for the load libraries, the new extent information is dynamically updated and you do not need to stop and start the address space. If PDSs are used, load failures may occur because the new extent information is not available.

Assigning procedures and functions to WLM application environments

Workload manager routes work to address spaces based on the application environment name and service class associated with the stored procedure or function. You must use WLM panels to associate an application environment name with the JCL procedure used to start an address space. See *z/OS MVS Planning: Workload Management* for details about workload management panels.

Other tasks must be completed before a stored procedure or user-defined function can run in a WLM-established stored procedures address space. Here is a summary of those tasks:

1. Make sure you have a numeric value specified in the TIMEOUT VALUE field of installation panel DSNTIPX. If you have problems with setting up the environment, this timeout value ensures that your request to execute a stored procedure will not wait for an unlimited amount of time.
2. When you want to move stored procedures from a DB2-established environment to a WLM-established environment, you must link edit them or code them so that they use the Resource Recovery Services attachment facility (RRSAF) instead of the call attachment facility. Use the JCL startup procedure for WLM-established stored procedures address space that was created when you installed or migrated as a model. (The default name is *ssnm*WLM.)

   Unless a particular application environment or caller's service class is not used for a long time, WLM creates on demand at least one address space for each combination of application environment name and caller's service class that is encountered in the workload. For example, if there are five application environment names that each have calling threads with six possible service classes, and all those combinations are in demand, it is possible to have 30 address spaces containing stored procedures or user-defined functions.

   To prevent creating too many address spaces, create a relatively small number of WLM application environments and z/OS service classes.
3. Use the WLM application environment panels to associate the environment name with the JCL procedure. Figure 119 on page 1021 is an example of this panel.

```
  Application-Environment  Notes  Options  Help
-------------------------------------------------------------------------
                     Create an Application Environment
Command ===> ____________________________________________________________

Application Environment Name . : WLMENV2
Description  . . . . . . . . . . Large Stored Proc Env.
Subsystem Type . . . . . . . . . DB2
Procedure Name . . . . . . . . . DSN1WLM
Start Parameters . . . . . . . . DB2SSN=DB2A,NUMTCB=2,APPLENV=WLMENV2
                                 ______________________________________
                                 ____________________________________

Select one of the following options.
1   1.  Multiple server address spaces are allowed.
    2.  Only 1 server address space per MVS system is allowed.
```

Figure 119. WLM panel to create an application environment. You can also use the variable &IWMSSNM for the DB2SSN parameter (DB2SSN=&IWMSSNM). This variable represents the name of the subsystem for which you are starting this address space. This variable is useful for using the same JCL procedure for multiple DB2 subsystems.

4. Specify the WLM application environment name for the WLM_ENVIRONMENT option on CREATE or ALTER PROCEDURE (or FUNCTION) to associate a stored procedure or user-defined function with an application environment.
5. Using the install utility in the WLM application, install the WLM service definition that contains information about this application environment into the couple data set.
6. Activate a WLM policy from the installed service definition.
7. Begin running stored procedures.

Providing DB2 cost information for accessing user-defined table functions

User-defined table functions add additional access cost to the execution of an SQL statement. For DB2 to factor in the effect of user-defined table functions in the selection of the best access path for an SQL statement, the total cost of the user-defined table function must be determined.

The total cost of a table function consists of the following three components:
- The initialization cost that results from the first call processing
- The cost that is associated with acquiring a single row
- The final call cost that performs the clean up processing

These costs, though, are not known to DB2 when I/O costs are added to the CPU cost.

To assist DB2 in determining the cost of user-defined table functions, you can use four fields in SYSIBM.SYSROUTINES. Use the following fields to provide cost information:
- IOS_PER_INVOC for the estimated number of I/Os per row
- INSTS_PER_INVOC for the estimated number of instructions
- INITIAL_IOS for the estimated number of I/Os performed the first and last time the function is invoked
- INITIAL_INSTS for the estimated number of instructions for the first and last time the function is invoked

These values, along with the CARDINALITY value of the table being accessed, are used by DB2 to determine the cost. The results of the calculations can influence such things as the join sequence for a multi-table join and the cost estimates generated for and used in predictive governing.

Determine values for the four fields by examining the source code for the table function. Estimate the I/Os by examining the code executed during the FIRST call and FINAL call. Look for the code executed during the OPEN, FETCH, and CLOSE calls. The costs for the OPEN and CLOSE calls can be amortized over the expected number of rows returned. Estimate the I/O cost by providing the number of I/Os that will be issued. Include the I/Os for any file access.

Calculate the instruction cost by counting the number of high level instructions executed in the user-defined table function and multiplying it by a factor of 20. For assembler programs, the instruction cost is the number of assembler instructions.

If SQL statements are issued within the user-defined table function, use DB2 Estimator to determine the number of instructions and I/Os for the statements. Examining the JES job statistics for a batch program doing equivalent functions can also be helpful. For all fields, a precise number of instructions is not required. Because DB2 already accounts for the costs of invoking table functions, these costs should not be included in the estimates.

Example: The following statement shows how these fields can be updated. The authority to update is the same authority as that required to update any catalog statistics column.

```
UPDATE SYSIBM.SYSROUTINES SET
   IOS_PER_INVOC   = 0.0,
   INSTS_PER_INVOC = 4.5E3,
   INITIAL_IOS     = 2.0
   INITIAL_INSTS   = 1.0E4,
   CARDINALITY     = 5E3
WHERE
   SCHEMA = 'SYSADM' AND
   SPECIFICNAME = 'FUNCTION1' AND
   ROUTINETYPE = 'F';
```

Monitoring stored procedures with the accounting trace

Through a stored procedure, one SQL statement generates other SQL statements under the same thread. The processing that is performed by the stored procedure is included in class 1 and class 2 times for accounting.

The accounting report on the server has several fields that specifically relate to stored procedures processing, as shown in Figure 120 on page 1023.

```
AVERAGE       APPL(CL.1)  DB2 (CL.2)  IFI (CL.5)    CLASS 3 SUSPENSIONS   AVERAGE TIME  AV.EVENT    HIGHLIGHTS
------------  ----------  ----------  ----------    --------------------  ------------  --------    --------------------------
ELAPSED TIME    0.072929    0.029443         N/P    LOCK/LATCH(DB2+IRLM)      0.000011      0.04    #OCCURRENCES    :      193
 NONNESTED      0.072929    0.029443         N/A    SYNCHRON. I/O             0.010170      9.16    #ALLIEDS        :        0
 STORED PROC A  0.000000    0.000000         N/A     DATABASE I/O             0.006325      8.16    #ALLIEDS DISTRIB:        0
 UDF            0.000000    0.000000         N/A     LOG WRITE I/O            0.003845      1.00    #DBATS          :      193
 TRIGGER        0.000000    0.000000         N/A    OTHER READ I/O            0.000000      0.00    #DBATS DISTRIB. :        0
                                                    OTHER WRTE I/O            0.000148      0.04    #NO PROGRAM DATA:        0
CPU TIME        0.026978    0.018994         N/P    SER.TASK SWTCH            0.000115      0.04    #NORMAL TERMINAT:      193
 AGENT          0.026978    0.018994         N/A     UPDATE COMMIT            0.000000      0.00    #ABNORMAL TERMIN:        0
  NONNESTED     0.026978    0.018994         N/P     OPEN/CLOSE               0.000000      0.00    #CP/X PARALLEL. :        0
  STORED PRC    0.000000    0.000000         N/A     SYSLGRNG REC             0.000115      0.04    #IO PARALLELISM :        0
  UDF           0.000000    0.000000         N/A     EXT/DEL/DEF              0.000000      0.00    #INCREMENT. BIND:        0
  TRIGGER       0.000000    0.000000         N/A     OTHER SERVICE            0.000000      0.00    #COMMITS        :      193
 PAR.TASKS      0.000000    0.000000         N/A    ARC.LOG(QUIES)            0.000000      0.00    #ROLLBACKS      :        0
                                                    ARC.LOG READ              0.000000      0.00    #SVPT REQUESTS  :        0
SUSPEND TIME    0.000000    0.010444         N/A    DRAIN LOCK                0.000000      0.00    #SVPT RELEASE   :        0
 AGENT               N/A    0.010444         N/A    CLAIM RELEASE             0.000000      0.00    #SVPT ROLLBACK  :        0
 PAR.TASKS           N/A    0.000000         N/A    PAGE LATCH                0.000000      0.00    MAX SQL CASC LVL:        0
 STORED PROC B  0.000000         N/A         N/A    NOTIFY MSGS               0.000000      0.00    UPDATE/COMMIT   :    40.00
 UDF            0.000000         N/A         N/A    GLOBAL CONTENTION         0.000000      0.00    SYNCH I/O AVG.  : 0.001110
                                                    COMMIT PH1 WRITE I/O      0.000000      0.00
NOT ACCOUNT.         N/A    0.000004         N/A    ASYNCH CF REQUESTS        0.000000      0.00
DB2 ENT/EXIT         N/A      182.00         N/A    TOTAL CLASS 3             0.010444      9.28
EN/EX-STPROC         N/A        0.00         N/A
EN/EX-UDF            N/A        0.00         N/A
DCAPT.DESCR.         N/A         N/A         N/P
LOG EXTRACT.         N/A         N/A         N/P

⋮

STORED PROCEDURES   AVERAGE     TOTAL
-----------------  --------  --------
CALL STATEMENTS C      0.00         0
ABENDED                0.00         0
TIMED OUT D            0.00         0
REJECTED               0.00         0

⋮
```

Figure 120. Partial long accounting report for stored procedures

Descriptions of fields:

- The part of the total CPU time that was spent satisfying stored procedures requests is indicated in **A**.
- The amount of time spent waiting for a stored procedure to be scheduled and the time that is needed to return control to DB2 after the stored procedure has completed is indicated in **B**.
- The number of calls to stored procedures is indicated in **C**.
- The number of times a stored procedure timed out waiting to be scheduled is shown in **D**.

What to do for excessive timeouts or wait time: If you have excessive wait time (**B**) or timeouts (**D**) for user-defined functions or stored procedures, the possible causes include:

- The goal of the service class that is assigned to the WLM stored procedure's address space, as it was initially started, is not high enough. The address space uses this goal to honor requests to start processing stored procedures.
- The priority of the service class that is running the stored procedure is not high enough.
- Make sure that the application environment is available by using the z/OS command DISPLAY WLM,APPLENV=*applenv*. If the application environment is quiesced, WLM does not start any address spaces for that environment; CALL statements are queued or rejected.

Accounting for nested activities

The accounting class 1 and class 2 CPU and elapsed times for triggers, stored procedures, and user-defined functions are accumulated in separate fields and exclude any time accumulated in other nested activity. These CPU and elapsed times are accumulated for each category during the execution of each agent until agent deallocation. Package accounting can be used to break out accounting data for execution of individual stored procedures, user-defined functions, or triggers. Figure 121 shows an agent that executes multiple types of DB2 nested activities.

```
Time   Application   DB2                          Stored procedure            User-defined function
-----  ------------  ----------------------------  --------------------------  --------------------------
 T0    Code
 T1    SQL---------->
 T2       <---------
 T3    SQL---------->
 T4                  Trigger
 T5                  SQL
 T6                  CALL triggered--------------->
 T7                                                Stored procedure code
 T8                                 <-------------SQL
 T9                                 ------------->Stored procedure code
 T10                                <-------------SQL(User-defined function)
 T11                 Start User-defined function
 T12                 ------------------------------------------------------->User-defined function code
 T13                 <-------------------------------------------------------SQL
 T14                 ------------------------------------------------------->User-defined function code
 T16                 <-------------------------------------------------------User-defined function ends
 T17                 Back to Stored procedure----->Stored procedure code
 T18                 SQL            <-------------Back to trigger
 T19                 Trigger ends
 T20  Code<----------Return to Application
 T21  End
```

Figure 121. Time spent executing nested activities

Table 202 shows the formula used to determine time for nested activities.

Table 202. Sample for time used for execution of nested activities

Count for	Formula	Class
Application elapsed	T21-T0	1
Application task control block (TU)	T21-T0	1
Application in DB2 elapsed	T2-T1 + T4-T3 + T20-T19	2
Application in DB2 task control block (TU)	T2-T1 + T4-T3 + T20-T19	2
Trigger in DB2 elapsed	T6-T4 + T19-T18	2
Trigger in DB2 task control block (TU)	T6-T4 + T19-T18	2
Wait for STP time	T7-T6 + T18–T17	3
Stored procedure elapsed	T11-T6 + T18-T16	1
Stored procedure task control block (TU)	T11-T6 + T18-T16	1
Stored procedure SQL elapsed	T9-T8 + T11-T10 + T17-16	2
Stored procedure SQL elapsed	T9-T8 + T11-T10 + T17-T16	2

Table 202. Sample for time used for execution of nested activities (continued)

Count for	Formula	Class
Wait for user-defined function time	T12-T11	3
User-defined function elapsed	T16-T11	1
User-defined function task control block (TU)	T16-T11	1
User-defined function SQL elapsed	T14-T13	2
User-defined function SQL task control block (TU)	T14-T13	2
Note: TU = time used.		

The total class 2 time is the total of the ″in DB2″ times for the application, trigger, stored procedure, and user-defined function. The class 1 ″wait″ times for the stored procedures and user-defined functions need to be added to the total class 3 times.

Comparing the types of stored procedure address spaces

Table 203 summarizes the differences between stored procedures that run in WLM-established stored procedures address spaces and those that run in a DB2-established stored procedures address space. You cannot run any user-defined functions in a DB2-established stored procedures address space. Only those stored procedures that were created in a release of DB2 prior to Version 8 can be run in a DB2-established address space.

Table 203. Comparing WLM-established and DB2-established stored procedures

DB2-established	WLM-established	More information
Use a single address space for stored procedures: • A failure in one stored procedure can affect other stored procedures that are running in that address space. • Can be difficult to support more than 50 stored procedures running at the same time because of storage that language products need below the 16MB line.	Use many address spaces for stored procedures and user-defined functions: • Possible to isolate procedures and functions from one another so that failures do not affect others that are running in other address spaces. • Reduces demand for storage below the 16MB line and thereby removes the limitation on the number of procedures and functions that can run concurrently. • Only one utility can be invoked by a stored procedure in one address space at any given time. The start parameter NUMTCB on the WLM Application-Environment panel has to be set to 1.	“Controlling address space storage” on page 1019and Figure 119 on page 1021
Incoming requests for stored procedures are handled in a first-in, first-out order.	Requests are handled in priority order.	“Using z/OS workload management to set performance objectives” on page 739
Stored procedures run at the priority of the stored procedures address space.	Stored procedures inherit the z/OS dispatching priority of the DB2 thread that issues the CALL statement. User-defined functions inherit the priority of the DB2 thread that invoked the procedure.	“Using z/OS workload management to set performance objectives” on page 739

Table 203. Comparing WLM-established and DB2-established stored procedures (continued)

DB2-established	WLM-established	More information
No ability to customize the environment.	Each address space is associated with a WLM *application environment* that you specify. An application environment is an attribute that you associate on the CREATE statement for the function or procedure. The environment determines which JCL procedure is used to run a particular stored procedure.	"Assigning procedures and functions to WLM application environments" on page 1020
Must run as a MAIN program.	Can run as a MAIN or SUB program. SUB programs can run significantly faster, but the subprogram must do more initialization and cleanup processing itself rather than relying on Language Environment to handle that.	*DB2 Application Programming and SQL Guide*
You can access non-relational data, but that data is not included in your SQL unit of work. It is a separate unit of work.	You can access non-relational data. If the non-relational data is managed by RRS, the updates to that data are part of your SQL unit of work.	*DB2 Application Programming and SQL Guide*
Stored procedures access protected z/OS resources with the authority of the stored procedures address space.	Procedures or functions can access protected z/OS resources with one of three authorities, as specified on the SECURITY option of the CREATE FUNCTION or CREATE PROCEDURE statement: • The authority of the WLM-established address space (SECURITY=DB2) • The authority of the invoker of the stored procedure or user-defined function (SECURITY=USER) • The authority of the definer of the stored procedure or user-defined function (SECURITY=DEFINER)	*DB2 Administration Guide*

The tuning information for stored procedures in a DB2-established address space is very similar to the information for user-defined functions and stored procedures in a WLM-established address space:

- You can control address space storage as described in "Controlling address space storage" on page 1019. Each task control block that runs in the DB2-established stored procedures address space requires approximately 100 KB below the 16MB line.
- As described in "Monitoring stored procedures with the accounting trace" on page 1022 "Accounting for nested activities" on page 1024, the processing done by a stored procedure running in a DB2-established address spaces is included in DB2's class 1 and class 2 accounting times.

  Possible causes of excessive wait time include:

  – Someone issued the DB2 command STOP PROCEDURE ACTION(QUEUE) that caused requests to queue up for a long time and time out.

  – The stored procedures are hanging onto the *ssnm*SPAS task control blocks for too long. In this case, you need to find out why this is happening.

    If you get many DB2 lock suspensions, you might have too many *ssnm*SPAS task control blocks, causing them to encounter too many lock conflicts with one another. You might need to make code changes to your application or change your database design to reduce the number of lock suspensions.

– If the stored procedures are starting and stopping quickly, you might not have enough *ssnm*SPAS task control blocks to handle the work load. In this case, increase the number on field NUMBER OF TCBS on installation panel DSNTIPX.

Part 6. Appendixes

© Copyright IBM Corp. 1982, 2008

Appendix A. DB2 sample tables

The information in this appendix is General-use Programming Interface and Associated Guidance Information as defined in “Notices” on page 1301.

Most of the examples in this book refer to the tables described in this appendix. As a group, the tables include information that describes employees, departments, projects, and activities, and make up a sample application that exemplifies most of the features of DB2. The sample storage group, databases, tablespaces, tables, and views are created when you run the installation sample jobs DSNTEJ1 and DSNTEJ7. DB2 sample objects that include LOBs are created in job DSNTEJ7. All other sample objects are created in job DSNTEJ1. The CREATE INDEX statements for the sample tables are not shown here; they, too, are created by the DSNTEJ1 and DSNTEJ7 sample jobs.

Authorization on all sample objects is given to PUBLIC in order to make the sample programs easier to run. The contents of any table can easily be reviewed by executing an SQL statement, for example SELECT * FROM DSN8810.PROJ. For convenience in interpreting the examples, the department and employee tables are listed here in full.

The following topics provide additional information:

- “Activity table (DSN8810.ACT)”
- “Department table (DSN8810.DEPT)” on page 1032
- “Employee table (DSN8810.EMP)” on page 1034
- “Employee photo and resume table (DSN8810.EMP_PHOTO_RESUME)” on page 1037
- “Project table (DSN8810.PROJ)” on page 1038
- “Project activity table (DSN8810.PROJACT)” on page 1039
- “Employee to project activity table (DSN8810.EMPPROJACT)” on page 1040
- “Unicode sample table (DSN8810.DEMO_UNICODE)” on page 1041
- “Relationships among the sample tables” on page 1042
- “Views on the sample tables” on page 1042
- “Storage of sample application tables” on page 1047

Activity table (DSN8810.ACT)

The activity table describes the activities that can be performed during a project. The table resides in database DSN8D81A and is created with the following statement:

```
CREATE TABLE DSN8810.ACT
      (ACTNO    SMALLINT       NOT NULL,
       ACTKWD   CHAR(6)        NOT NULL,
       ACTDESC  VARCHAR(20)    NOT NULL,
       PRIMARY KEY (ACTNO)             )
  IN DSN8D81A.DSN8S81P
  CCSID EBCDIC;
```

Content of the activity table:

© Copyright IBM Corp. 1982, 2008

Table 204 shows the content of the columns.

Table 204. Columns of the activity table

Column	Column Name	Description
1	ACTNO	Activity ID (the primary key)
2	ACTKWD	Activity keyword (up to six characters)
3	ACTDESC	Activity description

The activity table has these indexes:

Table 205. Indexes of the activity table

Name	On Column	Type of Index
DSN8810.XACT1	ACTNO	Primary, ascending
DSN8810.XACT2	ACTKWD	Unique, ascending

Relationship to other tables:

The activity table is a parent table of the project activity table, through a foreign key on column ACTNO.

Department table (DSN8810.DEPT)

The department table describes each department in the enterprise and identifies its manager and the department to which it reports.

The table, shown in Table 208 on page 1033, resides in table space DSN8D81A.DSN8S81D and is created with the following statement:

```
CREATE TABLE DSN8810.DEPT
      (DEPTNO    CHAR(3)          NOT NULL,
       DEPTNAME  VARCHAR(36)      NOT NULL,
       MGRNO     CHAR(6)                  ,
       ADMRDEPT  CHAR(3)          NOT NULL,
       LOCATION  CHAR(16)                 ,
       PRIMARY KEY (DEPTNO)               )
  IN DSN8D81A.DSN8S81D
  CCSID EBCDIC;
```

Because the table is self-referencing, and also is part of a cycle of dependencies, its foreign keys must be added later with these statements:

```
ALTER TABLE DSN8810.DEPT
      FOREIGN KEY RDD (ADMRDEPT) REFERENCES DSN8810.DEPT
              ON DELETE CASCADE;

ALTER TABLE DSN8810.DEPT
      FOREIGN KEY RDE (MGRNO) REFERENCES DSN8810.EMP
              ON DELETE SET NULL;
```

Content of the department table:

Table 206 shows the content of the columns.

Table 206. Columns of the department table

Column	Column Name	Description
1	DEPTNO	Department ID, the primary key

Table 206. Columns of the department table (continued)

Column	Column Name	Description
2	DEPTNAME	A name describing the general activities of the department
3	MGRNO	Employee number (EMPNO) of the department manager
4	ADMRDEPT	ID of the department to which this department reports; the department at the highest level reports to itself
5	LOCATION	The remote location name

Table 207 shows the indexes of the department table:

Table 207. Indexes of the department table

Name	On Column	Type of Index
DSN8810.XDEPT1	DEPTNO	Primary, ascending
DSN8810.XDEPT2	MGRNO	Ascending
DSN8810.XDEPT3	ADMRDEPT	Ascending

Table 208 shows the content of the department table:

Table 208. DSN8810.DEPT: department table

DEPTNO	DEPTNAME	MGRNO	ADMRDEPT	LOCATION
A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00	----------
B01	PLANNING	000020	A00	----------
C01	INFORMATION CENTER	000030	A00	----------
D01	DEVELOPMENT CENTER	------	A00	----------
E01	SUPPORT SERVICES	000050	A00	----------
D11	MANUFACTURING SYSTEMS	000060	D01	----------
D21	ADMINISTRATION SYSTEMS	000070	D01	----------
E11	OPERATIONS	000090	E01	----------
E21	SOFTWARE SUPPORT	000100	E01	----------
F22	BRANCH OFFICE F2	------	E01	----------
G22	BRANCH OFFICE G2	------	E01	----------
H22	BRANCH OFFICE H2	------	E01	----------
I22	BRANCH OFFICE I2	------	E01	----------
J22	BRANCH OFFICE J2	------	E01	----------

The LOCATION column contains nulls until sample job DSNTEJ6 updates this column with the location name.

Relationship to other tables:

The table is self-referencing: the value of the administering department must be a department ID.

The table is a parent table of:

- The employee table, through a foreign key on column WORKDEPT
- The project table, through a foreign key on column DEPTNO.

It is a dependent of the employee table, through its foreign key on column MGRNO.

Employee table (DSN8810.EMP)

The employee table identifies all employees by an employee number and lists basic personnel information.

The table shown in Table 211 on page 1035 and Table 212 on page 1036 resides in the partitioned table space DSN8D81A.DSN8S81E. Because it has a foreign key referencing DEPT, that table and the index on its primary key must be created first. Then EMP is created with the following statement:

```
CREATE TABLE DSN8810.EMP
      (EMPNO     CHAR(6)                        NOT NULL,
       FIRSTNME  VARCHAR(12)                    NOT NULL,
       MIDINIT   CHAR(1)                        NOT NULL,
       LASTNAME  VARCHAR(15)                    NOT NULL,
       WORKDEPT  CHAR(3)                                ,
       PHONENO   CHAR(4)         CONSTRAINT NUMBER CHECK
                 (PHONENO >= '0000' AND
                  PHONENO <= '9999')                    ,
       HIREDATE  DATE                                   ,
       JOB       CHAR(8)                                ,
       EDLEVEL   SMALLINT                               ,
       SEX       CHAR(1)                                ,
       BIRTHDATE DATE                                   ,
       SALARY    DECIMAL(9,2)                           ,
       BONUS     DECIMAL(9,2)                           ,
       COMM      DECIMAL(9,2)                           ,
       PRIMARY KEY (EMPNO)                              ,
       FOREIGN KEY RED (WORKDEPT) REFERENCES DSN8810.DEPT
                 ON DELETE SET NULL                    )
  EDITPROC  DSN8EAE1
  IN DSN8D81A.DSN8S81E
  CCSID EBCDIC;
```

Content of the employee table:

Table 209 shows the content of the columns. The table has a check constraint, NUMBER, which checks that the phone number is in the numeric range 0000 to 9999.

Table 209. Columns of the employee table

Column	Column Name	Description
1	EMPNO	Employee number (the primary key)
2	FIRSTNME	First name of employee
3	MIDINIT	Middle initial of employee
4	LASTNAME	Last name of employee
5	WORKDEPT	ID of department in which the employee works
6	PHONENO	Employee telephone number
7	HIREDATE	Date of hire
8	JOB	Job held by the employee
9	EDLEVEL	Number of years of formal education

Table 209. Columns of the employee table (continued)

Column	Column Name	Description
10	SEX	Sex of the employee (M or F)
11	BIRTHDATE	Date of birth
12	SALARY	Yearly salary in dollars
13	BONUS	Yearly bonus in dollars
14	COMM	Yearly commission in dollars

Table 210 shows the indexes of the employee table:

Table 210. Indexes of the employee table

Name	On Column	Type of Index
DSN8810.XEMP1	EMPNO	Primary, partitioned, ascending
DSN8810.XEMP2	WORKDEPT	Ascending

Table 211 and Table 212 on page 1036 show the content of the employee table:

Table 211. Left half of DSN8810.EMP: employee table. Note that a blank in the MIDINIT column is an actual value of " " rather than null.

EMPNO	FIRSTNME	MIDINIT	LASTNAME	WORKDEPT	PHONENO	HIREDATE
000010	CHRISTINE	I	HAAS	A00	3978	1965-01-01
000020	MICHAEL	L	THOMPSON	B01	3476	1973-10-10
000030	SALLY	A	KWAN	C01	4738	1975-04-05
000050	JOHN	B	GEYER	E01	6789	1949-08-17
000060	IRVING	F	STERN	D11	6423	1973-09-14
000070	EVA	D	PULASKI	D21	7831	1980-09-30
000090	EILEEN	W	HENDERSON	E11	5498	1970-08-15
000100	THEODORE	Q	SPENSER	E21	0972	1980-06-19
000110	VINCENZO	G	LUCCHESSI	A00	3490	1958-05-16
000120	SEAN		O'CONNELL	A00	2167	1963-12-05
000130	DOLORES	M	QUINTANA	C01	4578	1971-07-28
000140	HEATHER	A	NICHOLLS	C01	1793	1976-12-15
000150	BRUCE		ADAMSON	D11	4510	1972-02-12
000160	ELIZABETH	R	PIANKA	D11	3782	1977-10-11
000170	MASATOSHI	J	YOSHIMURA	D11	2890	1978-09-15
000180	MARILYN	S	SCOUTTEN	D11	1682	1973-07-07
000190	JAMES	H	WALKER	D11	2986	1974-07-26
000200	DAVID		BROWN	D11	4501	1966-03-03
000210	WILLIAM	T	JONES	D11	0942	1979-04-11
000220	JENNIFER	K	LUTZ	D11	0672	1968-08-29
000230	JAMES	J	JEFFERSON	D21	2094	1966-11-21
000240	SALVATORE	M	MARINO	D21	3780	1979-12-05
000250	DANIEL	S	SMITH	D21	0961	1969-10-30
000260	SYBIL	P	JOHNSON	D21	8953	1975-09-11
000270	MARIA	L	PEREZ	D21	9001	1980-09-30
000280	ETHEL	R	SCHNEIDER	E11	8997	1967-03-24
000290	JOHN	R	PARKER	E11	4502	1980-05-30
000300	PHILIP	X	SMITH	E11	2095	1972-06-19
000310	MAUDE	F	SETRIGHT	E11	3332	1964-09-12
000320	RAMLAL	V	MEHTA	E21	9990	1965-07-07
000330	WING		LEE	E21	2103	1976-02-23
000340	JASON	R	GOUNOT	E21	5698	1947-05-05

Table 211. Left half of DSN8810.EMP: employee table (continued). Note that a blank in the MIDINIT column is an actual value of ″ ″ rather than null.

EMPNO	FIRSTNME	MIDINIT	LASTNAME	WORKDEPT	PHONENO	HIREDATE
200010	DIAN	J	HEMMINGER	A00	3978	1965-01-01
200120	GREG		ORLANDO	A00	2167	1972-05-05
200140	KIM	N	NATZ	C01	1793	1976-12-15
200170	KIYOSHI		YAMAMOTO	D11	2890	1978-09-15
200220	REBA	K	JOHN	D11	0672	1968-08-29
200240	ROBERT	M	MONTEVERDE	D21	3780	1979-12-05
200280	EILEEN	R	SCHWARTZ	E11	8997	1967-03-24
200310	MICHELLE	F	SPRINGER	E11	3332	1964-09-12
200330	HELENA		WONG	E21	2103	1976-02-23
200340	ROY	R	ALONZO	E21	5698	1947-05-05

Table 212. Right half of DSN8810.EMP: employee table

(EMPNO)	JOB	EDLEVEL	SEX	BIRTHDATE	SALARY	BONUS	COMM
(000010)	PRES	18	F	1933-08-14	52750.00	1000.00	4220.00
(000020)	MANAGER	18	M	1948-02-02	41250.00	800.00	3300.00
(000030)	MANAGER	20	F	1941-05-11	38250.00	800.00	3060.00
(000050)	MANAGER	16	M	1925-09-15	40175.00	800.00	3214.00
(000060)	MANAGER	16	M	1945-07-07	32250.00	600.00	2580.00
(000070)	MANAGER	16	F	1953-05-26	36170.00	700.00	2893.00
(000090)	MANAGER	16	F	1941-05-15	29750.00	600.00	2380.00
(000100)	MANAGER	14	M	1956-12-18	26150.00	500.00	2092.00
(000110)	SALESREP	19	M	1929-11-05	46500.00	900.00	3720.00
(000120)	CLERK	14	M	1942-10-18	29250.00	600.00	2340.00
(000130)	ANALYST	16	F	1925-09-15	23800.00	500.00	1904.00
(000140)	ANALYST	18	F	1946-01-19	28420.00	600.00	2274.00
(000150)	DESIGNER	16	M	1947-05-17	25280.00	500.00	2022.00
(000160)	DESIGNER	17	F	1955-04-12	22250.00	400.00	1780.00
(000170)	DESIGNER	16	M	1951-01-05	24680.00	500.00	1974.00
(000180)	DESIGNER	17	F	1949-02-21	21340.00	500.00	1707.00
(000190)	DESIGNER	16	M	1952-06-25	20450.00	400.00	1636.00
(000200)	DESIGNER	16	M	1941-05-29	27740.00	600.00	2217.00
(000210)	DESIGNER	17	M	1953-02-23	18270.00	400.00	1462.00
(000220)	DESIGNER	18	F	1948-03-19	29840.00	600.00	2387.00
(000230)	CLERK	14	M	1935-05-30	22180.00	400.00	1774.00
(000240)	CLERK	17	M	1954-03-31	28760.00	600.00	2301.00
(000250)	CLERK	15	M	1939-11-12	19180.00	400.00	1534.00
(000260)	CLERK	16	F	1936-10-05	17250.00	300.00	1380.00
(000270)	CLERK	15	F	1953-05-26	27380.00	500.00	2190.00
(000280)	OPERATOR	17	F	1936-03-28	26250.00	500.00	2100.00
(000290)	OPERATOR	12	M	1946-07-09	15340.00	300.00	1227.00
(000300)	OPERATOR	14	M	1936-10-27	17750.00	400.00	1420.00
(000310)	OPERATOR	12	F	1931-04-21	15900.00	300.00	1272.00
(000320)	FIELDREP	16	M	1932-08-11	19950.00	400.00	1596.00
(000330)	FIELDREP	14	M	1941-07-18	25370.00	500.00	2030.00
(000340)	FIELDREP	16	M	1926-05-17	23840.00	500.00	1907.00
(200010)	SALESREP	18	F	1933-08-14	46500.00	1000.00	4220.00
(200120)	CLERK	14	M	1942-10-18	29250.00	600.00	2340.00
(200140)	ANALYST	18	F	1946-01-19	28420.00	600.00	2274.00
(200170)	DESIGNER	16	M	1951-01-05	24680.00	500.00	1974.00
(200220)	DESIGNER	18	F	1948-03-19	29840.00	600.00	2387.00
(200240)	CLERK	17	M	1954-03-31	28760.00	600.00	2301.00
(200280)	OPERATOR	17	F	1936-03-28	26250.00	500.00	2100.00
(200310)	OPERATOR	12	F	1931-04-21	15900.00	300.00	1272.00

Table 212. Right half of DSN8810.EMP: employee table (continued)

(EMPNO)	JOB	EDLEVEL	SEX	BIRTHDATE	SALARY	BONUS	COMM
(200330)	FIELDREP	14	F	1941-07-18	25370.00	500.00	2030.00
(200340)	FIELDREP	16	M	1926-05-17	23840.00	500.00	1907.00

Relationship to other tables:

The table is a parent table of:
- The department table, through a foreign key on column MGRNO
- The project table, through a foreign key on column RESPEMP.

It is a dependent of the department table, through its foreign key on column WORKDEPT.

Employee photo and resume table (DSN8810.EMP_PHOTO_RESUME)

The employee photo and resume table complements the employee table. Each row of the photo and resume table contains a photo of the employee, in two formats, and the employee"s resume. The photo and resume table resides in table space DSN8D81A.DSN8S81E. The following statement creates the table:

```
CREATE TABLE  DSN8810.EMP_PHOTO_RESUME
              (EMPNO      CHAR(06) NOT NULL,
               EMP_ROWID  ROWID NOT NULL GENERATED ALWAYS,
               PSEG_PHOTO BLOB(500K),
               BMP_PHOTO  BLOB(100K),
               RESUME        CLOB(5K))
               PRIMARY KEY  (EMPNO)
       IN  DSN8D81L.DSN8S81B
       CCSID  EBCDIC;
```

DB2 requires an auxiliary table for each LOB column in a table. These statements define the auxiliary tables for the three LOB columns in DSN8810.EMP_PHOTO_RESUME:

```
CREATE AUX TABLE DSN8810.AUX_BMP_PHOTO
               IN DSN8D81L.DSN8S81M
               STORES DSN8810.EMP_PHOTO_RESUME
               COLUMN BMP_PHOTO;

CREATE AUX TABLE DSN8810.AUX_PSEG_PHOTO
               IN DSN8D81L.DSN8S81L
               STORES DSN8810.EMP_PHOTO_RESUME
               COLUMN PSEG_PHOTO;

CREATE AUX TABLE DSN8810.AUX_EMP_RESUME
               IN DSN8D81L.DSN8S81N
               STORES DSN8810.EMP_PHOTO_RESUME
               COLUMN RESUME;
```

Content of the employee photo and resume table:

Table 213 shows the content of the columns.

Table 213. Columns of the employee photo and resume table

Column	Column Name	Description
1	EMPNO	Employee ID (the primary key)

Table 213. Columns of the employee photo and resume table (continued)

Column	Column Name	Description
2	EMP_ROWID	Row ID to uniquely identify each row of the table. DB2 supplies the values of this column.
3	PSEG_PHOTO	Employee photo, in PSEG format
4	BMP_PHOTO	Employee photo, in BMP format
5	RESUME	Employee resume

Table 214 shows the indexes for the employee photo and resume table:

Table 214. Indexes of the employee photo and resume table

Name	On Column	Type of Index
DSN8810.XEMP_PHOTO_RESUME	EMPNO	Primary, ascending

Table 215 shows the indexes for the auxiliary tables for the employee photo and resume table:

Table 215. Indexes of the auxiliary tables for the employee photo and resume table

Name	On Table	Type of Index
DSN8810.XAUX_BMP_PHOTO	DSN8810.AUX_BMP_PHOTO	Unique
DSN8810.XAUX_PSEG_PHOTO	DSN8810.AUX_PSEG_PHOTO	Unique
DSN8810.XAUX_EMP_RESUME	DSN8810.AUX_EMP_RESUME	Unique

Relationship to other tables:

The table is a parent table of the project table, through a foreign key on column RESPEMP.

Project table (DSN8810.PROJ)

The project table describes each project that the business is currently undertaking. Data contained in each row include the project number, name, person responsible, and schedule dates.

The table resides in database DSN8D81A. Because it has foreign keys referencing DEPT and EMP, those tables and the indexes on their primary keys must be created first. Then PROJ is created with the following statement:

```
CREATE TABLE DSN8810.PROJ
               (PROJNO   CHAR(6) PRIMARY KEY NOT NULL,
                PROJNAME VARCHAR(24)    NOT NULL WITH DEFAULT
                  'PROJECT NAME UNDEFINED',
                DEPTNO   CHAR(3)        NOT NULL REFERENCES
                  DSN8810.DEPT ON DELETE RESTRICT,
                RESPEMP  CHAR(6)        NOT NULL REFERENCES
                  DSN8810.EMP ON DELETE RESTRICT,
                PRSTAFF  DECIMAL(5, 2)          ,
                PRSTDATE DATE                   ,
                PRENDATE DATE                   ,
                MAJPROJ  CHAR(6))
        IN DSN8D81A.DSN8S81P
        CCSID EBCDIC;
```

Because the table is self-referencing, the foreign key for that restraint must be added later with:

```
ALTER TABLE DSN8810.PROJ
      FOREIGN KEY RPP (MAJPROJ) REFERENCES DSN8810.PROJ
              ON DELETE CASCADE;
```

Content of the project table:

Table 216 shows the content of the columns.

Table 216. Columns of the project table

Column	Column Name	Description
1	PROJNO	Project ID (the primary key)
2	PROJNAME	Project name
3	DEPTNO	ID of department responsible for the project
4	RESPEMP	ID of employee responsible for the project
5	PRSTAFF	Estimated mean number of persons needed between PRSTDATE and PRENDATE to achieve the whole project, including any subprojects
6	PRSTDATE	Estimated project start date
7	PRENDATE	Estimated project end date
8	MAJPROJ	ID of any project of which this project is a part

Table 217 shows the indexes for the project table:

Table 217. Indexes of the project table

Name	On Column	Type of Index
DSN8810.XPROJ1	PROJNO	Primary, ascending
DSN8810.XPROJ2	RESPEMP	Ascending

Relationship to other tables:

The table is self-referencing: a nonnull value of MAJPROJ must be a project number. The table is a parent table of the project activity table, through a foreign key on column PROJNO. It is a dependent of:

- The department table, through its foreign key on DEPTNO
- The employee table, through its foreign key on RESPEMP.

Project activity table (DSN8810.PROJACT)

The project activity table lists the activities performed for each project. The table resides in database DSN8D81A. Because it has foreign keys referencing PROJ and ACT, those tables and the indexes on their primary keys must be created first. Then PROJACT is created with the following statement:

```
CREATE TABLE DSN8810.PROJACT
      (PROJNO    CHAR(6)                        NOT NULL,
       ACTNO     SMALLINT                       NOT NULL,
       ACSTAFF   DECIMAL(5,2)                           ,
       ACSTDATE  DATE                           NOT NULL,
       ACENDATE  DATE                                   ,
       PRIMARY KEY (PROJNO, ACTNO, ACSTDATE),
       FOREIGN KEY RPAP (PROJNO) REFERENCES DSN8810.PROJ
```

```
                                    ON DELETE RESTRICT,
      FOREIGN KEY RPAA (ACTNO) REFERENCES DSN8810.ACT
                                    ON DELETE RESTRICT)
  IN DSN8D81A.DSN8S81P
  CCSID EBCDIC;
```

Content of the project activity table:

Table 218 shows the content of the columns.

Table 218. Columns of the project activity table

Column	Column Name	Description
1	PROJNO	Project ID
2	ACTNO	Activity ID
3	ACSTAFF	Estimated mean number of employees needed to staff the activity
4	ACSTDATE	Estimated activity start date
5	ACENDATE	Estimated activity completion date

Table 219 shows the index of the project activity table:

Table 219. Index of the project activity table

Name	On Columns	Type of Index
DSN8810.XPROJAC1	PROJNO, ACTNO, ACSTDATE	primary, ascending

Relationship to other tables:

The table is a parent table of the employee to project activity table, through a foreign key on columns PROJNO, ACTNO, and EMSTDATE. It is a dependent of:
- The activity table, through its foreign key on column ACTNO
- The project table, through its foreign key on column PROJNO

Employee to project activity table (DSN8810.EMPPROJACT)

The employee to project activity table identifies the employee who performs an activity for a project, tells the proportion of the employee's time required, and gives a schedule for the activity.

The table resides in database DSN8D81A. Because it has foreign keys referencing EMP and PROJACT, those tables and the indexes on their primary keys must be created first. Then EMPPROJACT is created with the following statement:

```
CREATE TABLE DSN8810.EMPPROJACT
      (EMPNO     CHAR(6)                        NOT NULL,
       PROJNO    CHAR(6)                        NOT NULL,
       ACTNO     SMALLINT                       NOT NULL,
       EMPTIME   DECIMAL(5,2)                           ,
       EMSTDATE  DATE                                   ,
       EMENDATE  DATE                                   ,
       FOREIGN KEY REPAPA (PROJNO, ACTNO, EMSTDATE)
                 REFERENCES DSN8810.PROJACT
                                    ON DELETE RESTRICT,
```

```
      FOREIGN KEY REPAE (EMPNO) REFERENCES DSN8810.EMP
                                     ON DELETE RESTRICT)
  IN DSN8D81A.DSN8S81P
  CCSID EBCDIC;
```

Content of the employee to project activity table:

Table 220 shows the content of the columns.

Table 220. Columns of the employee to project activity table

Column	Column Name	Description
1	EMPNO	Employee ID number
2	PROJNO	Project ID of the project
3	ACTNO	ID of the activity within the project
4	EMPTIME	A proportion of the employee's full time (between 0.00 and 1.00) to be spent on the activity
5	EMSTDATE	Date the activity starts
6	EMENDATE	Date the activity ends

Table 221 shows the indexes for the employee to project activity table:

Table 221. Indexes of the employee to project activity table

Name	On Columns	Type of Index
DSN8810.XEMPPROJACT1	PROJNO, ACTNO, EMSTDATE, EMPNO	Unique, ascending
DSN8810.XEMPPROJACT2	EMPNO	Ascending

Relationship to other tables:

The table is a dependent of:

- The employee table, through its foreign key on column EMPNO
- The project activity table, through its foreign key on columns PROJNO, ACTNO, and EMSTDATE.

Unicode sample table (DSN8810.DEMO_UNICODE)

The Unicode sample table is used to verify that data conversions to and from EBCDIC and Unicode are working as expected. The table resides in database DSN8D81A, and is defined with the following statement:

```
CREATE TABLE DSN8810.DEMO_UNICODE
           (LOWER_A_TO_Z      CHAR(26)                    ,
            UPPER_A_TO_Z      CHAR(26)                    ,
            ZERO_TO_NINE      CHAR(10)                    ,
            X00_TO_XFF    VARCHAR(256)        FOR BIT DATA)
   IN DSN8D81E.DSN8S81U
   CCSID UNICODE;
```

Content of the Unicode sample table:

Table 222 shows the content of the columns:

Table 222. Columns of the Unicode sample table

Column	Column Name	Description
1	LOWER_A_TO_Z	Array of characters, 'a' to 'z'
2	UPPER_A_TO_Z	Array of characters, 'A' to 'Z'
3	ZERO_TO_NINE	Array of characters, '0' to '9'
4	X00_TO_XFF	Array of characters, x'00' to x'FF'

This table has no indexes

Relationship to other tables:

This table has no relationship to other tables.

Relationships among the sample tables

Figure 122 shows relationships among the tables. These are established by foreign keys in dependent tables that reference primary keys in parent tables. You can find descriptions of the columns with descriptions of the tables.

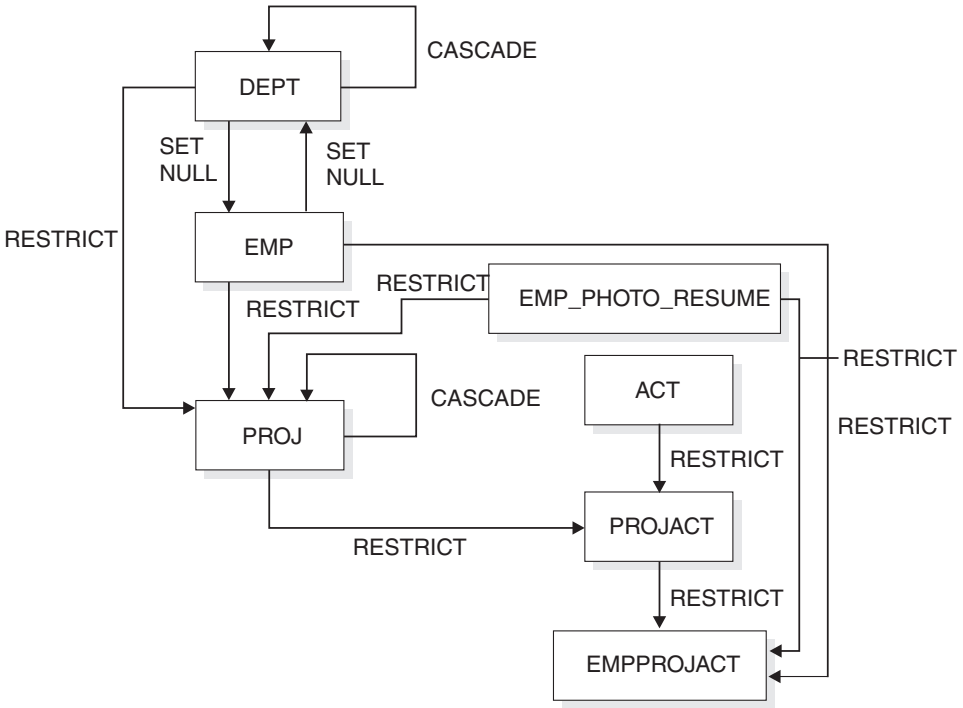


Figure 122. Relationships among tables in the sample application

Views on the sample tables

DB2 creates a number of views on the sample tables for use in the sample applications. Table 223 on page 1043 indicates the tables on which each view is defined and the sample applications that use the view. All view names have the qualifier DSN8810.

Table 223. Views on sample tables

View name	On tables or views	Used in application
VDEPT	DEPT	Organization Project
VHDEPT	DEPT	Distributed organization
VEMP	EMP	Distributed organization Organization Project
VPROJ	PROJ	Project
VACT	ACT	Project
VPROJACT	PROJACT	Project
VEMPPROJACT	EMPROJACT	Project
VDEPMG1	DEPT EMP	Organization
VEMPDPT1	DEPT EMP	Organization
VASTRDE1	DEPT	
VASTRDE2	VDEPMG1 EMP	Organization
VPROJRE1	PROJ EMP	Project
VPSTRDE1	VPROJRE1 VPROJRE2	Project
VPSTRDE2	VPROJRE1	Project
VFORPLA	VPROJRE1 EMPPROJACT	Project
VSTAFAC1	PROJACT ACT	Project
VSTAFAC2	EMPPROJACT ACT EMP	Project
VPHONE	EMP DEPT	Phone
VEMPLP	EMP	Phone

The following SQL statements are used to create the sample views:

```
CREATE VIEW DSN8810.VDEPT
   AS SELECT ALL      DEPTNO  ,
                      DEPTNAME,
                      MGRNO   ,
                      ADMRDEPT
   FROM DSN8810.DEPT;
```

Figure 123. VDEPT

```
 CREATE VIEW DSN8810.VHDEPT
    AS SELECT ALL      DEPTNO  ,
                       DEPTNAME,
                       MGRNO   ,
                       ADMRDEPT,
                       LOCATION
    FROM DSN8810.DEPT;
```

Figure 124. VHDEPT

```
CREATE VIEW DSN8810.VEMP
   AS SELECT ALL      EMPNO   ,
                      FIRSTNME,
                      MIDINIT ,
                      LASTNAME,
                      WORKDEPT
   FROM DSN8810.EMP;
```

Figure 125. VEMP

```
CREATE VIEW DSN8810.VPROJ
   AS SELECT ALL
             PROJNO, PROJNAME, DEPTNO, RESPEMP, PRSTAFF,
             PRSTDATE, PRENDATE, MAJPROJ
   FROM DSN8810.PROJ ;
```

Figure 126. VPROJ

```
CREATE VIEW DSN8810.VACT
      AS SELECT ALL   ACTNO   ,
                      ACTKWD  ,
                      ACTDESC
           FROM DSN8810.ACT ;
```

Figure 127. VACT

```
CREATE VIEW DSN8810.VPROJACT
      AS SELECT ALL
            PROJNO,ACTNO, ACSTAFF, ACSTDATE, ACENDATE
            FROM DSN8810.PROJACT ;
```

Figure 128. VPROJACT

```
CREATE VIEW DSN8810.VEMPPROJACT
      AS SELECT ALL
          EMPNO, PROJNO, ACTNO, EMPTIME, EMSTDATE, EMENDATE
          FROM DSN8810.EMPPROJACT ;
```

Figure 129. VEMPPROJACT

```
CREATE VIEW DSN8810.VDEPMG1
      (DEPTNO, DEPTNAME, MGRNO, FIRSTNME, MIDINIT,
       LASTNAME, ADMRDEPT)
    AS SELECT ALL
       DEPTNO, DEPTNAME, EMPNO, FIRSTNME, MIDINIT,
        LASTNAME, ADMRDEPT
        FROM DSN8810.DEPT LEFT OUTER JOIN DSN8810.EMP
        ON MGRNO = EMPNO ;
```

Figure 130. VDEPMG1

```
CREATE VIEW DSN8810.VEMPDPT1
      (DEPTNO, DEPTNAME, EMPNO, FRSTINIT, MIDINIT,
       LASTNAME, WORKDEPT)
    AS SELECT ALL
       DEPTNO, DEPTNAME, EMPNO, SUBSTR(FIRSTNME, 1, 1), MIDINIT,
       LASTNAME, WORKDEPT
        FROM DSN8810.DEPT RIGHT OUTER JOIN DSN8810.EMP
        ON WORKDEPT = DEPTNO ;
```

Figure 131. VEMPDPT1

```
CREATE VIEW DSN8810.VASTRDE1
    (DEPT1NO,DEPT1NAM,EMP1NO,EMP1FN,EMP1MI,EMP1LN,TYPE2,
     DEPT2NO,DEPT2NAM,EMP2NO,EMP2FN,EMP2MI,EMP2LN)
    AS SELECT ALL
        D1.DEPTNO,D1.DEPTNAME,D1.MGRNO,D1.FIRSTNME,D1.MIDINIT,
        D1.LASTNAME, '1',
        D2.DEPTNO,D2.DEPTNAME,D2.MGRNO,D2.FIRSTNME,D2.MIDINIT,
        D2.LASTNAME
        FROM DSN8810.VDEPMG1 D1, DSN8810.VDEPMG1 D2
        WHERE D1.DEPTNO = D2.ADMRDEPT ;
```

Figure 132. VASTRDE1

```
CREATE VIEW DSN8810.VASTRDE2
    (DEPT1NO,DEPT1NAM,EMP1NO,EMP1FN,EMP1MI,EMP1LN,TYPE2,
     DEPT2NO,DEPT2NAM,EMP2NO,EMP2FN,EMP2MI,EMP2LN)
    AS SELECT ALL
         D1.DEPTNO,D1.DEPTNAME,D1.MGRNO,D1.FIRSTNME,D1.MIDINIT,
         D1.LASTNAME,'2',
         D1.DEPTNO,D1.DEPTNAME,E2.EMPNO,E2.FIRSTNME,E2.MIDINIT,
         E2.LASTNAME
         FROM DSN8810.VDEPMG1 D1, DSN8810.EMP E2
         WHERE D1.DEPTNO = E2.WORKDEPT;
```

Figure 133. VASTRDE2

```
CREATE VIEW DSN8810.VPROJRE1
  (PROJNO,PROJNAME,PROJDEP,RESPEMP,FIRSTNME,MIDINIT,
   LASTNAME,MAJPROJ)
   AS SELECT ALL
      PROJNO,PROJNAME,DEPTNO,EMPNO,FIRSTNME,MIDINIT,
      LASTNAME,MAJPROJ
     FROM DSN8810.PROJ, DSN8810.EMP
     WHERE RESPEMP = EMPNO ;
```

Figure 134. VPROJRE1

```
CREATE VIEW DSN8810.VPSTRDE1
  (PROJ1NO,PROJ1NAME,RESP1NO,RESP1FN,RESP1MI,RESP1LN,
   PROJ2NO,PROJ2NAME,RESP2NO,RESP2FN,RESP2MI,RESP2LN)
   AS SELECT ALL
        P1.PROJNO,P1.PROJNAME,P1.RESPEMP,P1.FIRSTNME,P1.MIDINIT,
        P1.LASTNAME,
        P2.PROJNO,P2.PROJNAME,P2.RESPEMP,P2.FIRSTNME,P2.MIDINIT,
        P2.LASTNAME
     FROM DSN8810.VPROJRE1 P1,
       DSN8810.VPROJRE1 P2
     WHERE P1.PROJNO = P2.MAJPROJ ;
```

Figure 135. VPSTRDE1

```
CREATE VIEW DSN8810.VPSTRDE2
  (PROJ1NO,PROJ1NAME,RESP1NO,RESP1FN,RESP1MI,RESP1LN,
   PROJ2NO,PROJ2NAME,RESP2NO,RESP2FN,RESP2MI,RESP2LN)
   AS SELECT ALL
        P1.PROJNO,P1.PROJNAME,P1.RESPEMP,P1.FIRSTNME,P1.MIDINIT,
        P1.LASTNAME,
        P1.PROJNO,P1.PROJNAME,P1.RESPEMP,P1.FIRSTNME,P1.MIDINIT,
        P1.LASTNAME
     FROM DSN8810.VPROJRE1 P1
       WHERE NOT EXISTS
         (SELECT * FROM DSN8810.VPROJRE1 P2
           WHERE P1.PROJNO = P2.MAJPROJ) ;
```

Figure 136. VPSTRDE2

```
CREATE VIEW DSN8810.VFORPLA
  (PROJNO,PROJNAME,RESPEMP,PROJDEP,FRSTINIT,MIDINIT,LASTNAME)
   AS SELECT ALL
      F1.PROJNO,PROJNAME,RESPEMP,PROJDEP, SUBSTR(FIRSTNME, 1, 1),
      MIDINIT, LASTNAME
     FROM DSN8810.VPROJRE1 F1 LEFT OUTER JOIN DSN8810.EMPPROJACT F2
     ON F1.PROJNO = F2.PROJNO;
```

Figure 137. VFORPLA

```
CREATE VIEW DSN8810.VSTAFAC1
  (PROJNO, ACTNO, ACTDESC, EMPNO, FIRSTNME, MIDINIT, LASTNAME,
   EMPTIME,STDATE,ENDATE, TYPE)
   AS SELECT ALL
         PA.PROJNO, PA.ACTNO, AC.ACTDESC,' ', ' ', ' ', ' ',
         PA.ACSTAFF, PA.ACSTDATE,
         PA.ACENDATE,'1'
     FROM DSN8810.PROJACT PA, DSN8810.ACT AC
     WHERE PA.ACTNO = AC.ACTNO ;
```

Figure 138. VSTAFAC1

```
CREATE VIEW DSN8810.VSTAFAC2
  (PROJNO, ACTNO, ACTDESC, EMPNO, FIRSTNME, MIDINIT, LASTNAME,
   EMPTIME,STDATE, ENDATE, TYPE)
   AS SELECT ALL
         EP.PROJNO, EP.ACTNO, AC.ACTDESC, EP.EMPNO,EM.FIRSTNME,
         EM.MIDINIT, EM.LASTNAME, EP.EMPTIME, EP.EMSTDATE,
         EP.EMENDATE,'2'
     FROM DSN8810.EMPPROJACT EP, DSN8810.ACT AC, DSN8810.EMP EM
     WHERE EP.ACTNO = AC.ACTNO  AND EP.EMPNO = EM.EMPNO ;
```

Figure 139. VSTAFAC2

```
CREATE VIEW DSN8810.VPHONE
              (LASTNAME,
               FIRSTNAME,
               MIDDLEINITIAL,
               PHONENUMBER,
               EMPLOYEENUMBER,
               DEPTNUMBER,
               DEPTNAME)
   AS SELECT ALL      LASTNAME,
                      FIRSTNME,
                      MIDINIT ,
                      VALUE(PHONENO,'    '),
                      EMPNO,
                      DEPTNO,
                      DEPTNAME
   FROM DSN8810.EMP, DSN8810.DEPT
   WHERE WORKDEPT = DEPTNO;
```

Figure 140. VPHONE

```
CREATE VIEW DSN8810.VEMPLP
              (EMPLOYEENUMBER,
               PHONENUMBER)
   AS SELECT ALL      EMPNO   ,
                      PHONENO
   FROM DSN8810.EMP ;
```

Figure 141. VEMPLP

Storage of sample application tables

Figure 142 shows how the sample tables are related to databases and storage groups. Two databases are used to illustrate the possibility. Normally, related data is stored in the same database.

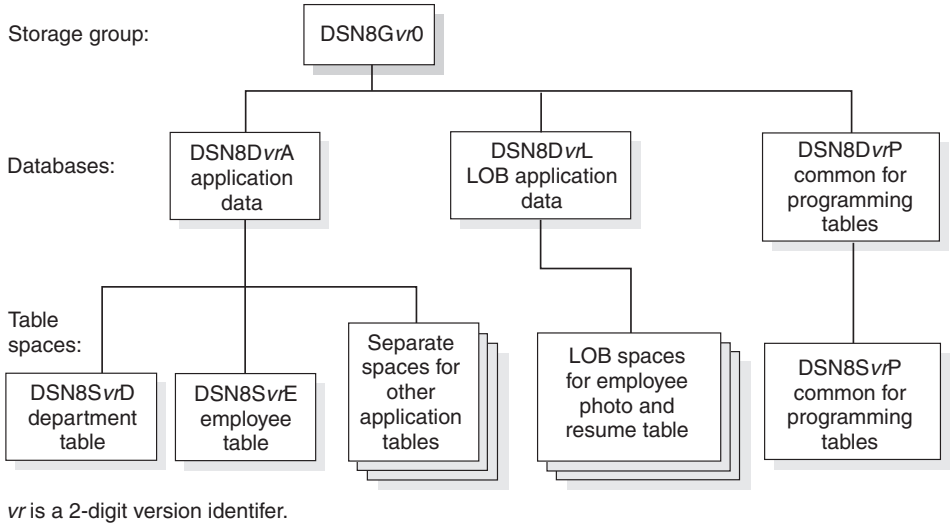


Figure 142. Relationship among sample databases and table spaces

In addition to the storage group and databases shown in Figure 142, the storage group DSN8G81U and database DSN8D81U are created when you run DSNTEJ2A.

Storage group

The default storage group, SYSDEFLT, created when DB2 is installed, is not used to store sample application data. The storage group used to store sample application data is defined by this statement:

```
CREATE STOGROUP DSN8G810
  VOLUMES (DSNV01)
  VCAT  DSNC810;
```

Databases

The default database, created when DB2 is installed, is not used to store the sample application data. DSN8D81P is the database that is used for tables that are related to programs. The remainder of the databases are used for tables that are related to applications. They are defined by the following statements:

```
CREATE DATABASE DSN8D81A
  STOGROUP DSN8G810
  BUFFERPOOL BP0
  CCSID EBCDIC;

CREATE DATABASE DSN8D81P
  STOGROUP DSN8G810
  BUFFERPOOL BP0
  CCSID EBCDIC;

CREATE DATABASE DSN8D81L
  STOGROUP DSN8G810
  BUFFERPOOL BP0
  CCSID EBCDIC;

CREATE DATABASE DSN8D81E
  STOGROUP DSN8G810
  BUFFERPOOL BP0
  CCSID UNICODE;

CREATE DATABASE DSN8D81U
  STOGROUP DSN8G81U
  CCSID EBCDIC;
```

Table spaces

The following table spaces are explicitly defined by the following statements. The table spaces not explicitly defined are created implicitly in the DSN8D81A database, using the default space attributes.

```
CREATE TABLESPACE DSN8S81D
  IN DSN8D81A
  USING STOGROUP DSN8G810
        PRIQTY 20
        SECQTY 20
        ERASE NO
  LOCKSIZE PAGE LOCKMAX SYSTEM
  BUFFERPOOL BP0
  CLOSE NO
  CCSID EBCDIC;

CREATE TABLESPACE DSN8S81E
  IN DSN8D81A
  USING STOGROUP DSN8G810
        PRIQTY 20
        SECQTY 20
        ERASE NO
  NUMPARTS 4
    (PART 1 USING STOGROUP DSN8G810
                  PRIQTY 12
```

```
                 SECQTY 12,
     PART 3 USING STOGROUP DSN8G810
                 PRIQTY 12
                 SECQTY 12)
  LOCKSIZE PAGE LOCKMAX SYSTEM
  BUFFERPOOL BP0
  CLOSE NO
  COMPRESS YES
  CCSID EBCDIC;

CREATE TABLESPACE DSN8S81B
  IN DSN8D81L
  USING STOGROUP DSN8G810
        PRIQTY 20
        SECQTY 20
        ERASE NO
  LOCKSIZE PAGE
  LOCKMAX SYSTEM
  BUFFERPOOL BP0
  CLOSE NO
  CCSID EBCDIC;

CREATE LOB TABLESPACE DSN8S81M
  IN DSN8D81L
  LOG NO;

CREATE LOB TABLESPACE DSN8S81L
  IN DSN8D81L
  LOG NO;

CREATE LOB TABLESPACE DSN8S81N
  IN DSN8D81L
  LOG NO;

CREATE TABLESPACE DSN8S81C
  IN DSN8D81P
  USING STOGROUP DSN8G810
        PRIQTY 160
        SECQTY 80
  SEGSIZE 4
  LOCKSIZE TABLE
  BUFFERPOOL BP0
  CLOSE NO
  CCSID EBCDIC;

CREATE TABLESPACE DSN8S81P
  IN DSN8D81A
  USING STOGROUP DSN8G810
        PRIQTY 160
        SECQTY 80
  SEGSIZE 4
  LOCKSIZE ROW
  BUFFERPOOL BP0
  CLOSE NO
  CCSID EBCDIC;

CREATE TABLESPACE DSN8S81R
  IN DSN8D81A
  USING STOGROUP DSN8G810
        PRIQTY 20
        SECQTY 20
        ERASE NO
  LOCKSIZE PAGE LOCKMAX SYSTEM
  BUFFERPOOL BP0
  CLOSE NO
  CCSID EBCDIC;

CREATE TABLESPACE DSN8S81S
   IN DSN8D81A
   USING STOGROUP DSN8G810
```

```
                    PRIQTY 20
                    SECQTY 20
                    ERASE NO
   LOCKSIZE PAGE LOCKMAX SYSTEM
   BUFFERPOOL BP0
   CLOSE NO
   CCSID EBCDIC;
CREATE TABLESPACE DSN8S81Q
   IN DSN8D81P
   USING STOGROUP DSN8G810
                    PRIQTY 160
                    SECQTY 80
   SEGSIZE 4
   LOCKSIZE PAGE
   BUFFERPOOL BP0
   CLOSE NO
   CCSID EBCDIC;
CREATE TABLESPACE DSN8S81U
   IN DSN8D81E
   USING STOGROUP DSN8G810
                    PRIQTY 5
                    SECQTY 5
                    ERASE NO
   LOCKSIZE PAGE LOCKMAX SYSTEM
   BUFFERPOOL BP0
   CLOSE NO
   CCSID UNICODE;
```

Appendix B. Writing exit routines

The information in this appendix is Product-sensitive Programming Interface and Associated Guidance Information, as defined in "Notices" on page 1301.

DB2 provides installation-wide exit points to routines that you provide. These exit points are described in the following sections:

- "Connection routines and sign-on routines"
- "Access control authorization exit routine" on page 1061
- "Edit routines" on page 1077
- "Validation routines" on page 1080
- "Date and time routines" on page 1083
- "Conversion procedures" on page 1086
- "Field procedures" on page 1089
- "Log capture routines" on page 1101
- "Routines for dynamic plan selection in CICS" on page 1103
- "Routine for CICS transaction invocation stored procedure" on page 1104
- "General considerations for writing exit routines" on page 1104
- "Row formats for edit and validation routines" on page 1106
- "RACF access control module" on page 1110

Connection routines and sign-on routines

Your DB2 subsystem has two exit points for authorization routines, one in connection processing and one in sign-on processing. Both exit points perform crucial steps in the assignment of values to primary IDs, secondary IDs, and SQL IDs. You must have a routine for each exit. Default routines are provided for both. DSN3@ATH is the default exit routine for connections, and DSN3@SGN is the default exit routine for sign-ons.

If your installation has a connection exit routine and you plan to use CONNECT with the USER/USING clause, you should examine your exit routine and take the following into consideration. DB2 does not update the following to reflect the user ID and password that are specified in the USER/USING clause of the CONNECT statement:

- The security-related control blocks that are normally associated with the thread
- The address space that your exit can access

For a general overview of the roles of exit routines in assigning authorization IDs, see Chapter 11, "Controlling access to a DB2 subsystem," on page 231. That chapter explains how to implement the security features that you want by assigning identifiers through RACF, or some similar program, and by using the sample connection routines and sign-on routines that are provided by IBM.

This section describes the interfaces for those routines and the functions that they provide. If you want to use secondary authorization IDs, you must replace the default routines with the sample routines, or with routines of your own.

"General considerations for writing exit routines" on page 1104 applies to these routines. One exception to the description of execution environments is that the routines execute in non-cross-memory mode.

© Copyright IBM Corp. 1982, 2008

Specifying connection and sign-on routines

Your connection routine must have a CSECT name and entry point of DSN3@ATH. The connection routine's load module name can be the same, or it can be a different name. Your sign-on routine must have a CSECT name and entry point of DSN3@SGN. The sign-on routine's load module name can be the same, or it can be a different name.

You can use an ALIAS statement of the linkage editor to provide the entry-point name.

Default routines exist in library *prefix*.SDSNLOAD. To use your routines instead, place your routines in library *prefix*.SDSNEXIT. You can use the install job DSNTIJEX to assemble and link-edit the routines and place them in the new library. If you use any other library, you might need to change the STEPLIB or JOBLIB concatenations in the DB2 start-up procedures.

You can combine both routines into one CSECT and load module if you wish, but the module must include both entry points, DSN3@ATH and DSN3@SGN. Use standard assembler and linkage editor control statements to define the entry points. DB2 loads the module twice at startup, by issuing the z/OS LOAD macro first for entry point DSN3@ATH and then for entry point DSN3@SGN. However, because the routines are reentrant, only one copy of each remains in virtual storage.

Sample connection and sign-on routines

The sample exit routines provide examples of the functions and interfaces that are described in this section. These exit routines are provided in source code as members of *prefix*.SDSNSAMP. To examine the sample connection routine, list or assemble member DSN3SATH. To examine the sample sign-on routine, list or assemble member DSN3SSGN. Because both routines use features that are not available in Assembler XF, you must use Assembler H to assemble them.

Change required for some CICS users: You must change the sample sign-on exit routine (DSN3SSGN) before assembling and using it, if the following conditions are true:

- You attach to DB2 with an AUTH parameter in the RCT other than AUTH=GROUP.
- You have the RACF list-of-groups option active.
- You have transactions whose initial primary authorization ID is not defined to RACF

To change the sample sign-on exit routine (DSN3SSGN), perform the following steps:

1. Locate the following statement in DSN3SSGN as a reference point:
```
SSGN035   DS     OH         BLANK BACKSCAN LOOP REENTRY
```
2. Locate the following statement, which comes after the reference point:
```
          B      SSGN037    ENTIRE NAME IS BLANK, LEAVE
```
3. Replace the statement with the following statement:
```
          B      SSGN090    NO GROUP NAME...  BYPASS RACF CHECK
```

By changing the statement, you avoid an abend with SQLCODE -922. The routine with the new statement provides no secondary IDs unless you use AUTH=GROUP.

When connection and sign-on routines are taken

Different local processes enter the access control procedure at different points, depending on the environment from which they originate. Different criteria apply to remote requests, which are described in "Controlling requests from remote applications" on page 238.

The following processes go through connection processing only:

- Requests that originate in TSO foreground and background (including online utilities and requests through the call attachment facility)
- JES-initiated batch jobs
- Requests through started task control address spaces (from the MVS START command)

The following processes go through connection processing, and can later go through the sign-on exit:

- The IMS control region
- The CICS recovery coordination task
- DL/I batch
- Requests through the Resource Recovery Services attachment facility (RRSAF)

The following processes go through sign-on processing:

- Requests from IMS-dependent regions (including MPP, BMP, and Fast Path)
- CICS transaction subtasks

For instructions on controlling the IDs that are associated with connection requests, see "Processing connections" on page 232. For instructions on controlling the IDs associated with sign-on requests, see "Processing sign-ons" on page 236.

EXPL for connection and sign-on routines

Figure 143 shows how the parameter list points to other information.

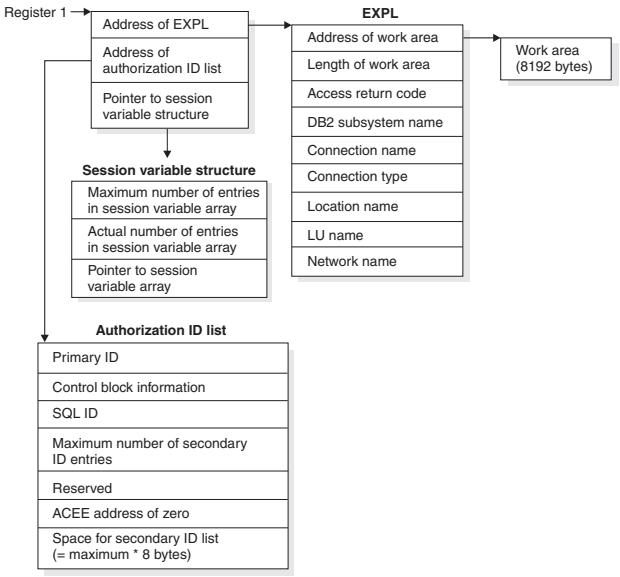


Figure 143. How a connection or sign-on parameter list points to other information

Exit parameter list for connection and sign-on routines

Connection routines and sign-on routines use 28 more bytes of the exit parameter list EXPL than other routines. Table 224 shows the entire list of connection routines and sign-on routines. The exit parameter list is described by macro DSNDEXPL.

Table 224. Exit parameter list for connection routines and sign-on routines

Name	Hex offset	Data type	Description
EXPLWA	0	Address	Address of a 8192-byte work area to be used by the routine.
EXPLWL	4	Signed 4-byte integer	Length of the work area, in bytes; value is 8192.
EXPLRSV1	8	Signed 2-byte integer	Reserved.
EXPLRC1	A	Signed 2-byte integer	Not used.
EXPLRC2	C	Signed 4-byte integer	Not used.
EXPLARC	10	Signed 4-byte integer	Access return code. Values can be: **0** Access allowed; DB2 continues processing. **12** Access denied; DB2 terminates processing with an error.
EXPLSSNM	14	Character, 8 bytes	DB2 subsystem name, left justified; for example, 'DSN '.
EXPLCONN	1C	Character, 8 bytes	Connection name for requesting location.
EXPLTYPE	24	Character, 8 bytes	Connection type for requesting location. For DDF threads, the connection type is 'DIST '.
EXPLSITE	2C	Character, 16 bytes	For SNA protocols, this is the location name of the requesting location or <*luname*>. For TCP/IP protocols, this is the dotted decimal IP address of the requester.
EXPLLUNM	3C	Character, 8 bytes	For SNA protocols, this is the locally known LU name of the requesting location. For TCP/IP protocols, this is the character string 'TCPIP '.
EXPLNTID	44	Character, 17 bytes	For SNA protocols, the fully qualified network name of the requesting location. For TCP/IP protocols, this field is reserved.

Authorization ID parameter list for connection and sign-on routines

The second parameter list, which is specific to connection routines and sign-on routines, is called an authorization ID list. Its contents are shown in Table 225 on page 1055. The description is given by macro DSNDAIDL.

Table 225. Authorization ID list for a connection or sign-on exit routine

Name	Hex offset	Data type	Description
AIDLPRIM	0	Character, 8 bytes	Primary authorization ID for input and output; see descriptions in the text.
AIDLCODE	8	Character, 2 bytes	Control block identifier.
AIDLTLEN	A	Signed 2-byte integer	Total length of control block.
AIDLEYE	C	Character, 4 bytes	Eyecatcher for block, “AIDL”.
AIDLSQL	10	Character, 8 bytes	On output, the current SQL ID.
AIDLSCNT	18	Signed 4-byte integer	Number of entries allocated to secondary authorization ID list. Always equal to 1012.
AIDLSAPM	1C	Address	For a sign-on routine only, the address of an 8-character additional authorization ID. If RACF is active, the ID is the user ID's connected group name. If the address was not provided, the field contains zero.
AIDLCKEY	20	Character, 1 byte	Storage key of the ID pointed to by AIDLSAPM. To move that ID, use the “move with key” (MVCK) instruction, specifying this key.
AIDLRSV1	21	Character, 3 bytes	Reserved
AIDLRSV2	24	Signed 4-byte integer	Reserved
AIDLACEE	28	Signed 4-byte integer	The address of the ACEE structure, if known; otherwise, zero
AIDLRACL	2C	Signed 4-byte integer	Length of data area returned by RACF, plus 4 bytes
AIDLRACR	30	26 bytes	Reserved
AIDLSEC	4A	Character, maximum x 8 bytes	List of the secondary authorization IDs, 8 bytes each

Input values for connection routines

The input values for a connection routine are as follows:

- The initial primary authorization ID for a local request can be obtained from the z/OS address space extension block (ASXB).

  The ASXB contains at most only a seven-character value. That is always sufficient for a TSO user ID or a user ID from an z/OS JOB statement, and the ASXB is always used for those cases.

  For CICS, IMS, or other started tasks, z/OS can also pass an eight-character ID. If an eight-character ID is available, and if its first seven characters agree with the ASXB value, then DB2 uses the eight-character ID. Otherwise it uses the ASXB value.

  You can alter the sample exit routine to use the ASXB value always. For instructions, see “Processing in the sample connection and sign-on routines” on page 1057.

If RACF is active, the field used contains a verified RACF user ID; otherwise, it contains blanks.

- The primary ID for a remote request is the ID passed in the conversation attach request header (SNA FMH5) or in the DRDA SECCHK command.
- The SQL ID contains blanks.
- The list of secondary IDs contains blanks.

Input values for sign-on routines

The input values for a sign-on routine are as follows:

- The initial primary ID depends on the sign-on method. See “Processing sign-ons” on page 236 for information about how the primary ID is determined.
- The SQL ID and all secondary IDs contain blanks.
- Field AIDLSAPM in the authorization ID list can contain the address of an 8-character additional authorization ID, obtained by the CICS attachment facility using the RACROUTE REQUEST=EXTRACT service with the requester's user ID. If RACF is active, this ID is the RACF-connected group name from the ACEE corresponding to the requester's user ID. Otherwise, this field contains blanks. IMS does not pass this parameter.
- Field AIDLCKEY contains the storage key of the identifier pointed to by AIDLSAPM. To move that ID, use the “move with key” (MVCK) instruction, specifying this key.
- Field AIDLACEE contains the ACEE address only for a sign-on through the CICS attachment facility and only when the CICS RCT uses AUTH=GROUP.

Expected output for connection and sign-on routines

DB2 uses the output values of the primary ID, the SQL ID, and the secondary IDs. Your routines can set these IDs to any value that is an SQL short identifier.

Important: If your identifier does not meet the 8-character criteria, the request abends. Therefore, you should add blanks to the end of short identifiers to ensure that they meet the criteria.

If the values that are returned **are not blank**, DB2 interprets them in the following ways:

- The primary ID becomes the primary authorization ID.
- The list of secondary IDs, down to the first blank entry or to a maximum of 1012 entries, becomes the list of secondary authorization IDs. The space allocated for the secondary ID list is only large enough to contain the maximum number of authorization IDs. This number is in field AIDLSCNT.

  Attention: If you allow more than 1012 secondary authorization IDs, abends and storage overlays can occur.
- The SQL ID is checked to see if it is the same as the primary or one of the secondary IDs. If it is not, the connection or sign-on process abends. Otherwise, the validated ID becomes the current SQL ID.

If the returned value of the primary ID **is blank**, DB2 takes the following steps:

- In connection processing, the default ID that is defined when DB2 is installed (UNKNOWN AUTHID on panel DSNTIPP) is substituted as the primary authorization ID and the current SQL ID. The list of secondary IDs is set to blanks.
- Sign-on processing abends. No default value exists for the primary ID.

If the returned value of the SQL ID is blank, DB2 makes it equal to the value of the primary ID. If the list of secondary IDs is blank, it remains blank. No default secondary IDs exist.

Your routine must also set a return code in word 5 of the exit parameter list to allow or deny access (field EXPLARC). By those means you can deny the connection altogether. The code must have one of the values that are shown in Table 226.

Table 226. Required return code in EXPLARC

Value	Meaning
0	Access allowed; continue processing.
12	Access denied; terminate.

Any other value will cause an abend.

Processing in the sample connection and sign-on routines

The sample routines provided by IBM can serve as models for the processing that is required in connection routines and sign-on routines.

Recommendation: Consider using the sample routines as a starting point when you write your own routines.

Both the sample connection routine (DSN3SATH) and the sample sign-on routine have similar sections for setup, constants, and storage areas. Both routines set values of the primary ID, the SQL ID, and the secondary IDs in three numbered sections.

In the sample connection routine (DSN3SATH): The three sections of the sample connection routine perform the following functions:

Section 1

Section 1 provides the same function as in the default connection routine. It determines whether the first character of the input primary ID has a value that is greater than blank (hex 40), and performs the following operations:

- If the first character is greater than hex 40, the value is not changed.
- If the first character is not greater than hex 40, the value is set according to the following rules:
  - If the request is from a TSO foreground address space, the primary ID is set to the logon ID.
  - If the request is not from a TSO foreground address space, the primary ID is set to the job user ID from the JES job control table.
  - If no primary ID is located, Section 2 is bypassed.

Section 2

At the beginning of Section 2, you can restore one commented-out instruction, which then truncates the primary authorization ID to 7 characters. (The instruction is identified by comments in the code.)

Section 2 next tests RACF options and makes the following changes in the list of secondary IDs, which is initially blank:

- If RACF is not active, the list remains blank.
- If the list of groups option is not active, but an ACEE exists, the connected group name is copied as the only secondary ID.
- If the list of groups option is active, the list of group names from the ICHPCGRP block is copied into AIDLSEC in the authorization ID list.

Section 3
Section 3 performs the following steps:

1. The SQL ID is set equal to the primary ID.
2. If the TSO data set name prefix is a valid primary or secondary ID, the SQL ID is replaced with the TSO data set name prefix. Otherwise, the SQL ID remains set to the primary ID.

In the sample sign-on routine (DSN3SSGN): The three sections of the sample sign-on routine perform the following functions:

Section 1
Section 1 does not change the primary ID.

Section 2
Section 2 sets the SQL ID to the value of the primary ID.

Section 3
Section 3 tests RACF options and makes the following changes in the list of secondary IDs, which is initially blank:

- If RACF is not active, the list remains blank.
- If the list of groups option is active, section 3 attempts to find an existing ACEE from which to copy the authorization ID list.
  - If AIDLACEE contains a valid ACEE, it is used.

    Otherwise, look for a valid ACEE chained from the TCB or from the ASXB or, if no usable ACEE exists, issue RACROUTE to have RACF build an ACEE structure for the primary ID.

    Copy the list of group names from the ACEE structure into the secondary authorization list.
  - If the exit issued RACROUTE to build an ACEE, another RACROUTE macro is issued and the structure is deleted.
- If a list of secondary authorization IDs has not been built, and AIDLSAPM is not zero, the data that is pointed to by AIDLSAPM is copied into AIDLSEC.

Performance considerations for connection and sign-on routines

Your sign-on exit routine is part of the critical path for transaction processing in IMS and CICS, so you want it to execute as quickly as possible. Avoid writing SVC calls like GETMAIN, FREEMAIN, and ATTACH. Also avoid I/O operations to any data set or database. To improve performance, you might be able to delete the list of groups that process in Section 3 of the sample sign-on exit.

The sample sign-on exit routine can issue the RACF RACROUTE macro with the default option SMC=YES. If another product issues RACROUTE with SMC=NO, a deadlock might occur.

Your routine can also enhance the performance of later authorization checking. Authorization for dynamic SQL statements is checked first for the CURRENT SQLID, then for the primary authorization ID, and then for the secondary authorization IDs. If you know that a user's privilege most often comes from a secondary authorization ID, then set the CURRENT SQLID to this secondary ID within your exit routine.

Debugging connection and sign-on routines

The diagnostic aids that are described in this section can assist you in debugging connection exit routines and sign-on exit routines.

Subsystem support identify recovery: The identify ESTAE recovery routine, DSN3IDES, generates the VRADATA entries that are shown in Table 227. The last entry, key VRAIMO, is generated only if the abend occurred within the connection exit routine.

Table 227. VRADATA entries that are generated by DSN3IDES

VRA keyname	Key hex value	Data length	Content
VRAFPI	22	8	Constant 'IDESTRAK'
VRAFP	23	24	• 32-bit recovery tracking flags • 32-bit integer AGNT block unique identifier • AGNT block address • AIDL block address • Initial primary authorization ID as copied from ASXBUSER
VRAIMO	7C	10	• Connection exit load module load point address • Connection exit entry point address • Offset of failing address in the PSW from the connection exit entry point address

Subsystem support sign-on recovery: The sign-on ESTAE recovery routine DSN3SIES generates the VRADATA entries that are shown in Table 228. The last entry, key VRAIMO, is generated only if the abend occurred within the sign-on exit routine.

Table 228. VRADATA entries that are generated by DSN3SIES

VRA keyname	Key hex value	Data length	Content
VRAFPI	22	8	Constant 'SIESTRAK'
VRAFP	23	20	• Primary authorization ID (CCBUSER) • AGNT block address • Identify-level CCB block address • Sign-on-level CCB block address
VRAIMO	7C	10	• Sign-on exit load module load point address • Sign-on exit entry point address • Offset of failing address in the PSW from the sign-on exit entry point address

Diagnostics for connection exit routines and sign-on exit routines: The connection (identify) recovery routine and the sign-on recovery routine provide diagnostics for the corresponding exit routines. The diagnostics are produced only when the abend occurs in the exit routine. The following diagnostics are available:

Dump title

The component failing module name is “DSN3@ATH” for a connection exit or “DSN3@SGN” for a sign-on exit.

z/OS and RETAIN® symptom data
SDWA symptom data fields SDWACSCT (CSECT/) and SDWAMODN (MOD/) are set to "DSN3@ATH" or "DSN3@SGN", as appropriate.

Summary dump additions
The AIDL, if addressable, and the SADL, if present, are included in the summary dump for the failing allied agent. If the failure occurred in connection or sign-on processing, the exit parameter list (EXPL) is also included. If the failure occurred in the system services address space, the entire SADL storage pool is included in the summary dump.

Session variables in connection and sign-on routines

DB2 supplies default session variables, as described in *DB2 SQL Reference*. In addition to the default session variables, the connection exit routine and the sign-on exit routine support up to 10 more session variables. You can define these additional session variables and use them to provide information to applications by using the GETVARIABLE function.

The session variable structure: The connection exit routine and the sign-on exit routine point to the session variable structure (DSNDSVS). DSNDSVS specifies the maximum number of entries in the session array, the actual number of entries in the session array, and a pointer to the session variable array. The default value for the actual number of session variables is zero.

Defining session variables: To define session variables, use the session variable array (DSNDSVA) to list up to 10 session variables as name and value pairs. The session variables that you establish in the connection exit routine and the sign-on exit routine are defined in the SESSION schema. The values that the exit routine supplies in the session variable array replace the previous values.

Example: The session variable array that is shown in Table 229 lists six session variables.

Table 229. Sample session variable array

Name	Value
default_database	DATAXM
default_driver	PZN4Y7
location	Kyoto
member_of	GROUP_42
filename	report.txt
account_number	A1-X142783

The unqualified names are defined as VARCHAR(128), and the values are defined as VARCHAR(255). The exit routines must provide these values in Unicode CCSID 1208.

Access control authorization exit routine

Is this exit routine right for you?

Using the RACF (Security Server for OS/390 or z/OS) to perform access control is not the best choice for every customer. Consider the following points before choosing RACF to perform access control:

- If you want the database administrators to manage security, integration with DB2 is very important. Using RACF access control provides less integration with DB2. In most of these cases, DB2 authorization provides advantages.
- If you want security administrators to manage security, integration with the security server is more important. In most of these cases, using RACF for access control provides advantages. Furthermore, if you want a security group to define authorization and a centralized security control point, RACF access control is an excellent match.

If you change from DB2 authorization to RACF access control, you must change to RACF methods for some authorization techniques, and you must understand how DB2 and RACF work together. Expect to make the following changes when you implement RACF access control:

- Plan to use RACF facilities (such as groups and patterns) more.
- Plan to use patterns instead of individual item access profiles and permissions.
- Plan to use RACF groups instead of secondary authorization IDs, which are not implemented in RACF. OWNER(secondaryID) generally must be a valid group.
- Find an alternative to BINDAGENT. BINDAGENT is based on secondary authorization IDs, which are not implemented in RACF. BINDAGENT provides a relatively weak security separation. Customers have found alternatives.
- Understand how SET CURRENT SQLID works with RACF. SET CURRENT SQLID can set a qualifier, but does not change authorization.
- Know that authorizations are not dropped when objects are dropped or renamed.
- Be aware of the relationship between objects and revoked privileges. Plans and packages are not invalidated when authorizations are revoked. Views are not dropped when authorizations are revoked.

DB2 provides an exit point that lets you provide your own access control authorization exit routine, or lets RACF or an equivalent security system perform DB2 authorization checking. Your routine specifies whether the authorization checking should all be done by RACF only, or by both RACF and DB2. (Also, the routine can be called and still let all checking be performed by DB2.) For more information about how to use the routine that is provided, see *DB2 RACF Access Control Module Guide*.

When DB2 invokes the routine, it passes three possible functions to the routine:

- Initialization (DB2 startup)
- Authorization check
- Termination (DB2 shutdown)

The bulk of the work in the routine is for authorization checking. When DB2 must determine the authorization for a privilege, it invokes your routine. The routine determines the authorization for the privilege and then indicates to DB2 whether the privilege is authorized or not authorized, or whether DB2 should do its own authorization check, instead.

When the exit routine is bypassed: In the following situations, the exit routine is not called to check authorization:

- The authorization ID that DB2 uses to determine access has installation SYSADM or installation SYSOPR authority (where installation SYSOPR authority is sufficient to authorize the request). This authorization check is made strictly within DB2. For example, if the execute privilege is being checked on a package, DB2 performs the check on the plan owner that this package is in. If the plan owner has installation SYSADM, the routine is not called.
- DB2 security has been disabled. (You can disable DB2 security by specifying NO on the USE PROTECTION field of installation panel DSNTIPP).
- Authorization has been cached from a prior check.
- In a prior invocation of the exit routine, the routine indicated that it should not be called again.
- GRANT statements.

The routine executes in the *ssnm*DBM1 address space of DB2.

"General considerations for writing exit routines" on page 1104 applies to this routine, but with the following exceptions to the description of execution environments:

- The routine executes in non-cross-memory mode during initialization and termination (XAPLFUNC of 1 or 3, described in Table 230 on page 1067).
- During authorization checking the routine can execute under a TCB or SRB in cross-memory or non-cross-memory mode.

Specifying the access control authorization routine

Your access control authorization routine must have a CSECT name and an entry point of DSNX@XAC. The load module name or alias name must also be DSNX@XAC. A default routine with this name and entry point exists in library *prefix*.SDSNLOAD. To use your routine instead, place it in the *prefix*.SDSNEXIT library. Use installation job DSNTIJEX to assemble and link-edit the routine and to place it in the *prefix*.SDSNEXIT library. If you use any other library, you might need to change the STEPLIB or JOBLIB concatenations in the DB2 start-up procedures.

The source code for the default routine is in *prefix*.SDSNSAMP as DSNXSXAC. You can use it to write your own exit routine. To assemble it, you must use Assembler H.

RACF provides a sample exit routine DSNXRXAC, which is shipped with DB2. It can be found in *prefix*.SDSNSAMP. For more information, see *DB2 RACF Access Control Module Guide*.

The default access control authorization routine

The default exit routine returns a code to the DB2 authorization module. The code indicates that a user-defined access control authorization exit routine is not available. DB2 then performs normal authorization checking and does not attempt to invoke this exit routine again.

When the access control authorization routine is taken

This exit routine is taken in the following three instances:

At DB2 startup
This exit routine is taken when DB2 starts to allow the external authorization checking application to perform any required setup prior to authorization checking. For example, loading authorization profiles into storage is a required setup task. DB2 uses the reason code that the exit routine sets during startup to determine how to handle exception situations. See "Exception processing" on page 1075 for details.

When an authorization check is to be performed on a privilege
This exit routine is taken when DB2 accesses security tables in the catalog to check authorization on a privilege. The exit routine is taken only if none of the prior invocations have indicated that the exit routine must not be called again.

At DB2 shutdown
This exit routine is taken when DB2 is stopping, to let the external authorization checking application perform its cleanup before DB2 stops.

Considerations for the access control authorization routine

This section discusses special considerations for using the access control authorization exit routine.

When DB2 cannot provide an ACEE

Sometimes DB2 cannot provide an ACEE.

| **Example:** If you are not using external security in CICS (that is, SEC=NO is
| specified in the DFHSIT), CICS does not pass an ACEE to the CICS attachment
| facility. The ACEE address is passed for CICS transactions, if available.

When DB2 does not have an ACEE, it passes zeros in the XAPLACEE field. If this happens, your routine can return a 4 in the EXPLRC1 field, and let DB2 handle the authorization check.

DB2 does not pass the ACEE address for IMS transactions. The ACEE address is
passed for CICS transactions, if available.

DB2 does pass the ACEE address when it is available for DB2 commands that are
issued from a logged on z/OS console. DB2 does not pass the ACEE address for
DB2 commands that are issued from a console that is not logged on, or for the
START DB2 command, or commands issued automatically during DB2 startup.

Authorization IDs and ACEEs

XAPL has two authorization ID fields, XAPLUPRM (the primary authorization ID) and XAPLUCHK (the authorization ID that DB2 uses to perform the authorization). These two fields might have different values.

The ACEE passed in XAPLACEE is that of the primary authorization ID, XAPLUPRM.

The implications of the XAPLUPRM and XAPLUCHK relationship need to be clearly understood. XAPLUCHK, the authorization ID that DB2 uses to perform authorization may be the primary authorization ID (XAPLUPRM), a secondary authorization ID, or another authorization ID such as a package owner.

If the RACF access control module is used, the following rules apply:

- RACF uses the ACEE of the primary authorization ID (XAPLUPRM) to perform authorization.
- Secondary authorization IDs are not implemented in RACF. RACF groups should be used instead.

Examples: The following examples show how the rules apply:

- A plan or package may be bound successfully by using the privileges of the binder (XAPLUPRM). Then only the EXECUTE privilege on the plan or package is needed to execute it. If at some point this plan or package is marked invalid (for instance, if a table it depends upon is dropped and recreated), the next execution of it will cause an AUTOBIND, which will usually fail. It fails because the AUTOBIND checks the privilege on the runner. The plan or package must be rebound by using an authorization ID that has the necessary privileges.
- If the OWNER on the BIND command is based on secondary authorization IDs, which are not supported by RACF. RACF groups should be used instead.
- SET CURRENT SQLID can set a qualifier, but it cannot change authorization.
- DYNAMIC RULES settings have a limited effect on which authorization ID is checked. Only the primary authorization ID and secondary IDs that are valid RACF groups for this user are considered.
- User-defined function and stored procedure authorizations involve several authorization IDs, such as implementer, definer, invoker, and so forth. Only the primary authorization ID and secondary IDs that are RACF groups are considered.

Stored procedure authorization

At run time, DB2 checks the EXECUTE privilege on stored procedure packages.

The only authorization ID that is available to check at run time is the primary authorization ID that runs the stored procedure package. If your security plan requires that you do not grant the EXECUTE privilege to all authorization IDs, the ID that runs the stored procedure package might not have it.

However, you can ensure that the ID can run the package without granting the EXECUTE privilege on the stored procedure package to it. To do this, grant the privilege to the ID that binds the plan that calls the stored procedure package, and satisfy the following conditions:

- The stored procedure definition must not specify COLLID.
- The program that calls the stored procedure must be bound directly to the plan that calls it by using the MEMBER bind option and by specifying the name of the calling program. The stored procedure package must be referenced in the PKLIST as *collection-id*.* or *collection-id.stored-procedure-package-id* with the VALIDATE(BIND) option.
- The SET CURRENT PACKAGESET statement must not be used before the stored procedure is called in the program.

Invalid and inoperative plans and packages

In DB2, when a privilege that is required by a plan or package is revoked, the plan or package is invalidated. If you use an authorization access control routine, it cannot tell DB2 that a privilege is revoked. Therefore, DB2 cannot know to invalidate the plan or package.

If the revoked privilege is the EXECUTE privilege on a user-defined function, DB2 marks the plan or package inoperative instead of invalid.

If a privilege that the plan or package depends on is revoked, and if you want to invalidate the plan or package or make it inoperative, you must use the SQL GRANT statement to grant the revoked privilege and then use the SQL REVOKE statement to revoke it.

Dropping views

When a privilege that is required to create a view is revoked, the view is dropped. Similar to the revocation of plan privileges, such an event is not communicated to DB2 by the authorization checking routine.

If you want DB2 to drop the view when a privilege is revoked, you must use the SQL statements GRANT and REVOKE.

Caching of EXECUTE on plans, packages, and routines

The results of authorization checks on the EXECUTE privilege for plans are not cached when those checks are performed by the exit routine.

The results of authorization checks on the EXECUTE privilege for packages and routines are cached (assuming that package and routine authorization caching is enabled on your system). If this privilege is revoked in the exit routine, the cached information is not updated to reflect the revoke. You must use the GRANT statement and the REVOKE statement to update the cached information.

Caching of dynamic SQL statements

Dynamic statements can be cached when they have passed the authorization checks (assuming that dynamic statement caching is enabled on your system). If the privileges that this statement requires are revoked from the authorization ID that is cached with the statement, this cached statement must be invalidated. If the privilege is revoked in the exit routine this does not happen, and you must use the SQL statements GRANT and REVOKE to refresh the cache.

Resolution of user-defined functions

The create timestamp for the user-defined function must be older than the bind timestamp for the package or plan in which the user-defined function is invoked. If DB2 authorization checking is in effect, and DB2 performs an automatic rebind on a plan or package that invokes a user-defined function, any user-defined functions that were created after the original BIND or REBIND of the invoking plan or package are not candidates for execution.

If you use an access control authorization exit routine, some user-defined functions that were not candidates for execution before the original BIND or REBIND of the invoking plan or package might become candidates for execution during the automatic rebind of the invoking plan or package. If a user-defined function is invoked during an automatic rebind, and that user-defined function is invoked from a trigger body and receives a transition table, the form of the invoked function that DB2 uses for function selection includes only the columns of the

transition table that existed at the time of the original BIND or REBIND of the package or plan for the invoking program.

Creating materialized query tables

When a materialized query table is created, a CRTVUAUTT authorization check is performed. The CRTVUAUTT check is used to determine whether the creator of a materialized query table can provide the required SELECT privileges on base tables to the owner of the materialized query table. If the owner of the materialized query table has the required privileges, the CRTVUAUTT authorization check proves redundant. However, the check is performed before the owner of the materialized query table's privileges are determined. Therefore, if the materialized query table owner holds the necessary privileges and the creator of the materialized query table does not, the CRTVUAUTT check can produce unwanted error messages.

For an ACA exit routine to suppress unwanted error messages during the creation of materialized query tables, XAPLFSUP is turned on.

Parameter list for the access control authorization routine

Figure 144 shows how the parameter list points to other information.

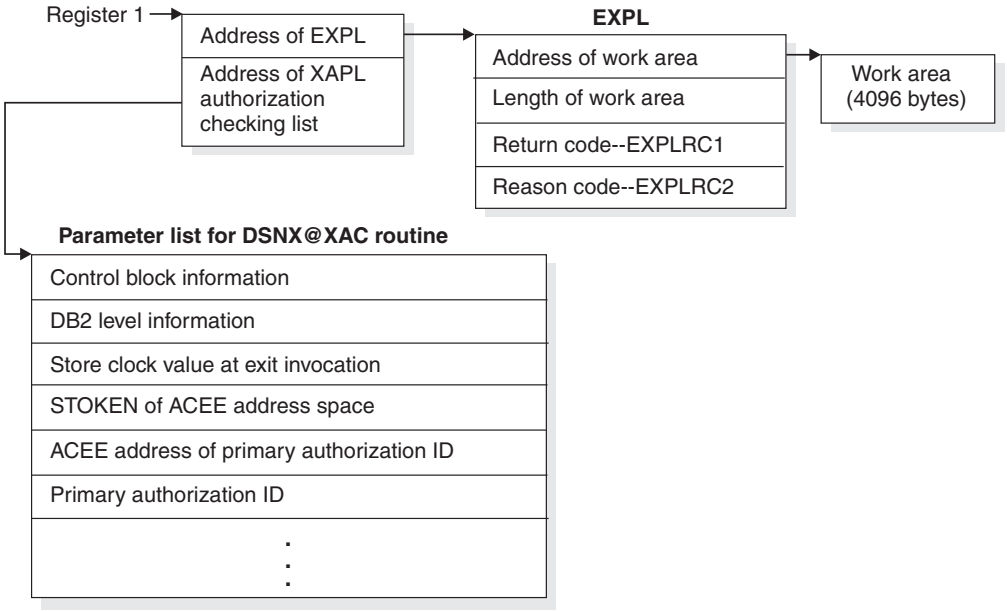


Figure 144. How an authorization routine's parameter list points to other information

The work area (4096 bytes) is obtained once during the startup of DB2 and only released when DB2 is shut down. The work area is shared by all invocations to the exit routine.

At invocation, registers are set as described in "Registers at invocation for exit routines" on page 1105, and the authorization checking routine uses the standard exit parameter list (EXPL) described there. Table 230 on page 1067 shows the exit-specific parameter list, described by macro DSNDXAPL.

Table 230. Parameter list for the access control authorization routine. Field names indicated by an asterisk (*) apply to initialization, termination, and authorization checking. Other fields apply to authorization checking only.

Name	Hex offset	Data type	Input or output	Description
XAPLCBID*	0	Character, 2-bytes	Input	Control block identifier; value X'216A'.
XAPLLEN *	2	Signed, 2-byte integer	Input	Length of XAPL; value X'100' (decimal 256).
XAPLEYE *	4	Character, 4 bytes	Input	Control block eye catcher; value “XAPL”.
XAPLLVL *	8	Character, 8 bytes	Input	DB2 version and level; for example, “VxRxMx”.
XAPLSTCK *	10	Character, 8 bytes	Input	The store clock value when the exit is invoked. Use this to correlate information to this specific invocation.
XAPLSTKN *	18	Character, 8 bytes	Input	STOKEN of the address space in which XAPLACEE resides. Binary zeroes indicate that XAPLACEE is in the home address space.
XAPLACEE *	20	Address	Input	ACEE address: • Of the DB2 address space (*ssnm*DBM1) when XAPLFUNC is 1 or 3. • Of the primary authorization ID associated with this agent when XAPLFUNC is 2. There may be cases were an ACEE address is not available for an agent. In such cases this field contains binary zeroes.
XAPLUPRM *	24	Character, 8 bytes	Input	One of the following IDs: • When XAPLFUNC is 1 or 3, it contains the User ID of the DB2 address space (*ssnm*DBM1) • When XAPLFUNC is 2, it contains the primary authorization ID associated with the agent
XAPLFUNC *	2C	Signed, 2-byte integer	Input	Function to be performed by exit routine 1 Initialization 2 Authorization Check 3 Termination
XAPLGPAT *	2E	Character, 4 bytes	Input	DB2 group attachment name for data sharing. The DB2 subsystem name if not data sharing.
XAPLRSV1	32	Character, 6bytes		Reserved
XAPLPRIV	38	Signed, 2-byte integer	Input	DB2 privilege being checked. See macro DSNXAPRV for a complete listing of privileges.

Table 230. Parameter list for the access control authorization routine (continued). Field names indicated by an asterisk (*) apply to initialization, termination, and authorization checking. Other fields apply to authorization checking only.

Name	Hex offset	Data type	Input or output	Description
XAPLTYPE	3A	Character, 1	Input	DB2 object type: **D** Database **R** Table space **T** Table **P** Application plan **K** Package **S** Storage group **C** Collection **B** Buffer pool **U** System privilege **E** Distinct type **F** User-defined function **M** Schema **O** Stored procedure **J** JAR **V** View **Q** Sequence
XAPLFLG1	3B	Character, 1	Input	The highest-order bit, bit 8, (XAPLCHKS) is on if the secondary IDs associated with this authorization ID (XAPLUCHK) are included in DB2's authorization check. If it is off, only this authorization ID is checked. Bit 7 (XAPLUTB) is on if this is a table or view privilege (SELECT, INSERT, and so on) and if SYSCTRL is not sufficient authority to perform the specified operation on a table or view. SYSCTRL does not have the privilege of accessing user data unless the privilege is specifically granted to it. Bit 6 (XAPLAUTO) is on if this is an AUTOBIND. See “Access control authorization exit routine” on page 1061 for more information on function resolution during an AUTOBIND. Bit 5 (XAPLCRVW) is on if the installation parameter DBADM CREATE AUTH is set to YES. Bit 4 (XAPLRDWR) is on if the privilege is a write privilege. If the privilege is a read-only privilege, bit 4 is off. Bit 3 (XAPLFSUP) is on to suppress error messages from the CRTVUAUTT authorization check during the creation of a materialized query table. These error messages are caused by intermediate checks that do not affect the final result. The remaining 2 bits are reserved.
XAPLUCHK	3C	Address, 4 bytes	Input	Address to the authorization ID on which DB2 performs the check. It could be the primary, secondary, or some other ID. This is a VARCHAR(128) field.

Table 230. Parameter list for the access control authorization routine (continued). Field names indicated by an asterisk (*) apply to initialization, termination, and authorization checking. Other fields apply to authorization checking only.

Name	Hex offset	Data type	Input or output	Description
XAPLOBJN	40	Address, 4 bytes	Input	Address to the unqualified name of the object with which the privilege is associated. This is a VARCHAR(128) field.It is one of the following names: **Name** / **Length** Database / 8 Table space / 8 Table / VARCHAR(128) Application plan / 8 Package / VARCHAR(128) Storage group / VARCHAR(128) Collection / VARCHAR(128) Buffer pool / 8 Schema / VARCHAR(128) Distinct type / VARCHAR(128) User-defined function / VARCHAR(128) JAR / VARCHAR(128) View / VARCHAR(128) Sequence / VARCHAR(128) For special system privileges (SYSADM, SYSCTRL, and so on) this field might contain binary zeroes. See macro DSNXAPRV.
XAPLOWNQ	44	Address, 4 bytes	Input	Address of the object owner (creator) or object qualifier. The contents of this parameter depends on either the privilege being checked or the object. See Table 232 on page 1071. This is a VARCHAR(128) field. If this field is not applicable, it contains binary zeros.
XAPLREL1	48	Address, 4 bytes	Input	Address of other related information 1. The contents of this parameter depend on either the privilege being checked or the object. See Table 232 on page 1071. This is a VARCHAR(128) field. If this field is not applicable, it contains binary zeros.
XAPLREL2	4C	Address, 4 bytes	Input	Address of other related information 2. The contents of this parameter depends on the privilege being checked. See Table 232 on page 1071. This is a VARCHAR(128) field. If this field is not applicable, it contains binary zeros.
XAPLDBSP	50	Address, 4 bytes	Input	Address of database information. This information is passed for CREATE VIEW and CREATE ALIAS. It points to database information as described in Table 231 on page 1070. If this field is not applicable, it contains binary zeros.
XAPLRSV2	54	Character, 79 bytes		Reserved.

Table 230. Parameter list for the access control authorization routine (continued). Field names indicated by an asterisk (*) apply to initialization, termination, and authorization checking. Other fields apply to authorization checking only.

Name	Hex offset	Data type	Input or output	Description
XAPLFROM	A3	Character, 1 byte	Input	Source of the request: **S** Remote request that uses DB2 private protocol. ' ' Not a remote request that uses DB2 private protocol. DB2 authorization restricts remote requests that use DB2 private protocol to the SELECT, UPDATE, INSERT and DELETE privileges.
XAPLXBTS	A4	Timestamp, 10 bytes	Input	The function resolution timestamp. Authorizations received prior to this timestamp are valid. Applicable to functions and procedures. See *DB2 SQL Reference* for more information on function resolution.
XAPLONWT	AE	Character, 1 byte	Output	Information required by DB2 from the exit routine for the UPDATE and REFERENCES table privileges: **Value** **Explanation** ' ' Requester has privilege on the entire table * Requester has privilege on just this column See macro DSNXAPRV for definition of these privileges.
XAPLRSV3	AF	Character, 1 byte		Reserved.
XAPLDIAG	B0	Character, 80 bytes	Output	Information returned by the exit routine to help diagnose problems.

Table 231 has database information for determining authorization for creating a view. The address to this parameter list is in XAPLREL2. See Table 232 on page 1071 for more information on CREATE VIEW.

Table 231. Parameter list for the access control authorization routine—database information

Name	Hex offset	Data type	Input or output	Description
XAPLDBNP	0	Address	Input	Address of information for the next database. X'00000000' indicates no next database exists.
XAPLDBNM	4	Character, 8 bytes	Input	Database name.

Table 231. Parameter list for the access control authorization routine—database information (continued)

Name	Hex offset	Data type	Input or output	Description
XAPLDBDA	C	Character, 1 byte	Output	Required by DB2 from the exit routine for CREATE VIEW. A value of **Y** and EXPLRC1=0 indicate that the user ID in field XAPLUCHK has database administrator authority on the database in field XAPLDBNM. When the exit checks if XAPLUCHK can create a view for another authorization ID, it first checks for SYSADM or SYSCTRL authority. If the check is successful, no more checking is necessary because SYSCTRL authority (for non-user tables) or SYSADM authority satisfies the requirement that the view owner has the SELECT privilege for all tables and views that the view might be based on. This is indicated by a blank value and EXPLRC1=0. If the authorization ID does not have SYSADM or SYSCTRL authority, the exit checks if the view creator has DBADM on each database of the tables that the view is based on because the DBADM authority on the database of the base table satisfies the requirement that the view owner has the SELECT privilege for all base tables in that database.
XAPLRSV5	D	Character, 3 bytes	none	Reserved.

XAPLOWNQ, XAPLREL1 and XAPLREL2 might further qualify the object or may provide additional information that can be used in determining authorization for certain privileges. These privileges and the contents of XAPLOWNQ, XAPLREL1 and XAPLREL2 are shown in Table 232.

Table 232. Related information for certain privileges

Privilege	Object type (XAPLTYPE)	XAPLOWNQ	XAPLREL1	XAPLREL2
0263 (USAGE)	E	Address of schema name	Address of distinct type owner	Contains binary zeroes
0064 (EXECUTE) 0265 (START) 0266 (STOP) 0267 (DISPLAY)	F	Address of schema name	Address of user-defined function owner	Contains binary zeroes
0263 (USAGE)	J	Address of schema name	Address of JAR owner	Contains binary zeroes
0064 (EXECUTE)	K	Address of collection ID	Contains binary zeroes	Contains binary zeroes
0065 (BIND)	K	Address of collection ID	Address of package owner	Contains binary zeroes
0073 (DROP)	K	Address of collection ID	Contains binary zeroes	Address of version ID
0097 (COMMENT)	K	Address of collection ID	Address of package owner	Contains binary zeroes
0225 (COPY ON PKG)	K	Address of collection ID	Address of package owner	Contains binary zeroes

Table 232. Related information for certain privileges (continued)

Privilege	Object type (XAPLTYPE)	XAPLOWNQ	XAPLREL1	XAPLREL2
0228 (ALLPKAUT)	K	Address of collection ID	Contains binary zeroes	Contains binary zeroes
0229 (SUBPKAUT)	K	Address of collection ID	Contains binary zeroes	Contains binary zeroes
0252 (ALTERIN) 0097 (COMMENT) 0252 (DROPIN)	M	Address of schema name	Address of object owner	Contains binary zeroes
0064 (EXECUTE) 0265 (START) 0266 (STOP) 0267 (DISPLAY)	O	Address of schema name	Address of procedure owner	Contains binary zeroes
0065 (BIND)	P	Address of plan owner	Contains binary zeroes	Contains binary zeroes
0097 (COMMENT)	P	Address of plan owner	Contains binary zeroes	Contains binary zeroes
0061 (ALTER) 0263 (USAGE)	Q	Address of schema name	Address of sequence name	Contains binary zeroes
0061 (ALTER)	R	Address of database name	Contains binary zeroes	Contains binary zeroes
0073 (DROP)	R	Address of database name	Contains binary zeroes	Contains binary zeroes
0087 (USE)	R	Address of database name	Contains binary zeroes	Contains binary zeroes
0053 (UPDATE) 0054 (REFERENCES)	T	Address of table name qualifier	Address of column name, if applicable	Address of database name
0022 (CATMAINT CONVERT) 0050 (SELECT) 0051 (INSERT) 0052 (DELETE) 0055 (TRIGGER) 0056 (CREATE INDEX) 0061 (ALTER) 0073 (DROP) 0075 (LOAD) 0076 (CHANGE NAME QUALIFIER) 0097 (COMMENT) 0098 (LOCK) 0233 (ANY TABLE PRIVILEGE) 0251 (RENAME) 0275 (REFRESH)	T	Address of table name qualifier	Contains binary zeroes	Address of database name
0020 (DROP ALIAS) 0104 (DROP SYNONYM)	T	Address of object name qualifier	Contains binary zeroes	Contains binary zeroes
0103 (ALTER INDEX) 0105 (DROP INDEX) 0274 (COMMENT ON INDEX)	T	Address of object name qualifier	Contains binary zeroes	Address of database name

Table 232. Related information for certain privileges (continued)

Privilege	Object type (XAPLTYPE)	XAPLOWNQ	XAPLREL1	XAPLREL2
0227 (BIND AGENT)	U	Address of package owner	Contains binary zeroes	Contains binary zeroes
0015 (CREATE ALIAS)	U	Contains binary zeroes	Contains binary zeroes	Address of database name, if the alias is on a table
0053 (UPDATE)	V	Address of view name qualifier	Address of column name, if applicable	Contains binary zeroes
0050 (SELECT) 0051 (INSERT) 0052 (DELETE) 0073 (DROP) 0097 (COMMENT) 0233 (ANY TABLE PRIVILEGE)	V	Address of view name qualifier	Contains binary zeroes	Contains binary zeroes
0061 (ALTER)	V	Address of view name qualifier	Contains binary zeroes	Contains binary zeroes

The data types and field lengths of the information shown in Table 232 on page 1071 is shown in Table 233.

Table 233. Data types and field lengths

Resource name or other	Type	Length
Database name	Character	8
Table name qualifier	Character	VARCHAR(128)
Object name qualifier	Character	VARCHAR(128)
Column name	Character	VARCHAR(128)
Collection ID	Character	VARCHAR(128)
Plan owner	Character	VARCHAR(128)
Package owner	Character	VARCHAR(128)
Package version ID	Character	VARCHAR(64)
Schema name	Character	VARCHAR(128)
Distinct type owner	Character	VARCHAR(128)
JAR owner	Character	VARCHAR(128)
User-defined function owner	Character	VARCHAR(128)
Procedure owner	Character	VARCHAR(128)
View name qualifier	Character	VARCHAR(128)
Sequence owner	Character	VARCHAR(128)
Sequence name	Character	VARCHAR(128)

Expected output for the access control authorization routine

Your authorization exit routine is expected to return certain fields when it is called. These output fields are indicated in Table 230 on page 1067. If an unexpected value is returned in any of these fields an abend occurs. Register 3 points to the field in

error, and abend code 00E70009 is issued.

Field	Required or optional
EXPLRC1	Required
EXPLRC2	Optional
XAPLONWT	Required only for UPDATE and REFERENCES table privileges
XAPLDIAG	Optional

Handling return codes

Place return codes from the exit routine in the EXPL field named EXPLRC1.

Return codes during initialization: EXPLRC1 must have one of the values that are shown in Table 234 during initialization.

Table 234. Required values in EXPLRC1 during initialization

Value	Meaning
0	Initialization successful.
12	Unable to service request; don't call exit again.

See "Exception processing" on page 1075 for an explanation of how the EXPLRC1 value affects DB2 processing.

Return codes during termination: DB2 does not check EXPLRC1 on return from the exit routine.

Return codes during authorization check: Make sure that EXPLRC1 has one of the values that are shown in Table 235 during the authorization check.

Table 235. Required values in EXPLRC1 during authorization check

Value	Meaning
0	Access permitted.
4	Unable to determine; perform DB2 authorization checking.
8	Access denied.
12	Unable to service request; don't call exit again.

See "Exception processing" on page 1075 for an explanation of how the EXPLRC1 value affects DB2 processing. On authorization failures, the return code is included in the IFCID 0140 trace record.

Handling reason codes

Reason codes during initialization: The reason code (EXPLRC2) that the exit routine returns after initialization determines how DB2 processes the return code (EXPLRC1) that the exit returns during initialization and authorization checking. The reason codes are shown in Table 236 on page 1075. See "Exception processing" on page 1075 for details.

Table 236. Reason codes during initialization

Value	Meaning
-1	Identifies the default exit routine shipped with DB2. If you replace or modify the default exit, you should not use this value.
16	Indicates to DB2 that it should terminate if the exit routine returns EXPLRC1=12, an invalid EXPLRC1 or abnormally terminates during initialization or authorization checking. **When the exit routine sets the reason code to 16, DB2 does an immediate shutdown, without waiting for tasks to end.** For long-running tasks, an immediate shutdown can mean that recovery times are long.
Other	Ignored by DB2.

Reason codes during authorization check: Field EXPLRC2 lets you put in any code that would be of use in determining why the authorization check in the exit routine failed. On authorization failures, the reason code is included in the IFCID 0140 trace record.

Exception processing

During initialization or authorization checking, DB2 issues diagnostic message DSNX210I to the operator's console, if one of the following conditions occur:

- The authorization exit returns a return code of 12 or an invalid return code.
- The authorization exit abnormally terminates.

Additional actions that DB2 performs depend on the reason code that the exit returns during initialization. Table 237 on page 1076 summarizes these actions.

Table 237. How an error condition affects DB2 actions during initialization and authorization checking

Exit result	Reason code of 16 returned by exit routine during initialization	Reason code other than 16 or -1 returned by exit routine during initialization[1]
Return code 12	• The task[2] abnormally terminates with reason code 00E70015 • DB2 terminates	• The task[2] abnormally terminates with reason code 00E70009 • DB2 switches to DB2 authorization checking
Invalid return code	• The task[2] abnormally terminates with reason code 00E70015 • DB2 terminates	• The task[2] abnormally terminates with reason code 00E70009 • DB2 switches to DB2 authorization checking
Abnormal termination during initialization	DB2 terminates	DB2 switches to DB2 authorization checking
Abnormal termination during authorization checking	You can use the subsystem parameter AEXITLIM[3] to control how DB2 and the exit behave. **Example:** If you set AEXITLIM to 10, the exit routine continues to run after the first 10 abnormal terminations. On the eleventh abnormal termination, the exit stops and DB2 terminates.	You can use the subsystem parameter AEXITLIM to control how DB2 and the exit behave. **Example:** If you set AEXITLIM to 10, the exit routine continues to run after the first 10 abnormal terminations. On the eleventh abnormal termination, the exit routine stops and DB2 switches to DB2 authorization checking.

Notes:

1. During initialization, DB2 sets a value of -1 to identify the default exit. The user exit routine should not set the reason code to -1.
2. During initialization, the task is DB2 startup. During authorization checking, the task is the application.
3. AEXITLI (authorization exit limit) can be updated online. Refer to SET SYSPARM in *DB2 Command Reference*.

Debugging the access control authorization routine

You can use IFCID 0314 to provide a trace record of the parameter list on return from the exit routine. You can activate this trace by turning on performance trace class 22.

Determining whether the access control authorization routine is active

To determine whether the exit routine or DB2 is performing authorization checks, perform the following steps:

1. Start audit trace class 1.
2. Choose a DB2 table on which to issue a SELECT statement and an authorization ID to perform the SELECT. The authorization ID must not have the DB2 SELECT privilege or the external security system SELECT privilege on the table.

3. Use the authorization ID to issue a SELECT statement on the table. The SELECT statement should fail.
4. Format the trace data and examine the return code (QW0140RC) in the IFCID 0140 trace record.
   - QW0140RC = –1 indicates that DB2 performed the authorization check and denied access.
   - QW0140RC = 8 indicates that the external security system performed the authorization check and denied access.

Edit routines

Edit routines are assigned to a table by the EDITPROC clause of CREATE TABLE. An edit routine receives the entire row of the base table in internal DB2 format; it can transform that row when it is stored by an INSERT or UPDATE SQL statement, or by the LOAD utility. It also receives the transformed row during retrieval operations and must change it back to its original form. Typical uses are to compress the storage representation of rows to save space on DASD and to encrypt the data.

You cannot use an edit routine on a table that contains a LOB or a ROWID column.

You cannot use EDITPROC on a table with a column name that is longer than 18
EBCDIC bytes. If you do, you will receive an error message.

The transformation your edit routine performs on a row (possibly encryption or compression) is called *edit-encoding*. The same routine is used to undo the transformation when rows are retrieved; that operation is called *edit-decoding*.

Important: The edit-decoding function must be the exact inverse of the edit-encoding function. For example, if a routine encodes 'ALABAMA' to '01', it must decode '01' to 'ALABAMA'. A violation of this rule can lead to an abend of the DB2 connecting thread, or other undesirable effects.

Your edit routine can encode the entire row of the table, including any index keys. However, index keys are extracted from the row before the encoding is done, therefore, index keys are stored in the index in *edit-decoded* form. Hence, for a table with an edit routine, index keys in the table **are** edit-coded; index keys in the index are **not** edit-coded.

The sample application contains a sample edit routine, DSN8EAE1. To print it, use ISPF facilities, IEBPTPCH, or a program of your own. Or, assemble it and use the assembly listing.

There is also a sample routine that does Huffman data compression, DSN8HUFF in library *prefix*.SDSNSAMP. That routine not only exemplifies the use of the exit parameters, it also has potentially some use for data compression. If you intend to use the routine in any production application, please pay particular attention to the warnings and restrictions given as comments in the code. You might prefer to let DB2 compress your data. For instructions, see "Compressing your data" on page 702.

"General considerations for writing exit routines" on page 1104 applies to edit routines.

Specifying the edit routine routine

To name an edit routine for a table, use the EDITPROC clause of the CREATE TABLE statement, followed by the name of the routine. If you plan to use an edit routine, specify it when you create the table. In operation, the routine is loaded on demand.

You cannot add an edit routine to a table that already exists: you must drop the table and re-create it. Also, you cannot alter a table with an edit routine to add a column. Again, you must drop the table and re-create it, and presumably also alter the edit routine in some way to account for the new column.

When the edit routine is taken

An edit routine is invoked to edit-code a row whenever DB2 inserts or updates one, including inserts made by the LOAD utility. It is invoked *after* any date routine, time routine, or field procedure. If there is also a validation routine, the edit routine is invoked **after** the validation routine. Any changes made to the row by the edit routine do not change entries made in an index.

The same edit routine is invoked to edit-decode a row whenever DB2 retrieves one. On retrieval, it is invoked *before* any date routine, time routine, or field procedure. If retrieved rows are sorted, the edit routine is invoked *before* the sort. An edit routine is not invoked for a DELETE operation without a WHERE clause that deletes an entire table in a segmented table space.

Parameter lists for the edit routine

At invocation, registers are set as described in "Registers at invocation for exit routines" on page 1105, and the edit routine uses the standard exit parameter list (EXPL) described there. Table 238 shows the exit-specific parameter list, described by macro DSNDEDIT. Figure 145 on page 1079 shows how the parameter list points to other row information.

Table 238. Parameter list for an edit routine

Name	Hex offset	Data type	Description
EDITCODE	0	Signed 4-byte integer	Edit code telling the type of function to be performed, as follows: 0 Edit-encode row for insert or update 4 Edit-decode row for retrieval
EDITROW	4	Address	Address of a row description. Its format is shown in Table 280 on page 1109.
	8	Signed 4-byte integer	Reserved
EDITILTH	C	Signed 4-byte integer	Length of the input row
EDITIPTR	10	Address	Address of the input row
EDITOLTH	14	Signed 4-byte integer	Length of output row. On entry, this is the size of the area in which to place the output row. The exit must not modify storage beyond this length.
EDITOPTR	18	Address	Address of the output row

Processing requirements for edit routines

Your routine must be based on the DB2 data formats; see “Row formats for edit and validation routines” on page 1106.

Incomplete rows and edit routines

Sometimes DB2 passes, to an edit routine, an input row that has fewer fields than there are columns in the table. In that case, the routine must stop processing the row after the last input field. Columns for which no input field is provided are always at the end of the row and are never defined as NOT NULL; either they allow nulls, they are defined as NOT NULL WITH DEFAULT, or the column is a ROWID column.

Use macro DSNDEDIT to get the starting address and row length for edit exits. Add the row length to the starting address to get the first invalid address beyond the end of the input buffer; your routine must *not* process any address as large as that.

Figure 145 shows how the parameter list points to other row information.

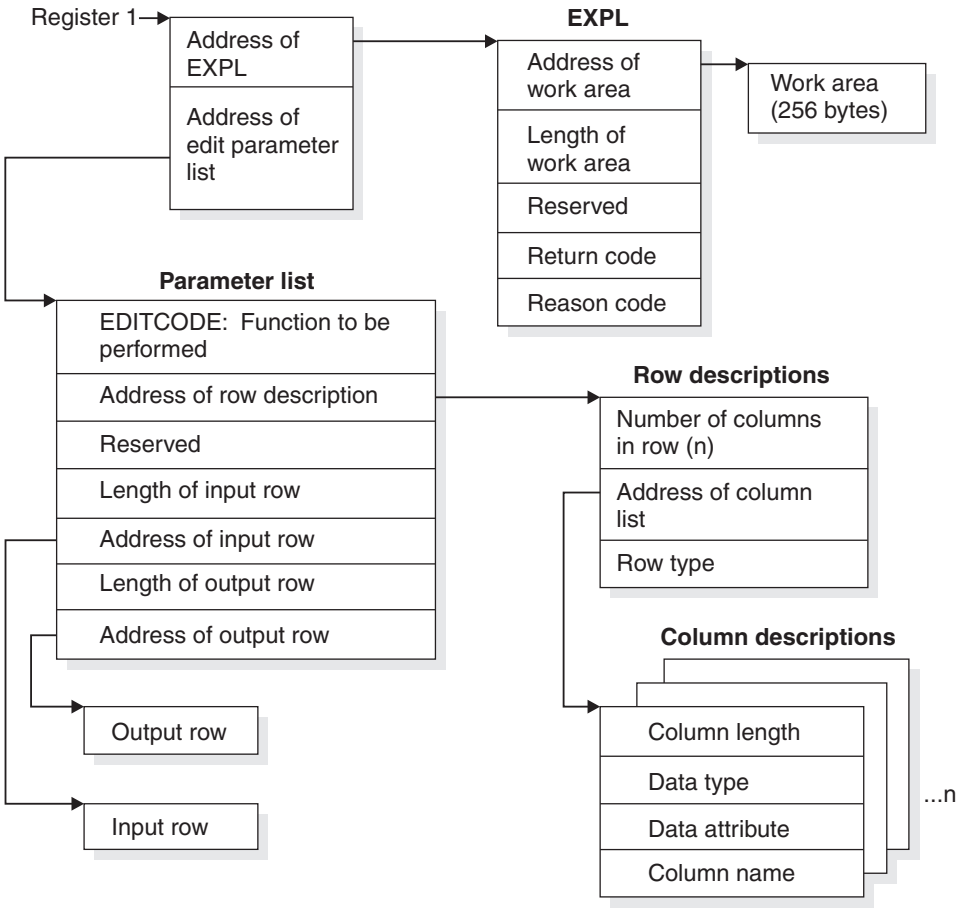


Figure 145. How the edit exit parameter list points to row information. The address of the *n*th column description is given by: RFMTAFLD + (*n*–1)×(FFMTE–FFMT); see “Parameter list for row format descriptions” on page 1108.

Expected output for edit routines

If EDITCODE contains 0, the input row is in decoded form. Your routine must encode it.

In that case, the maximum length of the output area, in EDITOLTH, is 10 bytes more than the maximum length of the record. In counting the maximum length, "record" includes fields for the lengths of VARCHAR and VARGRAPHIC columns, and for null indicators, but does not include the 6-byte record header.

If EDITCODE contains 4, the input row is in coded form. Your routine must decode it.

In that case, EDITOLTH contains the maximum length of the record. As before, "record" includes fields for the lengths of VARCHAR and VARGRAPHIC columns, and for null indicators, but not the 6-byte record header.

In either case, put the result in the output area, pointed to by EDITOPTR, and put the length of your result in EDITOLTH. The length of your result must not be greater than the length of the output area, as given in EDITOLTH on invocation, and your routine must not modify storage beyond the end of the output area.

Required return code: Your routine must also leave a return code in EXPLRC1, with the meanings that are listed in Table 239:

Table 239. Required return code in EXPLRC1

Value	Meaning
0	Function performed successfully.
Nonzero	Function failed.

If the function fails, the routine might also leave a reason code in EXPLRC2. DB2 returns SQLCODE -652 (SQLSTATE '23506') to the application program and puts the reason code in field SQLERRD(6) of the SQL communication area (SQLCA).

Validation routines

Validation routines are assigned to a table by the VALIDPROC clause of CREATE TABLE and ALTER TABLE. A validation routine receives an entire row of a base table as input, and can return an indication of whether or not to allow a following INSERT, UPDATE, or DELETE operation. Typically, a validation routine is used to impose limits on the information that can be entered in a table; for example, allowable salary ranges, perhaps dependent on job category, for the employee sample table.

Although VALIDPROCs can be specified for a table that contains a LOB column, the LOB values are not passed to the validation routine. The indicator column takes the place of the LOB column.

You cannot use VALIDPROC on a table with a column name that is longer than 18
EBCDIC bytes. If you do, you will receive an error message.

The return code from a validation routine is checked for a 0 value before any insert, update, or delete is allowed.

"General considerations for writing exit routines" on page 1104 applies to validation routines.

Specifying the validation routine

To name a validation routine for a table, use the VALIDPROC clause of the CREATE TABLE or ALTER TABLE statement, followed by the name of the routine. In operation, the routine is loaded on demand.

You can add a validation routine to a table that is already in existence, but it is not invoked to validate data already in the table. You can also cancel any validation routine for a table, by using VALIDPROC NULL in an ALTER TABLE statement.

When the validation routine is taken

A validation routine for a table is invoked when DB2 inserts or updates a row, including inserts made by the LOAD utility. The routine is invoked for most delete operations, including a mass delete of all the rows of a table made by a DELETE statement without a WHERE clause. If there are other exit routines, the validation routine is invoked *before* any edit routine, and *after* any date routine, time routine, or field procedure.

Parameter lists for the validation routine

At invocation, registers are set as described in "Registers at invocation for exit routines" on page 1105, and the validation routine uses the standard exit parameter list (EXPL) described there. Table 240 shows the exit-specific parameter list, described by macro DSNDRVAL.

Table 240. Parameter list for a validation routine

Name	Hex offset	Data type	Description
	0	Signed 4-byte integer	Reserved
RVALROW	4	Address	Address of a row description. The format of the row description is shown in Table 280 on page 1109.
	8	Signed 4-byte integer	Reserved
RVALROWL	C	Signed 4-byte integer	Length of the input row to be validated
RVALROWP	10	Address	Address of the input row to be validated
	14	Signed 4-byte integer	Reserved
	18	Signed 4-byte integer	Reserved
RVALPLAN	1C	Character, 8 bytes	Name of the plan issuing the request
RVALOPER	24	Unsigned 1-byte integer	Code identifying the operation being performed, as follows: 1 Insert, update, or load 2 Delete
RVALFL1	25	Character, 1 byte	The high-order bit is on if the requester has installation SYSADM authority. The remaining 7 bits are reserved.
RVALCSTC	26	Character, 2 bytes	Connection system type code. Values are defined in macro DSNDCSTC.

Processing requirements for validation routines

Your routine must be based on the DB2 data formats; see “Row formats for edit and validation routines” on page 1106.

Incomplete rows and validation routines

Sometimes DB2 passes, to a validation routine, an input row that has fewer fields than there are columns in the table. In that case, the routine must stop processing the row after the last input field. Columns for which no input field is provided are always at the end of the row and are never defined as NOT NULL; either they allow nulls, they are defined as NOT NULL WITH DEFAULT, or the column is a ROWID column.

Use macro DSNDRVAL to get the starting address and row length for validation exits. Add the row length to the starting address to get the first invalid address beyond the end of the input buffer; your routine must **not** process any address as large as that.

Expected output for validation routines

Your routine must leave a return code in EXPLRC1, with the meanings that are listed in Table 241.

Table 241. Required return code in EXPLRC1

Value	Meaning
0	Allow insert, update, or delete
Nonzero	Do not allow insert, update, or delete

If the operation is not allowed, the routine might also leave a reason code in EXPLRC2. DB2 returns SQLCODE -652 (SQLSTATE ’23506’) to the application program and puts the reason code in field SQLERRD(6) of the SQL communication area (SQLCA).

Figure 146 on page 1083 shows how the parameter list points to other information.

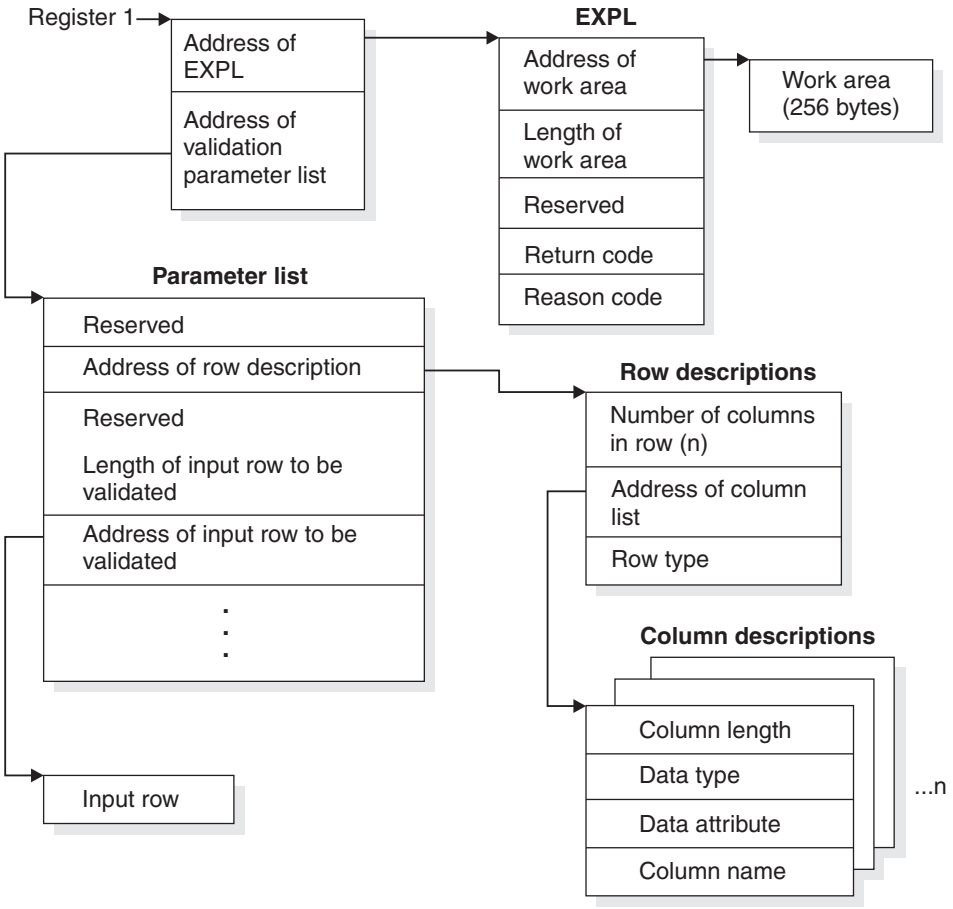


Figure 146. How a validation parameter list points to information. The address of the *n*th column description is given by: RFMTAFLD + (*n*–1)×(FFMTE–FFMT); see “Parameter list for row format descriptions” on page 1108.

Date and time routines

A date routine is a user-written exit routine to change date values from a locally defined format into a format recognized by DB2, when loading or inserting them into a column with data type DATE; and from the ISO format into the locally-defined format, when retrieving the values and assigning them to a host variable. Similarly, a time routine changes time values from a locally-defined format into one recognized by DB2, and from ISO into the locally-defined format. Table 242 shows the formats recognized by DB2.

Table 242. Date and time formats

Format name	Abbreviation	Typical date	Typical time
IBM European standard	EUR	25.12.2004	13.30.05
International Standards Organization	ISO	2004-12-25	13.30.05
Japanese Industrial Standard Christian Era	JIS	2004-12-25	13:30:05
IBM USA standard	USA	12/25/2004	1:30 PM

Example: Suppose that you want to insert and retrieve dates in a format like "September 21, 1992". You can use a date routine that transforms the date to a format that is recognized by DB2 on insertion, such as ISO: "2004-09-21". On retrieval, the routine can transform "2004-09-21" to "September 21, 2004".

You can have either a date routine, a time routine, or both. These routines do not apply to timestamps. Special rules apply if you execute queries at a remote DBMS, through the distributed data facility. For that case, see *DB2 SQL Reference*.

Specifying the date and time routine

To establish a date or time routine, set LOCAL DATE LENGTH or LOCAL TIME LENGTH, when installing DB2, to the length of the longest field required to hold a date or time in your local format. Allowable values range from 10 to 254. For example, if you intend to insert and retrieve dates in the form "September 21, 2004", then you need an 18-byte field. Set LOCAL DATE LENGTH to 18.

Also, replace all of the IBM-supplied exit routines, using CSECTs DSNXVDTX,
DSNXVDTA, and DSNXVDTU for a date routine, and DSNXVTMX, DSNXVTMA,
and DSNXVTMU for a time routine. The routines are loaded when DB2 starts.

To make the local date or time format the default for retrieval, set DATE FORMAT or TIME FORMAT to LOCAL when installing DB2. That has the effect that DB2 *always* takes the exit routine when you retrieve from a DATE or TIME column. In our example, suppose that you want to retrieve dates in your local format only occasionally; most of the time you use the USA format. Set DATE FORMAT to USA.

The install parameters for LOCAL DATE LENGTH, LOCAL TIME LENGTH, DATE FORMAT, and TIME FORMAT can also be updated after DB2 is installed. For instructions, see Part 2 of *DB2 Installation Guide*. If you change a length parameter, you may have to rebind applications.

When date and time routines are taken

On insertion: A date or time routine is invoked to change a value from the locally-defined format to a format recognized by DB2 in the following circumstances:

- When a date or time value is entered by an INSERT or UPDATE statement, or by the LOAD utility
- When a constant or host variable is compared to a column with a data type of DATE, TIME, or TIMESTAMP
- When the DATE or TIME scalar function is used with a string representation of a date or time in LOCAL format
- When a date or time value is supplied for a limit of a partitioned index in a CREATE INDEX statement

The exit is taken before any edit or validation routine.

- **If the default is LOCAL**, DB2 takes the exit immediately. If the exit routine does not recognize the data (EXPLRC1=8), DB2 then tries to interpret it as a date or time in one of the recognized formats (EUR, ISO JIS, or USA). DB2 rejects the data only if that interpretation also fails.
- **If the default is not LOCAL**, DB2 first tries to interpret the data as a date or time in one of the recognized formats. If that interpretation fails, DB2 then takes the exit routine, if it exists.

DB2 checks that the value supplied by the exit routine represents a valid date or time in some recognized format, and then converts it into an internal format for storage or comparison. If the value is entered into a column that is a key column in an index, the index entry is also made in the internal format.

On retrieval: A date or time routine can be invoked to change a value from ISO to the locally-defined format when a date or time value is retrieved by a SELECT or FETCH statement. If LOCAL is the default, the routine is always invoked unless overridden by a precompiler option or by the CHAR function, as by specifying CHAR(HIREDATE, ISO); that specification always retrieves a date in ISO format. If LOCAL is not the default, the routine is invoked only when specifically called for by CHAR, as in CHAR(HIREDATE, LOCAL); that always retrieves a date in the format supplied by your date exit routine.

On retrieval, the exit is invoked after any edit routine or DB2 sort. A date or time routine is not invoked for a DELETE operation without a WHERE clause that deletes an entire table in a segmented table space.

Parameter lists for date and time routines

At invocation, registers are set as described in "Registers at invocation for exit routines" on page 1105, and the date or time routine uses the standard exit parameter list (EXPL) described there. Table 243 shows its exit-specific parameter list, described by macro DSNDDTXP.

Table 243. Parameter list for a date or time routine

Name	Hex offset	Data type	Description
DTXPFN	0	Address	Address of a 2-byte integer containing a function code. The codes and their meanings are: 4 Convert from local format to ISO. 8 Convert from ISO to local format.
DTXPLN	4	Address	Address of a 2-byte integer containing the length in bytes of the local format. This is the length given as LOCAL DATE LENGTH or LOCAL TIME LENGTH when installing DB2.
DTXPLOC	8	Address	Address of the date or time value in local format
DTXPISO	C	Address	Address of the date or time value in ISO format (DTXPISO). The area pointed to is 10 bytes long for a date, 8 bytes for a time.

Expected output for date and time routines

If the function code is 4, the input value is in local format, in the area pointed to by DTXPLOC. Your routine must change it to ISO, and put the result in the area pointed to by DTXPISO.

If the function code is 8, the input value is in ISO, in the area pointed to by DTXPISO. Your routine must change it to your local format, and put the result in the area pointed to by DTXPLOC.

Your routine must also leave a return code in EXPLRC1, a 4-byte integer and the third word of the EXPL area. The return code can have the meanings that are shown in Table 244 on page 1086.

Table 244. Required return code in EXPLRC1

Value	Meaning
0	No errors; conversion was completed.
4	Invalid date or time value.
8	Input value not in valid format; if the function is insertion, and LOCAL is the default, DB2 next tries to interpret the data as a date or time in one of the recognized formats (EUR, ISO, JIS, or USA).
12	Error in exit routine.

Figure 147 shows how the parameter list points to other information.

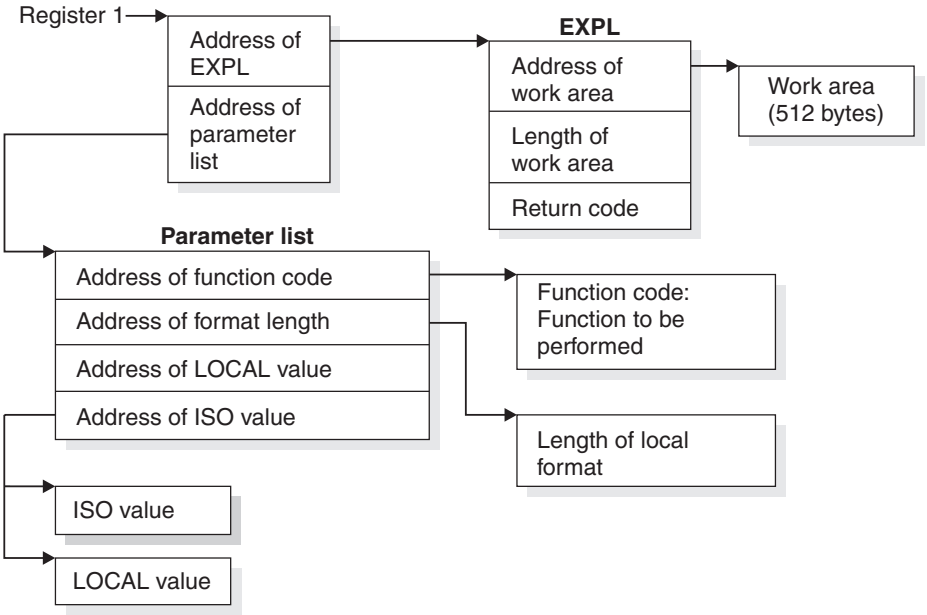


Figure 147. How a date or time parameter list points to other information

Conversion procedures

A conversion procedure is a user-written exit routine that converts characters from one coded character set to another coded character set. (For a general discussion of character sets, and definitions of those terms, see Appendix A of*DB2 Installation Guide*.) In most cases, any conversion that is needed can be done by routines provided by IBM. The exit for a user-written routine is available to handle exceptions.

"General considerations for writing exit routines" on page 1104 applies to conversion routines.

Specifying a conversion procedure

To establish a conversion procedure, insert a row into the catalog table SYSIBM.SYSSTRINGS. The row must contain values for the following columns:

INCCSID The coded character set identifier (CCSID) of the source string.

OUTCCSID The CCSID of the converted string.

TRANSTYPE The nature of the conversion. Values can be:

GG	ASCII GRAPHIC to EBCDIC GRAPHIC
MM	EBCDIC MIXED to EBCDIC MIXED
MP	EBCDIC MIXED to ASCII MIXED
MS	EBCDIC MIXED to EBCDIC SBCS
PM	ASCII MIXED to EBCDIC MIXED
PP	ASCII MIXED to ASCII MIXED
PS	ASCII MIXED to EBCDIC SBCS
SM	EBCDIC SBCS to EBCDIC MIXED
SP	SBCS (ASCII or EBCDIC) to ASCII MIXED
SS	EBCDIC SBCS to EBCDIC SBCS

TRANSPROC The name of your conversion procedure.

IBMREQD Must be N.

DB2 does not use the following columns, but checks them for the allowable values listed. Values you insert can be used by your routine in any way. If you insert no value in one of these columns, DB2 inserts the default value listed.

ERRORBYTE Any character, or null. The default is null.

SUBBYTE Any character not equal to the value of ERRORBYTE, or null. The default is null.

TRANSTAB Any character string of length 256 or the empty string. The default is an empty string.

When conversion procedures are taken

The exit is taken, and your procedure invoked, whenever a conversion is required from the coded character set identified by INCCSID to the coded character set identified by OUTCCSID.

Parameter lists for conversion procedures

At invocation, registers are set as described in "Registers at invocation for exit routines" on page 1105, and the conversion procedure uses the standard exit parameter list (EXPL) described there. A conversion procedure does *not* use an exit-specific parameter list, as described in "Parameter lists for exit routines" on page 1106. Instead, the area pointed to by register 1 at invocation includes three words, which contain the addresses of the following items:

1. The EXPL parameter list
2. A string value descriptor that contains the character string to be converted
3. A copy of a row from SYSIBM.SYSSTRINGS that names the conversion procedure identified in TRANSPROC.

The length of the work area pointed to by the exit parameter list is generally 512 bytes. However, if the string to be converted is ASCII MIXED data (the value of TRANSTYPE in the row from SYSSTRINGS is PM or PS), then the length of the work area is 256 bytes, plus the length attribute of the string.

The string value descriptor: The descriptor has the format shown in Table 245 on page 1088.

Table 245. Format of string value descriptor for a conversion procedure

Name	Hex offset	Data type	Description
FPVDTYPE	0	Signed 2-byte integer	Data type of the value: **Code** **Means** 20 VARCHAR 28 VARGRAPHIC
FPVDVLEN	2	Signed 2-byte integer	The maximum length of the string
FPVDVALE	4	None	The string. The first halfword is the string's actual length in characters. If the string is ASCII MIXED data, it is padded out to the maximum length by undefined bytes.

The row from SYSSTRINGS: The row copied from the catalog table SYSIBM.SYSSTRINGS is in the standard DB2 row format described in "Row formats for edit and validation routines" on page 1106. The fields ERRORBYTE and SUBBYTE each include a null indicator. The field TRANSTAB is of varying length and begins with a 2-byte length field.

Expected output for conversion procedures

Except in the case of certain errors, your conversion procedure should replace the string in FPVDVALE with the converted string. When converting MIXED data, your procedure must ensure that the result is well-formed. In any conversion, if you change the length of the string, you must set the length control field in FPVDVALE to the proper value. Over-writing storage beyond the maximum length of the FPVDVALE causes an abend.

Your procedure must also set a return code in field EXPLRC1 of the exit parameter list.

With the two codes that are shown in Table 246, provide the converted string in FPVDVALE.

Table 246. Codes for the converted string in FPVDVALE

Code	Meaning
0	Successful conversion
4	Conversion with substitution

For the remaining codes that are shown in Table 247, DB2 does not use the converted string.

Table 247. Remaining codes for the FPVDVALE

Code	Meaning
8	Length exception
12	Invalid code point
16	Form exception
20	Any other error
24	Invalid CCSID

Exception conditions: Return a length exception (code 8) when the converted string is longer than the maximum length allowed.

For an invalid code point (code 12), place the 1- or 2-byte code point in field EXPLRC2 of the exit parameter list.

Return a form exception (code 16) for EBCDIC MIXED data when the source string does not conform to the rules for MIXED data.

Any other uses of codes 8 and 16, or of EXPLRC2, are optional.

Error conditions: On return, DB2 considers any of the following conditions as a "conversion error":

- EXPLRC1 is greater than 16.
- EXPLRC1 is 8, 12, or 16 and the operation that required the conversion is *not* an assignment of a value to a host variable with an indicator variable.
- FPVDTYPE or FPVDVLEN has been changed.
- The length control field of FPVDVALE is greater than the original value of FPVDVLEN or is negative.

In the case of a conversion error, DB2 sets the SQLERRMC field of the SQLCA to HEX(EXPLRC1) CONCAT X'FF' CONCAT HEX(EXPLRC2).

Figure 148 shows how the parameter list points to other information.

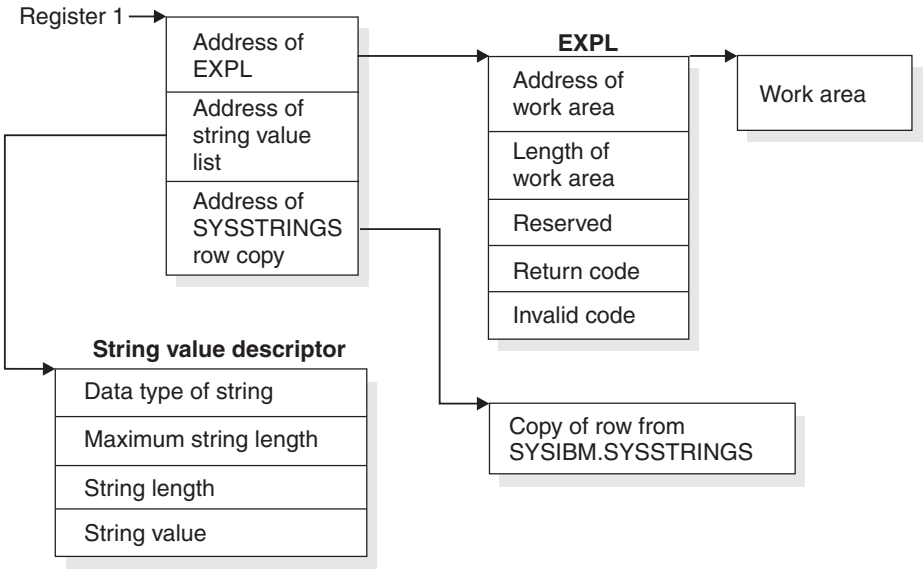


Figure 148. Pointers at entry to a conversion procedure

Field procedures

Field procedures are assigned to a table by the FIELDPROC clause of CREATE TABLE and ALTER TABLE. A field procedure is a user-written exit routine to transform values in a single short-string column. When values in the column are changed, or new values inserted, the field procedure is invoked for each value, and can transform that value (encode it) in any way. The encoded value is then stored. When values are retrieved from the column, the field procedure is invoked for each value, which is encoded, and must decode it back to the original string value.

Any indexes, including partitioned indexes, defined on a column that uses a field procedure are built with encoded values. For a partitioned index, the encoded value of the limit key is put into the LIMITKEY column of the SYSINDEXPART table. Hence, a field procedure might be used to alter the sorting sequence of values entered in a column. For example, telephone directories sometimes require that names like “McCabe” and “MacCabe” appear next to each other, an effect that the standard EBCDIC sorting sequence does not provide. And languages that do not use the Roman alphabet have similar requirements. However, if a column is provided with a suitable field procedure, it can be correctly ordered by ORDER BY.

The transformation your field procedure performs on a value is called *field-encoding*. The same routine is used to undo the transformation when values are retrieved; that operation is called *field-decoding*. Values in columns with a field procedure are described to DB2 in two ways:

1. The description of the column as defined in CREATE TABLE or ALTER TABLE appears in the catalog table SYSIBM.SYSCOLUMNS. That is the description of the field-decoded value, and is called the *column description*.
2. The description of the encoded value, as it is stored in the data base, appears in the catalog table SYSIBM.SYSFIELDS. That is the description of the field-encoded value, and is called the *field description*.

Important: The field-decoding function must be the exact inverse of the field-encoding function. For example, if a routine encodes 'ALABAMA' to '01', it must decode '01' to 'ALABAMA'. A violation of this rule can lead to an abend of the DB2 connecting thread, or other undesirable effects.

“General considerations for writing exit routines” on page 1104 applies to field procedures.

Field definition for field procedures

The field procedure is also invoked when the table is created or altered, to define the data type and attributes of an encoded value to DB2; that operation is called *field-definition*. The data type of the encoded value can be any valid SQL data type except DATE, TIME, TIMESTAMP, LONG VARCHAR, or LONG VARGRAPHIC; the allowable types are listed in the description of field FPVDTYPE in Table 250 on page 1095. The length, precision, or scale of the encoded value must be compatible with its data type.

A user-defined data type can be a valid field if the source type of the data type is a short string column that has a null default value. DB2 casts the value of the column to the source type before it passes it to the field procedure.

Specifying the field procedure

To name a field procedure for a column, use the FIELDPROC clause of the CREATE TABLE or ALTER TABLE statement, followed by the name of the procedure and, optionally, a list of parameters. You can use a field procedure only with a short string column. You cannot use a field procedure on a column defined using NOT NULL WITH DEFAULT.

If you plan to use a field procedure, specify it when you create the table. In operation, the procedure is loaded on demand. You cannot add a field procedure to an existing column of a table; you can, however, use ALTER TABLE to add to an existing table a new column that uses a field procedure.

You cannot use a field procedure on a LOB or a ROWID column. Field procedures can be specified for other columns of a table that contains a LOB or ROWID column.

You cannot use a field procedure on a column if the column name is longer than
18 EBCDIC bytes. If you do, you will receive an error message.

The optional parameter list that follows the procedure name is a list of constants, enclosed in parentheses, called the *literal list*. The literal list is converted by DB2 into a data structure called the *field procedure parameter value list* (FPPVL). That structure is passed to the field procedure during the field-definition operation. At that time, the procedure can modify it or return it unchanged. The output form of the FPPVL is called the *modified FPPVL*; it is stored in the DB2 catalog as part of the field description. The modified FPPVL is passed again to the field procedure whenever that procedure is invoked for field-encoding or field-decoding.

When field procedures are taken

A field procedure specified for a column is invoked in three general situations:

1. **For field-definition**, when the CREATE TABLE or ALTER TABLE statement that names the procedure is executed. During this invocation, the procedure is expected to:
   - Determine whether the data type and attributes of the column are valid.
   - Verify the literal list, and change it if wanted.
   - Provide the field description of the column.
   - Define the amount of working storage needed by the field-encoding and field-decoding processes.
2. **For field-encoding**, when a column value is to be field-encoded. That occurs for any value that:
   - Is inserted in the column by an SQL INSERT statement, or loaded by the DB2 LOAD utility.
   - Is changed by an SQL UPDATE statement.
   - Is compared to a column with a field procedure, unless the comparison operator is LIKE. The value being encoded is a host variable or constant. (When the comparison operator is LIKE, the column value is decoded.)
   - Defines the limit of a partition of an index. The value being encoded follows ENDING AT in the PARTITION clause of CREATE INDEX.

   If there are any other exit routines, the field procedure is invoked *before* any of them.
3. **For field-decoding**, when a stored value is to be field-decoded back into its original string value. This occurs for any value that is:
   - Retrieved by an SQL SELECT or FETCH statement, or by the unload phase of the REORG utility.
   - Compared to another value with the LIKE comparison operator. The value being decoded is from the column that uses the field procedure.

   In this case, the field procedure is invoked *after* any edit routine or DB2 sort.

A field procedure is never invoked to process a null value, nor for a DELETE operation without a WHERE clause on a table in a segmented table space.

A warning about blanks: When DB2 compares the values of two strings with different lengths, it temporarily pads the shorter string with blanks (in EBCDIC or

double-byte characters, as needed) up to the length of the longer string. If the shorter string is the value of a column with a field procedure, the padding is done to the encoded value, but the pad character is **not** encoded. Therefore, if the procedure changes blanks to some other character, encoded blanks at the end of the longer string are not equal to padded blanks at the end of the shorter string. That situation can lead to errors; for example, some strings that ought to be equal might not be recognized as such. Therefore, encoding blanks in a field procedure is not recommended.

Control blocks for execution of field procedures

This section describes certain control blocks that are used to communicate to a field procedure:

- "The field procedure parameter list (FPPL)"
- "The work area for field procedures" on page 1093
- "The field procedure information block (FPIB)" on page 1093
- "The field procedure parameter value list (FPPVL)" on page 1094
- "Value descriptors for field procedures" on page 1094

Next, this section describes specific requirements for the three operations of field-definition:

- "Field-definition (function code 8)" on page 1095
- "Field-encoding (function code 0)" on page 1098
- "Field-decoding (function code 4)" on page 1099

The contents of registers at invocation and at exit are different for each of those operations, and are described with the requirements for the operations.

The field procedure parameter list (FPPL)

The field procedure parameter list is pointed to by register 1 on entry to a field procedure. It, in turn, contains the addresses of five other areas, as shown in Figure 149 on page 1093. Those areas are described in the following pages. The FPPL and the areas it points to are all described by the mapping macro DSNDFPPB.

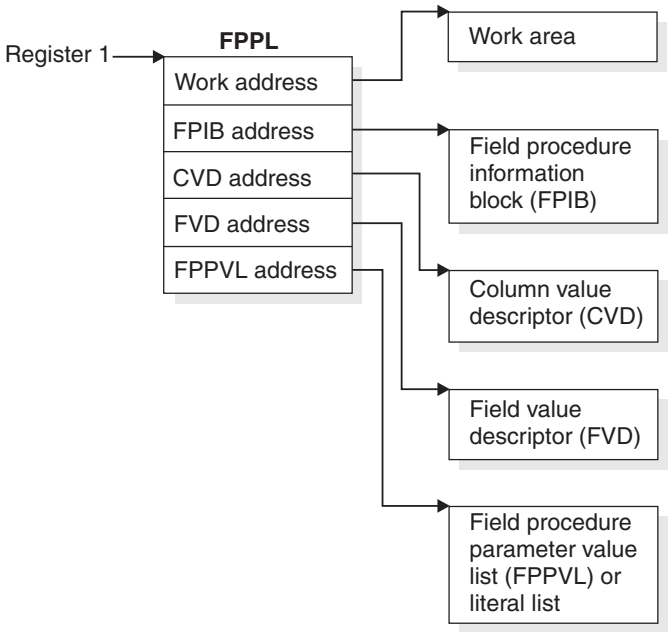


Figure 149. Field procedure parameter list

The work area for field procedures

The work area is a contiguous, uninitialized area of locally addressable, pageable, swappable, and fetch-protected storage that is obtained in storage key 7 and subpool 229.

The work area can be used by a field procedure as working storage. A new area is provided each time the procedure is invoked. The size of the area that you need depends on the way you program your field-encoding and field-decoding operations.

At field-definition time, DB2 allocates a 512-byte work area and passes the value of 512 bytes as the work area size to your routine for the field-definition operation. If subsequent field-encoding and field-decoding operations need a work area of 512 bytes or less, your field definition doesn't need to change the value as provided by DB2. If those operations need a work area larger than 512 bytes (i.e. 1024 bytes), your field definition must change the work area size to the larger size and pass it back to DB2 for allocation.

Whenever your field procedure is invoked for encoding or decoding operations, DB2 allocates a work area based on the size (i.e. 1024 bytes) that was passed back to it. Your field definition must not use a work area larger than what is allocated by DB2, even though subsequent operations need the larger work area.

The field procedure information block (FPIB)

The field procedure information block communicates general information to a field procedure. For example, it tells what operation is to be done, allows the field procedure to signal errors, and gives the size of the work area.

It has the format shown in Table 248.

Table 248. Format of FPIB, defined in copy macro DSNDFPPB

Name	Hex offset	Data type	Description
FPBFCODE	0	Signed 2-byte integer	Function code **Code** **Means** 0 Field-encoding 4 Field-decoding 8 Field-definition
FPBWKLN	2	Signed 2-byte integer	Length of work area; the maximum is 32767 bytes.
FPBSORC	4	Signed 2-byte integer	Reserved
FPBRTNC	6	Character, 2 bytes	Return code set by field procedure
FPBRSNCD	8	Character, 4 bytes	Reason code set by field procedure
FPBTOKPT	C	Address	Address of a 40-byte area, within the work area or within the field procedure's static area, containing an error message

The field procedure parameter value list (FPPVL)

The field procedure parameter value list communicates the literal list, supplied in the CREATE TABLE or ALTER TABLE statement, to the field procedure during field-definition. At that time the field procedure can reformat the FPPVL; it is the reformatted FPPVL that is stored in SYSIBM.SYSFIELDS and communicated to the field procedure during field-encoding and field-decoding as the *modified FPPVL*.

The FPPVL has the format shown in Table 249.

Table 249. Format of FPPVL, defined in copy macro DSNDFPPB

Name	Hex offset	Data type	Description
FPPVLEN	0	Signed 2-byte integer	Length in bytes of the area containing FPPVCNT and FPPVVDS. At least 254 for field-definition.
FPPVCNT	2	Signed 2-byte integer	Number of value descriptors that follow, equal to the number of parameters in the FIELDPROC clause. Zero if no parameters were listed.
FPPVVDS	4	Structure	For each parameter in the FIELDPROC clause, there is: 1. A signed 4-byte integer giving the length of the following value descriptor, which includes the lengths of FPVDTYPE, FPVDLEN, and FPVDVALE. 2. A value descriptor

Value descriptors for field procedures

A value descriptor describes the data type and other attributes of a value. Value descriptors are used with field procedures in these ways:

- During field-definition, they describe each constant in the field procedure parameter value list (FPPVL). The set of these value descriptors is part of the FPPVL control block.
- During field-encoding and field-decoding, the decoded (column) value and the encoded (field) value are described by the column value descriptor (CVD) and the field value descriptor (FVD).

The *column value descriptor (CVD)* contains a description of a column value and, if appropriate, the value itself. During field-encoding, the CVD describes the value to be encoded. During field-decoding, it describes the decoded value to be supplied by the field procedure. During field-definition, it describes the column as defined in the CREATE TABLE or ALTER TABLE statement.

The *field value descriptor (FVD)* contains a description of a field value and, if appropriate, the value itself. During field-encoding, the FVD describes the encoded value to be supplied by the field procedure. During field-decoding, it describes the value to be decoded. Field-definition must put into the FVD a description of the encoded value.

Value descriptors have the format shown in Table 250.

Table 250. Format of value descriptors

Name	Hex offset	Data type	Description
FPVDTYPE	0	Signed 2-byte integer	Data type of the value: **Code** **Means** 0 INTEGER 4 SMALLINT 8 FLOAT 12 DECIMAL 16 CHAR 20 VARCHAR 24 GRAPHIC 28 VARGRAPHIC
FPVDVLEN	2	Signed 2-byte integer	• For a varying-length string value, its maximum length • For a decimal number value, its precision (byte 1) and scale (byte 2) • For any other value, its length
FPVDVALE	4	None	The value. The value is in external format, not DB2 internal format. If the value is a varying-length string, the first halfword is the value's actual length in bytes. This field is not present in a CVD, or in an FVD used as input to the field-definition operation. An empty varying-length string has a length of zero with no data following.

Field-definition (function code 8)

The input provided to the field-definition operation, and the output required, are as follows:

On entry

The **registers** have the information that is listed in Table 251:

Table 251. Contents of the registers on entry

Register	Contains
1	Address of the field procedure parameter list (FPPL); see Figure 149 on page 1093 for a schematic diagram.
2 through 12	Unknown values that must be restored on exit.
13	Address of the register save area.
14	Return address.
15	Address of entry point of exit routine.

The contents of all other registers, and of fields not listed in the following tables, are unpredictable.

The **work area** consists of 512 contiguous uninitialized bytes.

The **FPIB** has the information that is listed in Table 252:

Table 252. Contents of the FPIB on entry

Field	Contains
FPBFCODE	8, the function code
FPBWKLN	512, the length of the work area

The **CVD** has the information that is listed in Table 253:

Table 253. Contents of the CVD on entry

Field	Contains
FPVDTYPE	One of these codes for the data type of the column value: **Code** **Means** 16 CHAR 20 VARCHAR 24 GRAPHIC 28 VARGRAPHIC
FPVDVLEN	The length attribute of the column

The FPVDVALE field is omitted.

The **FVD** provided is 4 bytes long.

The **FPPVL** has the information that is listed in Table 254:

Table 254. Contents of the FPPVL on entry

Field	Contains
FPPVLEN	The length, in bytes, of the area containing the parameter value list. The minimum value is 254, even if there are no parameters.
FPPVCNT	The number of value descriptors that follow; zero if there are no parameters.

Table 254. Contents of the FPPVL on entry (continued)

Field	Contains
FPPVVDS	A contiguous set of value descriptors, one for each parameter in the parameter value list, each preceded by a 4-byte length field.

On exit

The **registers** must have the information that is listed in Table 255:

Table 255. Contents of the registers on exit

Register	Contains
2 through 12	The values that they contained on entry.
15	The integer zero if the column described in the CVD is valid for the field procedure; otherwise the value must *not* be zero.

The following fields must be set as shown; all other fields must remain as on entry.

The **FPIB** must have the information that is listed in Table 256:

Table 256. Contents of the FPIB on exit

Field	Contains
FPBWKLN	The length, in bytes, of the work area to be provided to the field-encoding and field-decoding operations; 0 if no work area is required.
FPBRTNC	An optional 2-byte character return code, defined by the field procedure; blanks if no return code is given.
FPBRSNC	An optional 4-byte character reason code, defined by the field procedure; blanks if no reason code is given.
FPBTOKP	Optionally, the address of a 40-byte error message residing in the work area or in the field procedure's static area; zeros if no message is given.

Errors signalled by a field procedure result in SQLCODE -681 (SQLSTATE '23507'), which is set in the SQL communication area (SQLCA). The contents of FPBRTNC and FPBRSNC, and the error message pointed to by FPBTOKP, are also placed into the tokens, in SQLCA, as field SQLERRMT. The meaning of the error message is determined by the field procedure.

The **FVD** must have the information that is listed in Table 257:

Table 257. Contents of the FVD on exit

Field	Contains
FPVDTYPE	The numeric code for the data type of the field value. Any of the data types listed in Table 250 on page 1095 is valid.
FPVDVLEN	The length of the field value.

Field FPVDVALE must not be set; the length of the FVD is 4 bytes only.

The **FPPVL** can be redefined to suit the field procedure, and returned as the *modified FPPVL*, subject to the following restrictions:

- The field procedure must not increase the length of the FPPVL.

- FPPVLEN must contain the actual length of the modified FPPVL, or 0 if no parameter list is returned.

The modified FPPVL is recorded in the catalog table SYSIBM.SYSFIELDS, and is passed again to the field procedure during field-encoding and field-decoding. The modified FPPVL need not have the format of a field procedure parameter list, and it need not describe constants by value descriptors.

Field-encoding (function code 0)

The input provided to the field-encoding operation, and the output required, are as follows:

On entry

The **registers** have the information that is listed in Table 258:

Table 258. Contents of the registers on entry

Register	Contains
1	Address of the field procedure parameter list (FPPL); see Figure 149 on page 1093 for a schematic diagram.
2 through 12	Unknown values that must be restored on exit.
13	Address of the register save area.
14	Return address.
15	Address of entry point of exit routine.

The contents of all other registers, and of fields not listed, are unpredictable.

The **work area** is contiguous, uninitialized, and of the length specified by the field procedure during field-definition.

The **FPIB** has the information that is listed in Table 259:

Table 259. Contents of the FPIB on entry

Field	Contains
FPBFCODE	0, the function code
FPBWKLN	The length of the work area

The **CVD** has the information that is listed in Table 260:

Table 260. Contents of the CVD on entry

Field	Contains
FPVDTYPE	The numeric code for the data type of the column value, as shown in Table 250 on page 1095.
FPVDVLEN	The length of the column value.
FPVDVALE	The column value; if the value is a varying-length string, the first halfword contains its length.

The **FVD** has the information that is listed in Table 261:

Table 261. Contents of the FVD on entry

Field	Contains
FPVDTYPE	The numeric code for the data type of the field value.
FPVDVLEN	The length of the field value.
FPVDVALE	An area of unpredictable content that is as long as the field value.

The *modified* FPPVL, produced by the field procedure during field-definition, is provided.

On exit

The **registers** have the information that is listed in Table 262:

Table 262. Contents of the registers on exit

Register	Contains
2 through 12	The values that they contained on entry.
15	The integer zero if the column described in the CVD is valid for the field procedure; otherwise the value must not be zero.

The **FVD** must contain the encoded (field) value in field FPVDVALE. If the value is a varying-length string, the first halfword must contain its length.

The **FPIB** can have the information that is listed in Table 263:

Table 263. Contents of the FPIB on exit

Field	Contains
FPBRTNC	An optional 2-byte character return code, defined by the field procedure; blanks if no return code is given.
FPBRSNC	An optional 4-byte character reason code, defined by the field procedure; blanks if no reason code is given.
FPBTOKP	Optionally, the address of a 40-byte error message residing in the work area or in the field procedure's static area; zeros if no message is given.

Errors signalled by a field procedure result in SQLCODE -681 (SQLSTATE '23507'), which is set in the SQL communication area (SQLCA). The contents of FPBRTNC and FPBRSNC, and the error message pointed to by FPBTOKP, are also placed into the tokens, in SQLCA, as field SQLERRMT. The meaning of the error message is determined by the field procedure.

All other fields must remain as on entry.

Field-decoding (function code 4)

The input provided to the field-decoding operation, and the output required, are as follows:

On entry

The **registers** have the information that is listed in Table 264:

Table 264. Contents of the registers on entry

Register	Contains
1	Address of the field procedure parameter list (FPPL); see Figure 149 on page 1093 for a schematic diagram.
2 through 12	Unknown values that must be restored on exit.
13	Address of the register save area.
14	Return address.
15	Address of entry point of exit routine.

The contents of all other registers, and of fields not listed, are unpredictable.

The **work area** is contiguous, uninitialized, and of the length specified by the field procedure during field-definition.

The **FPIB** has the information that is listed in Table 265:

Table 265. Contents of the FPIB on entry

Field	Contains
FPBFCODE	4, the function code
FPBWKLN	The length of the work area

The **CVD** has the information that is listed in Table 266:

Table 266. Contents of the CVD on entry

Field	Contains
FPVDTYPE	The numeric code for the data type of the column value, as shown in Table 250 on page 1095.
FPVDVLEN	The length of the column value.
FPVDVALE	The column value. If the value is a varying-length string, the first halfword contains its length.

The **FVD** has the information that is listed in Table 267:

Table 267. Contents of the FVD on entry

Field	Contains
FPVDTYPE	The numeric code for the data type of the field value.
FPVDVLEN	The length of the field value.
FPVDVALE	The field value. If the value is a varying-length string, the first halfword contains its length.

The *modified* FPPVL, produced by the field procedure during field-definition, is provided.

On exit

The **registers** have the information that is listed in Table 268:

Table 268. Contents of the registers on exit

Register	Contains
2 through 12	The values they contained on entry.
15	The integer zero if the column described in the FVD is valid for the field procedure; otherwise the value must not be zero.

The **CVD** must contain the decoded (column) value in field FPVDVALE. If the value is a varying-length string, the first halfword must contain its length.

The **FPIB** can have the information that is listed in Table 269:

Table 269. Contents of the FPIB on exit

Field	Contains
FPBRTNC	An optional 2-byte character return code, defined by the field procedure; blanks if no return code is given.
FPBRSNC	An optional 4-byte character reason code, defined by the field procedure; blanks if no reason code is given.
FPBTOKP	Optionally, the address of a 40-byte error message residing in the work area or in the field procedure's static area; zeros if no message is given.

Errors signalled by a field procedure result in SQLCODE -681 (SQLSTATE '23507'), which is set in the SQL communication area (SQLCA). The contents of FPBRTNC and FPBRSNC, and the error message pointed to by FPBTOKP, are also placed into the tokens, in SQLCA, as field SQLERRMT. The meaning of the error message is determined by the field procedure.

All other fields must remain as on entry.

Log capture routines

A log capture exit routine makes DB2 log data available for recovery purposes in real time. The routine receives data when DB2 writes data to the active log. Your local specifications determine what the routine does with that data. The routine does not enter or return data to DB2.

Performance factor: Your log capture routine receives control often. Design it with care: a poorly designed routine can seriously degrade system performance. Whenever possible, use the instrumentation facility interface (IFI), rather than a log capture exit routine, to read data from the log. For instructions, see "Reading log records with IFI" on page 1122.

"General considerations for writing exit routines" on page 1104 applies, but with the following exceptions to the description of execution environments:

A log capture routine can execute in either TCB mode or SRB mode, depending on the function it is performing. When in SRB mode, it must not perform any I/O operations nor invoke any SVC services or ESTAE routines.

Specifying the log capture routine

The module name for the routine is DSNJL004. Its entry point is DSNJW117.

The module is loaded during DB2 initialization and deleted during DB2 termination. You must link the module into either the *prefix*.SDSNEXIT or the DB2 *prefix*.SDSNLOAD library. Specify the REPLACE parameter of the link-edit job to replace a module that is part of the standard DB2 library for this release. The module should have attributes AMODE(31) and RMODE(ANY).

When log capture routines are taken

The log capture exit is taken in three possible situations, identified by a character in the exit parameter list. In two of those situations, processing operates in TCB mode; in one situation, processing operates in SRB mode. The two modes have different processing capabilities, which your routine must be aware of. The character identifications, situations, and modes are:

- I=Initialization, Mode=TCB

  The TCB mode allows all z/OS DFSMSdfp functions to be utilized, including ENQ, ALLOCATION, and OPEN. No buffer addresses are passed in this situation. The routine runs in supervisor state, key 7, and enabled.

  This is the **only** situation in which DB2 checks a return code from the user's log capture exit routine. The DB2 subsystem is sensitive to a return code of X'20' here. **Never return X'20'** in register 15 in this situation.
- W=Write, Mode=SRB (service request block)

  The SRB mode restricts the exit routine's processing capabilities. No supervisor call (SVC) instructions can be used, including ALLOCATION, OPEN, WTO, any I/O instruction, and so on. At the exit point, DB2 is running in supervisor state, key 7, and is enabled.

  Upon entry, the exit routine has access to buffers that have log control intervals with “blocked log records”. The first and last buffer address and control interval size fields can be used to determine how many buffers are being passed.

  See *z/OS MVS Programming: Authorized Assembler Services Guide* for additional material on SRB-mode processing.

  Performance warning: All processing time required by the exit routine lengthens the time required to write the DB2 log. The DB2 address space usually has a high priority, and all work done in it in SRB mode precedes all TCB access, so any errors or long processing times can impact all DB2 processing and cause system-wide performance problems. The performance of your routine is *extremely* critical in this phase.
- T=Termination, Mode=TCB

  Processing capabilities are the same as for initialization.

A log control interval can be passed more than once. Use the time stamp to determine the last occurrence of the control interval. This last occurrence should replace all others. The time stamp is found in the control interval.

Parameter lists for log capture routines

At invocation, registers are set as described in “Registers at invocation for exit routines” on page 1105, and the log capture routine uses the standard exit parameter list (EXPL) described there. (The reason and return codes in that list can be ignored.) Table 270 on page 1103 shows the exit-specific parameter list; it is mapped by macro DSNDLOGX.

Table 270. Log capture routine specific parameter list

Name	Hex offset	Data type	Description
LOGXEYE	00	Character, 4 bytes	Eye catcher: LOGX
LOGXLNG	04	Signed 2-byte integer	Length of parameter list
	06		Reserved
	08		Reserved
LOGXTYPE	10	Character, 1 byte	Situation identifier: **I** Initialization **W** Write **T** Termination **P** Partial control interval (CI) call
LOGXFLAG	11	Hex	Mode identifier. **X'00'** SRB mode **X'01'** TCB mode
LOGXSRBA	12	Character, 6 bytes	First log RBA, set when DB2 is started. The value remains constant while DB2 is active.
LOGXARBA	18	Character, 6 bytes	Highest log archive RBA used. The value is updated after completion of each log archive operation.
	1E		Reserved
LOGXRBUF	20	Character, 8 bytes	Range of consecutive log buffers: Address of first log buffer Address of last log buffer
LOGXBUFL	28	Signed 4-byte integer	Length of single log buffer (constant 4096)
LOGXSSID	2C	Character, 4 bytes	DB2 subsystem ID, 4 characters left justified
LOGXSTIM	30	Character, 8 bytes	DB2 subsystem startup time (TIME format with DEC option: 0CYYDDDFHHMMSSTH)
LOGXREL	38	Character, 3 bytes	DB2 subsystem release level
LOGXMAXB	3B	Character, 1 byte	Maximum number of buffers that can be passed on one call. The value remains constant while DB2 is active.
	3C	8 bytes	Reserved
LOGXUSR1	44	Character, 4 bytes	First word of a doubleword work area for the user routine. (The content is not changed by DB2.)
LOGXUSR2	48	Character, 4 bytes	Second word of user work area.

Routines for dynamic plan selection in CICS

You can create application packages and plans that allow application programs to access DB2 data at execution time. In a CICS environment, you can design CICS transactions around the application packages and plans or use dynamic plan allocation.

You can enable dynamic plan allocation by using one of the following techniques:

- Use DB2 packages and versioning to manage the relationship between CICS transactions and DB2 plans. This technique can help minimize plan outage time, processor time, and catalog contention.
- Use a dynamic plan exit routine to determine the plan to use for each CICS transaction.

Recommendation: Use DB2 packages and versioning, instead of a CICS dynamic plan exit routine, for dynamic plan allocation. For more information on using packages, see *CICS Transaction Server for z/OS: DB2 Guide*.

For guidance on using packages, see *DB2 Application Programming and SQL Guide*.

Routine for CICS transaction invocation stored procedure

The DB2-supplied CICS transaction routine stored procedure invokes a user exit that you use to change values that the stored procedure caller provides. For information about this stored procedure, see Appendix I of *DB2 Application Programming and SQL Guide*.

General considerations for writing exit routines

The rules, requirements, and suggestions in this section apply to most of the foregoing exit routines.

Important: Using an exit routine requires coordination with your system programmers. An exit routine runs as an extension of DB2 and has all the privileges of DB2. It can impact the security and integrity of the database. Conceivably, an exit routine could also expose the integrity of the operating system. Instructions for avoiding that exposure can be found in the appropriate z/OS publication.

Coding rules for exit routines

An exit routine must conform to these rules:

- It must be written in assembler.
- It must reside in an authorized program library, either the library containing DB2 modules (*prefix*.SDSNLOAD) or in a library concatenated ahead of *prefix*.SDSNLOAD in the procedure for the database services started task (the procedure named *ssnm*DBM1, where *ssnm* is the DB2 subsystem name). Authorization routines must be accessible to the *ssnm*MSTR procedure. For all routines, we recommend using the library *prefix*.SDSNEXIT, which is concatenated ahead of *prefix*.SDSNLOAD in both started-task procedures.
- Routines that are listed in Table 271 *must* have the names shown. The name of other routines should not start with “DSN”, to avoid conflict with the DB2 modules.

Table 271. Required load module name

Type of routine	Required load module name
Date	DSNXVDTX
Time	DSNXVTMX
Connection	DSN3@ATH
Sign-on	DSN3@SGN

- It must be written to be reentrant and must restore registers before return.
- It must be link-edited with the REENTRANT parameter.
- In the MVS/ESA™ environment, it must be written and link-edited to execute AMODE(31),RMODE(ANY).
- It must not invoke any DB2 services—for example, through SQL statements.
- It must not invoke any SVC services or ESTAE routines.

Even though DB2 has functional recovery routines of its own, you can establish your own functional recovery routine (FRR), specifying MODE=FULLXM and EUT=YES.

Modifying exit routines

Because exit routines operate as extensions of DB2, they should not be changed or modified while DB2 is running.

Execution environment for exit routines

Exit routines are invoked by standard CALL statements. With some exceptions, which are noted under "General Considerations" in the description of particular types of routine, the execution environment is:

- Supervisor state
- Enabled for interrupts
- PSW key 7
- No MVS locks held
- For local requests, under the TCB of the application program that requested the DB2 connection
- For remote requests, under a TCB within the DB2 distributed data facility address space
- 31-bit addressing mode
- Cross-memory mode

  In cross-memory mode, the current primary address space is not equal to the home address space. Therefore, some z/OS macro services you cannot use at all, and some you can use only with restrictions. For more information about cross-memory restrictions for macro instructions, which macros can be used fully, and the complete description of each macro, refer to the appropriate z/OS publication.

Registers at invocation for exit routines

When DB2 passes control to an exit routine, the registers are set as listed in Table 272:

Table 272. Contents of registers when DB2 passes control to an exit routine

Register	Contains
1	Address of pointer to the exit parameter list (shown in Table 273 on page 1106). *For a field procedure*, the address is that of the field procedure parameter list (see Figure 149 on page 1093).
13	Address of the register save area.
14	Return address.
15	Address of entry point of exit routine.

Parameter lists for exit routines

Register 1 points to the address of parameter list EXPL, described by macro DSNDEXPL and shown in Figure 150. The word following points to a second parameter list, which differs for each type of exit routine.

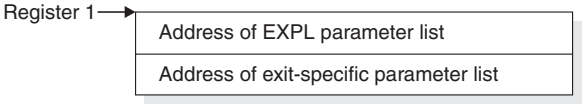


Figure 150. Use of register 1 on invoking an exit routine. (Field procedures and translate procedures do not use the standard exit-specific parameter list.)

Table 273 shows the EXPL parameter list. Its description is given by macro DSNDEXPL.

Table 273. Contents of EXPL parameter list

Name	Hex offset	Data type	Description
EXPLWA	0	Address	Address of a work area to be used by the routine
EXPLWL	4	Signed 4-byte integer	Length of the work area. The value is: 2048 for connection routines and sign-on routines 512 for date and time routines and translate procedures (see Note 1). 256 for edit, validation, and log capture routines
EXPLRSV1	8	Signed 2-byte integer	Reserved
EXPLRC1	A	Signed 2-byte integer	Return code
EXPLRC2	C	Signed 4-byte integer	Reason code
EXPLARC	10	Signed 4-byte integer	Used only by connection routines and sign-on routines
EXPLSSNM	14	Character, 8 bytes	Used only by connection routines and sign-on routines
EXPLCONN	1C	Character, 8 bytes	Used only by connection routines and sign-on routines
EXPLTYPE	24	Character, 8 bytes	Used only by connection routines and sign-on routines

Notes:
1. When translating a string of type PC MIXED, a translation procedure has a work area of 256 bytes plus the length attribute of the string.

Row formats for edit and validation routines

In writing an edit or validation routine, you must be aware of the format in which DB2 stores the rows of tables. This section describes the special features of that format.

Column boundaries for edit and validation routines

DB2 stores columns contiguously, regardless of word boundaries in physical storage.

LOB columns are an exception. LOB values are not stored contiguously. An indicator column is stored in a base table in place of the LOB value.

Edit procedures cannot be specified for any table that contains a LOB column or a ROWID column. In addition, LOB values are not available to validation routines; indicator columns and ROWID columns represent LOB columns as input to a validation procedure.

Null values for edit procedures, field procedures, and validation routines

If null values are allowed for a column, an extra byte is stored before the actual column value. This byte is X'00' if the column value is **not** null; it is X'FF' if the value is null.

The extra byte is included in the column length attribute (parameter FFMTFLEN in Table 281 on page 1109).

Fixed-length rows for edit and validation routines

If all columns in a table are fixed-length, its rows are stored in fixed-length format. The rows are byte strings.

Example: The sample project activity table has five fixed-length columns. The first two columns do not allow nulls; the last three do. Table 274 shows a row in the table.

Table 274. A row in fixed-length format

Column 1	Column 2	Column 3		Column 4		Column 5	
MA2100	10	00	0.5	00	820101	00	821101

Varying-length rows for edit and validation routines

If a table has any varying-length columns, its rows contain varying-length values, and are varying-length rows. Each varying-length value has a 2-byte length field in front of it. Those 2 bytes are **not** included in the column length attribute (FFMTFLEN).

Table 275 shows a row of the sample department table.

Table 275. A varying-length row in the sample department table.. The first value in Column 2 indicates the column length as a hexadecimal value.

Column 1	Column 2		Column 3	Column 4
C01	0012	Information center	000030	A00

There are no gaps after varying-length columns. Hence, columns that appear after varying-length columns are at variable offsets in the row. To get to such a column, you must scan the columns sequentially after the first varying-length column. An empty string has a length of zero with no data following.

ROWID and indicator columns are treated like varying length columns. Row IDs are VARCHAR(17). An indicator columns is VARCHAR(4); it is stored in a base table in place of a LOB column, and indicates whether the LOB value for the column is null or zero length.

Varying-length rows with nulls for edit and validation routines

A varying-length column can also allow null values. The value in the length field includes the null indicator byte but does not include the length field itself.

Table 276 shows how the row in Table 275 on page 1107 would look in storage if nulls were allowed in Column 2.

Table 276. A varying-length row in the sample department table.. The first value in Column 2 indicates the column length as a hexadecimal value.

Column 1	Column 2		Column 3	Column 4
C01	0013	Information center	000030	A00

An empty string has a length of one, a X'00' null indicator, and no data following.

Dates, times, and timestamps for edit and validation routines

The values in columns with data types of DATE, TIME, and TIMESTAMP are stored in the formats that are shown in the following tables. For each format, each byte consists of two packed decimal digits.

Table 277 shows the DATE format, which consists of 4 total bytes

Table 277. DATE format

Year	Month	Day
2 bytes	1 byte	1 byte

Table 278 shows the TIME format, which consists of 3 total bytes.

Table 278. TIME format

Hours	Minutes	Seconds
1 byte	1 byte	1 byte

Table 279 shows the TIMESTAMP format, which consists of 10 total bytes.

Table 279. TIMESTAMP format

Year	Month	Day	Hours	Minutes	Seconds	Microseconds
2 bytes	1 byte	1 byte	1 byte	1 byte	1 byte	3 bytes

Parameter list for row format descriptions

DB2 passes a description of the row format to an edit or validation routine through a parameter list, generated by macro DSNDROW. The description includes both the general row characteristics and the characteristics of each column. Table 280 on page 1109 shows the general row description.

Table 280. Description of a row format

Name	Hex offset	Data type	Description
RFMTNFLD	0	Signed fullword integer	Number of columns in a row
RFMTAFLD	4	Address	Address of a list of column descriptions. The format of each column is shown in Table 281.
RFMTTYPE	8	Character, 1 byte	Row type: X'00' = row with fixed-length columns X'04' = row with varying-length columns
	9	Character, 3 bytes	Reserved

Table 281 shows the description of each column.

Table 281. Description of a column format

Name	Hex offset	Data type	Description
FFMTFLEN	0	Signed fullword integer	Column length attribute (see Table 282)
FFMTFTYP	4	Character, 1 byte	Data type code (see Table 282)
FFMTNULL	5	Character, 1 byte	Data attribute: X'00' = Null values are allowed. X'04' = Null values are not allowed.
FFMTFNAM	6	Character, 18 bytes	Column name

Table 282 shows a description of data type codes and length attributes.

Table 282. Description of data type codes and length attributes

Data type	Code (FFMTFTYP)	Length attribute (FFMTFLEN)
INTEGER	X'00'	4
SMALLINT	X'04'	2
FLOAT (single precision)	X'08'	4
FLOAT (double precision)	X'08'	8
DECIMAL	X'0C'	INTEGER($p/2$), where p is the precision
CHAR	X'10'	The length of the string
VARCHAR	X'14'	The length of the string
DATE	X'20'	4
TIME	X'24'	3
TIMESTAMP	X'28'	10
ROWID	X'2C'	17
INDICATOR COLUMN	X'30'	4

DB2 codes for numeric data in edit and validation routines

DB2 stores numeric data in a specially encoded format. That format is called *DB2-coded*. To retrieve numeric data in its original form, you must DB2-decode it, according to its data type, as is listed in Table 283:

Table 283. DB2 decoding procedure according to data type

Data type	DB2 decoding procedure
SMALLINT	Invert the sign bit (high order bit). **Value** **Meaning** **8001** 0001 (+1 decimal) **7FF3** FFF3 (-13 decimal)
INTEGER	Invert the sign bit (high order bit). **Value** **Meaning** **800001F2** 000001F2 (+498 decimal) **7FFFFF85** FFFFFF85 (-123 decimal)
FLOAT	If the sign bit (high order bit) is 1, invert only that bit. Otherwise, invert all bits. **Value** **Meaning** **C110000000000000** 4110000000000000 (+1.0 decimal) **3EEFFFFFFFFFFFFF** C110000000000000 (-1.0 decimal)
DECIMAL	Save the high-order hexadecimal digit (sign digit). Shift the number to the left one hexadecimal digit. If the sign digit is X'F', put X'C' in the low-order position. Otherwise, invert all bits in the number and put X'D' in the low-order position. **Value** **Meaning** **F001** 001C (+1) **0FFE** 001D (-1)

RACF access control module

The RACF access control module allows you to use RACF as an alternative to DB2 authorization checking for DB2 objects, authorities, and utilities. You can activate the RACF access control module at the DB2 access control authorization exit point (DSNX@XAC), where you can replace the default routine. The RACF access control module is provided as an assembler source module in the DSNXRXAC member of DB2.SDSNSAMP.

For more information about the RACF access control module, see *DB2 RACF Access Control Module Guide*.

Appendix C. Reading log records

The information in this appendix is Product-sensitive Programming Interface and Associated Guidance Information as defined in "Notices" on page 1301.

This appendix discusses the following information about the log:
"Contents of the log"
"The physical structure of the log" on page 1116

For diagnostic or recovery purposes, it can be useful to read DB2 log records. This appendix also discusses three approaches to writing programs that read log records:

- "Reading log records with IFI" on page 1122

  This is an online method using the instrumentation facility interface (IFI) when DB2 is running. You use the READA (read asynchronously) command of IFI to read log records into a buffer and the READS (read synchronously) command to pick up specific log control intervals from a buffer.
- "Reading log records with OPEN, GET, and CLOSE" on page 1126

  This is a stand-alone method that can be used when DB2 is down. You use the assembler language macro DSNJSLR to submit OPEN, GET, and CLOSE functions. This method can be used to capture log records that you cannot pick up with IFI after DB2 goes down.
- "Reading log records with the log capture exit routine" on page 1134

  This is an online method using the log capture exit when DB2 is running. You write an exit routine to use this exit to capture and transfer log records in real time.

Contents of the log

The log contains the information needed to recover the results of program execution, the contents of the database, and the DB2 subsystem. It does not contain information for accounting, statistics, traces, or performance evaluation.

There are three main types of log records which are described under these headings:
"Unit of recovery log records" on page 1112
"Checkpoint log records" on page 1115
"Database page set control records" on page 1116

Exception information that is not included in any of these types is described under "Other exception information" on page 1116.

Each log record has a header that indicates its type, the DB2 subcomponent that made the record, and, for unit-of-recovery records, the unit-of-recovery identifier. The log records can be extracted and printed by the DSN1LOGP program. For instructions, refer to Part 3 of *DB2 Utility Guide and Reference*.

The log relative byte address and log record sequence number: The DB2 log can contain up to 2^{48} bytes, where 2^{48} is 2 to the 48th power. Each byte is addressable by its offset from the beginning of the log. That offset is known as its *relative byte address* (RBA).

© Copyright IBM Corp. 1982, 2008

A log record is identifiable by the RBA of the first byte of its header; that RBA is called the *relative byte address of the record*. The record RBA is like a timestamp because it uniquely identifies a record that starts at a particular point in the continuing log.

In the data sharing environment, each member has its own log. The *log record sequence number (LRSN)* uniquely identifies the log records of a data sharing member. The LRSN is a 6-byte hexadecimal value derived from a store clock timestamp. DB2 uses the LRSN for recovery in the data sharing environment.

Effects of ESA data compression: Log records can contain compressed data if a table contains compressed data. For example, if the data in a DB2 row are compressed, all data logged because of changes to that row (resulting from inserts, updates and deletes) are compressed. If logged, the record prefix is not compressed, but all of the data in the record are in compressed format. Reading compressed data requires access to the dictionary that was in use when the data was compressed.

Unit of recovery log records

Most of the log records describe changes to the DB2 database. All such changes are made within units of recovery. This section describes changes to the DB2 database, the effects of these changes, and the log records that correspond to the changes.

Undo and redo records

When a change is made to the database, DB2 logs an *undo/redo* record that describes the change. The *redo* information is required if the work is committed and later must be recovered. The *undo* information is used to back out work that is not committed.

If the work is rolled back, the undo/redo record is used to remove the change. At the same time that the change is removed, a new redo/undo record is created that contains information, called *compensation information*, that is used if necessary to reverse the change. For example, if a value of 3 is changed to 5, redo compensation information changes it back to 3.

If the work must be recovered, DB2 scans the log forward and applies the redo portions of log records and the redo portions of compensation records, without keeping track of whether the unit of recovery was committed or rolled back. If the unit of recovery had been rolled back, DB2 would have written compensation redo log records to record the original undo action as a redo action. Using this technique, the data can be completely restored by applying only redo log records on a single forward pass of the log.

DB2 also logs the creation and deletion of data sets. If the work is rolled back, the operations are reversed. For example, if a table space is created using DB2-managed data sets, DB2 creates a data set; if rollback is necessary, the data set is deleted. If a table space using DB2-managed data sets is dropped, DB2 deletes the data set when the work is committed, not immediately. If the work is rolled back, DB2 does nothing.

Database exception table records

Database exception table (DBET) log records register several types of information: exception states and image copies of special table spaces. DBET log records also register exception information that is not related to units of recovery (see “Other exception information” on page 1116 for more information.

Exception states: DBET log records register whether any database, table space, index space, or partition is in an exception state. To list all objects in a database that are in an exception state, use the command DISPLAY DATABASE (*database name*) RESTRICT. For a further explanation of the list produced and of the exception states, see the description of message DSNT392I in Part 2 of *DB2 Messages*.

Image copies of special table spaces: Image copies of DSNDB01.SYSUTILX, DSNDB01.DBD01, and DSNDB06.SYSCOPY are registered in the DBET log record rather than in SYSCOPY. During recovery, they are recovered from the log, and then image copies of other table spaces are located from the recovered SYSCOPY.

Typical unit of recovery log records

Table 284 shows a sequence of log records that might be written for an insert of one row via TSO. The following record types are included:

Begin_UR
The first request to change a database begins a unit of recovery. The log record of that event is identified by its log RBA. That same RBA serves as an ID for the entire unit of recovery (the URID). All records related to that unit have that RBA in their log record headers (LRH). For rapid backout, the records are also linked by a backward chain in the LRH.

Undo/Redo
Log records are written for each insertion, deletion, or update of a row. They register the changes to the stored data, but not the SQL statement that caused the change. Each record identifies one data or index page and its changes.

End Phase 2 records
The end of a UR is marked by log records that tell whether the UR was committed or rolled back, and whether DB2 has completed the work associated with it. If DB2 terminates before a UR has completed, it completes the work at the next restart.

Table 284. Example of a log record sequence for an INSERT of one row using TSO

Type of record	Information recorded
1. Begin_UR	Beginning of the unit of recovery. Includes the connection name, correlation name, authorization ID, plan name, and LUWID.
2. Undo/Redo for data	Insertion of data. Includes the database ID (DBID), page set ID, page number, internal record identifier (RID), and the data inserted.
3. Undo/Redo for Index	Insertion of index entry. Includes the DBID, index space object ID, page number, and index entry to be added.
4. Begin Commit 1	The beginning of the commit process. The application has requested a commit either explicitly (EXEC SQL COMMIT) or implicitly (for example, by ending the program).
5. Phase 1-2 Transition	The agreement to commit in TSO. In CICS and IMS, an End Phase 1 record notes that DB2 agrees to commit. If both parties agree, a Begin Phase 2 record is written; otherwise, a Begin Abort record is written, noting that the unit of recovery is to be rolled back.
6. End Phase 2	Completion of all work required for commit.

Table 285 shows the log records for processing and rolling back an insertion.

Table 285. Log records written for rolling back an insertion

Type of record	Information recorded
1. Begin_UR	Beginning of the unit of recovery.
2. Undo/Redo for data	Insertion of data. Includes the database ID (DBID), page set ID, page number, internal record identifier, and the data inserted.
3. Begin_Abort	Beginning of the rollback process.
4. Compensation Redo/Undo	Backing-out of data. Includes the database ID (DBID), page set ID, page number, internal record ID (RID), and data to undo the previous change.
5. End_Abort	End of the unit of recovery, with rollback complete.

Classes of changes to data

Table 286 summarizes the information logged for data and index changes.

Table 286. Information logged for database changes

Operation	Information logged
Insert data	The new row • On redo, the row is inserted with its original RID. • On undo, the row is deleted and the RID is made available for another row.
Delete data	The deleted row • On redo, the RID is made available for another row. • On undo, the row is inserted again with its former RID.
Update data	The old and new values of the changed data. • On redo, the new data is replaced. • On undo, the old data is replaced.
Note: If an update occurs to a table defined with DATA CAPTURE(CHANGES), the entire before-image and after-image of the data row is logged.	
Insert index entry	The new key value and the data RID.
Delete index entry	The deleted key value and the data RID.

There are three basic classes of changes to a data page:

- Changes to control information. Those changes include pages that map available space and indicators that show that a page has been modified. The COPY utility uses that information when making incremental image copies.
- Changes to database pointers. Pointers are used in two situations:
  - The DB2 catalog and directory, but not user databases, contain pointers that connect related rows. Insertion or deletion of a row changes pointers in related data rows.
  - When a row in a user database becomes too long to fit in the available space, it is moved to a new page. An address, called an *overflow pointer*, that points to the new location is left in the original page. With this technique, index entries and other pointers do not have to be changed. Accessing the row in its original position gives a pointer to the new location.

- Changes to data. In DB2, a row is confined to a single page. Each row is uniquely identified by a RID containing:
  - The number of the page
  - A 1-byte ID that identifies the row within the page. A single page can contain up to 255 rows; [9] IDs are reused when rows are deleted.

The log record identifies the RID, the operation (insert, delete, or update), and the data. Depending on the data size and other variables, DB2 can write a single log record with both undo and redo information, or it can write separate log records for undo and redo.

Checkpoint log records

To reduce restart time, DB2 takes periodic checkpoints during normal operation, in the following circumstances:

- When a predefined number of log records have been written or a predetermined amount of time in minutes has elapsed.

  This number is defined by field CHECKPOINT FREQ on installation panel DSNTIPL described in Part 2 of *DB2 Installation Guide*.
- When switching from one active log data set to another
- At the end of a successful restart
- At normal termination

At a checkpoint, DB2 logs its current status and registers the log RBA of the checkpoint in the bootstrap data set (BSDS). At restart, DB2 uses the information in the checkpoint records to reconstruct its state when it terminated.

Many log records can be written for a single checkpoint. DB2 can write one to begin the checkpoint; others can then be written, followed by a record to end the checkpoint. Table 287 summarizes the information logged.

Table 287. Contents of checkpoint log records

Type of log record	Information logged
Begin_Checkpoint	Marks the start of the summary information. All later records in the checkpoint have type X'0100' (in the LRH).
Unit of Recovery Summary	Identifies an incomplete unit of recovery (by the log RBA of the Begin_UR log record). Includes the date and time of its creation, its connection ID, correlation ID, authorization ID, the plan name it used, and its current state (inflight, indoubt, in-commit, or in-abort).
Page Set Summary	Contains information for allocating and opening objects at restart, and identifies (by the log RBA) the earliest checkpoint interval containing log records about data changes that have not been applied to the DASD version of the data or index. There is one record for each open page set (table space or index space).
Page Set Exception Summary	Identifies the type of exception state. For descriptions of the states, see “Database page set control records” on page 1116. There is one record for each database and page set with an exception state.
Page Set UR Summary Record	Identifies page sets modified by any active UR (inflight, in-abort, or in-commit) at the time of the checkpoint.

9. A page in a catalog table space that has links can contain up to 127 rows.

Table 287. Contents of checkpoint log records (continued)

Type of log record	Information logged
End_Checkpoint	Marks the end of the summary information about a checkpoint.

Database page set control records

Page set control records primarily register the allocation, opening, and closing of every page set (table space or index space). That same information is in the DB2 directory (SYSIBM.SYSLGRNX). It is also registered in the log so that it is available at restart.

Other exception information

Entries for data pages that are logically in error (logical page list or LPL entries) or physically in error (write error page range or WEPR entries) are registered in the DBET log record.

The physical structure of the log

Each active log data set must be a VSAM linear data set (LDS). The physical output unit written to the active log data set is a control interval (CI) of 4096 bytes (4 KB). Each CI contains one VSAM record.

Physical and logical log records

The VSAM CI provides 4089 bytes to hold DB2 information. That space is called a *physical* record. The information to be logged at a particular time forms a *logical* record, whose length varies independently of the space available in the CI. Hence, one physical record can contain several logical records, one or more logical records and part of another, or only part of one logical record. The physical record must also contain 21 bytes of DB2 control information, called the *log control interval definition* (LCID), which is further described in "The log control interval definition (LCID)" on page 1118.

Figure 151 on page 1117 shows a VSAM CI containing four log records or segments, namely:

- The last segment of a log record of 768 bytes (X'0300'). The length of the segment is 100 bytes (X'0064').
- A complete log record of 40 bytes (X'0028').
- A complete log record of 1024 bytes (X'0400').
- The first segment of a log record of 4108 bytes (X'100C'). The length of the segment is 2911 bytes (X'0B5F').

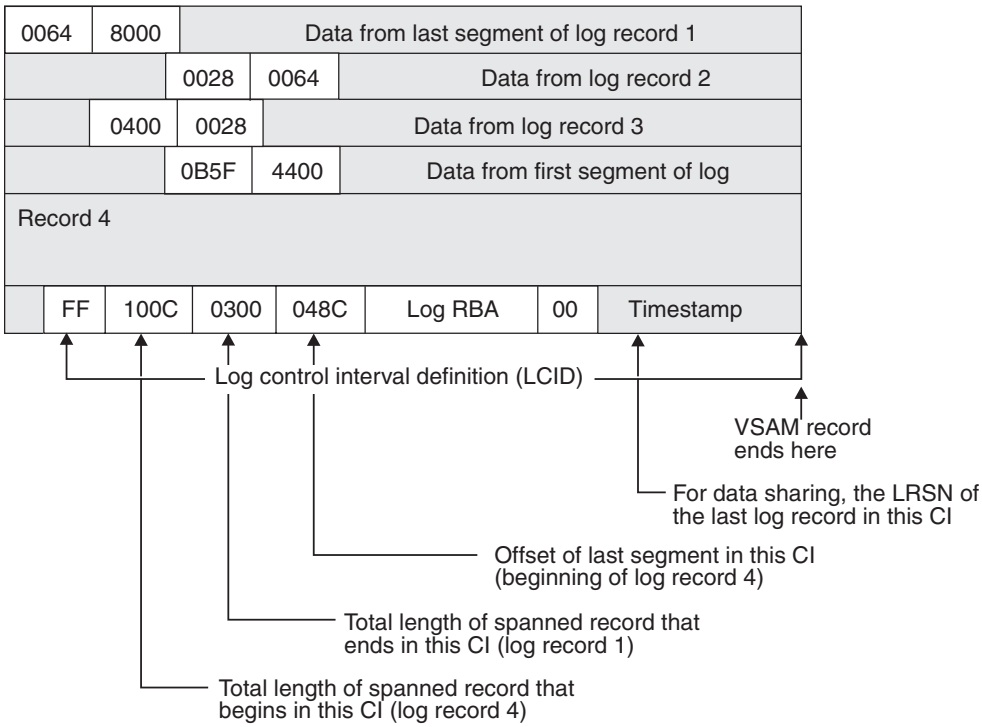


Figure 151. A VSAM CI and its contents

The term *log record* refers to a logical record, unless the term *physical log record* is used. A part of a logical record that falls within one physical record is called a *segment*.

The log record header

Each logical record includes a prefix, called a *log record header* (LRH), which contains control information. For the contents of the log record header see Table 288.

The first segment of a log record must contain the header and some bytes of data. If the current physical record has too little room for the minimum segment of a new record, the remainder of the physical record is unused, and a new log record is written in a new physical record.

The log record can span many VSAM CIs. For example, a minimum of nine CIs are required to hold the maximum size log record of 32815 bytes. Only the first segment of the record contains the entire LRH; later segments include only the first two fields. When a specific log record is needed for recovery, all segments are retrieved and presented together as if the record were stored continuously.

Table 288. Contents of the log record header

Hex offset	Length	Information
00	2	Length of this record or segment

Table 288. Contents of the log record header (continued)

Hex offset	Length	Information
02	2	Length of any previous record or segment in this CI; 0 if this is the first entry in the CI. The two high-order bits tell the segment type: B'00' A complete log record B'01' The first segment B'11' A middle segment B'10' The last segment
04	2	Type of log record [1]
06	2	Subtype of the log record [1]
08	1	Resource manager ID (RMID) of the DB2 component that created the log record
09	1	Flags
0A	6	Unit of recovery ID, if this record relates to a unit of recovery[2]; otherwise, 0
10	6	Log RBA of the previous log record, if this record relates to a unit of recovery[2]; otherwise, 0
16	1	Release identifier
17	1	Length of header
18	6	Undo next LSN
1E	8	LRHTIME

Notes:

1. For record types and subtypes, see "Log record type codes" on page 1120 and "Log record subtype codes" on page 1120.
2. For a description of units of recovery, see "Unit of recovery log records" on page 1112

The log control interval definition (LCID)

Each physical log record includes a suffix called the *log control interval definition* (LCID), which tells how record segments are placed in the physical control interval. For the contents of the LCID, see Table 289.

Table 289. Contents of the log control interval definition

Hex offset	Length	Information
00	1	An indication of whether the CI contains free space: X'00' = Yes, X'FF' = No
01	2	Total length of a segmented record that begins in this CI; 0 if no segmented record begins in this CI
03	2	Total length of a segmented record that ends in this CI; 0 if no segmented record ends in this CI
05	2	Offset of the last record or segment in the CI
07	6	Log RBA of the start of the CI
0D	6	Timestamp, reflecting the date and time that the log buffer was written to the active log data set. The timestamp is the high-order 7 bytes of the store clock value (STCK).
13	2	Reserved

Each recovery log record consists of two parts: a *header*, which describes the record, and data. Figure 152 shows the format schematically; the following list describes each field.

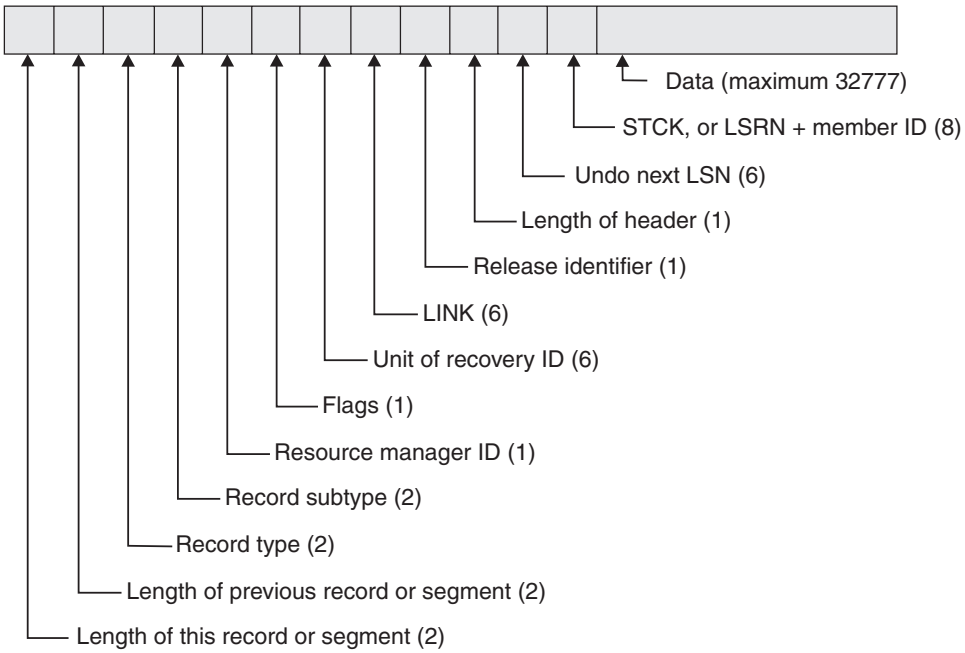


Figure 152. Format of a DB2 recovery log record

The fields are:

Length of this record
The total length of the record in bytes.

Length of previous record
The total length of the previous record in bytes.

Type The code for the type of recovery log record. See "Log record type codes" on page 1120.

Subtype
Some types of recovery log records are further divided into subtypes. See "Log record subtype codes" on page 1120.

Resource manager ID
Identifier of the resource manager that wrote the record into the log. When the log is read, the record can be given for processing to the resource manager that created it.

Unit of recovery ID
A unit of recovery to which the record is related. Other log records can be related to the same unit of recovery; all of them must be examined to recover the data. The URID is the RBA (relative byte address) of the Begin-UR log record, and indicates the start of that unit of recovery in the log.

LINK Chains all records written using their RBAs. For example, the link in an end checkpoint record links the chains back to the begin checkpoint record.

Release identifier
Identifies in which release the log was written.

Log record header length
The total length of the header of the log record.

Undo next LSN
Identifies the log RBA of the next log record to be undone during backwards (UNDO processing) recovery.

STCK, or LRSN+member ID
In a non-data-sharing environment, this is a 6-byte store clock value (STCK) reflecting the date and time the record was placed in the output buffer. The last 2 bytes contain zeros.

In a data sharing environment, this contains a 6-byte log record sequence number (LRSN) followed by a 2-byte member ID.

Data Data associated with the log record. The contents of the data field depend on the type and subtype of the recovery log record.

Log record type codes

The type code of a log record tells what kind of DB2 event the record describes:

Code	Type of event
0002	Page set control
0004	SYSCOPY utility
0010	System event
0020	Unit of recovery control
0100	Checkpoint
0200	Unit of recovery undo
0400	Unit of recovery redo
0800	Archive log command
1000 to 8000	Assigned by DB2
2200	Savepoint

A single record can contain multiple type codes that are combined. For example, 0600 is a combined UNDO/REDO record; F400 is a combination of four DB2-assigned types plus a REDO.

Log record subtype codes

The log record subtype code provides a more granular definition of the event that occurred to produce the log record. Log record subtype codes are unique only within the scope of the corresponding log record type code.

Log record type 0004 (SYSCOPY utility) has log subtype codes that correspond to the page set ID values of the table spaces that have their SYSCOPY records in the log (SYSIBM.SYSUTILX, SYSIBM.SYSCOPY and DSNDB01.DBD01).

For a description of log record types 0200 (unit of recovery undo) and 0400 (unit of recovery redo), see the SUBTYPE option of DSN1LOGP in Part 3 of *DB2 Utility Guide and Reference*.

Log record type 0800 (quiesce) does not have subtype codes.

Some log record types (1000 to 8000 assigned by DB2) can have proprietary log record subtype codes assigned.

Subtypes for type 0002 (page set control):

Code	Type of event

0001 Page set open
0002 Data set open
0003 Page set close
0004 Data set close
0005 Page set control checkpoint
0006 Page set write
0007 Page set write I/O
0008 Page set reset write
0009 Page set status

Subtypes for type 0010 (system event):

Code	Type of event
0001	Begin checkpoint
0002	End checkpoint
0003	Begin current status rebuild
0004	Begin historic status rebuild
0005	Begin active unit of recovery backout
0006	Pacing record

Subtypes for type 0020 (unit of recovery control):

Code	Type of event
0001	Begin unit of recovery
0002	Begin commit phase 1 (Prepare)
0004	End commit phase 1 (Prepare)
0008	Begin commit phase 2
000C	Commit phase 1 to commit phase 2 transition
0010	End commit phase 2
0020	Begin abort
0040	End abort
0081	End undo
0084	End todo
0088	End redo

Subtypes for type 0100 (checkpoint):

Code	Type of event
0001	Unit of recovery entry
0002	Restart unit of recovery entry

Subtypes for type 2200 (savepoint):

Code	Type of event
0014	Rollback to savepoint
000E	Release to savepoint

Interpreting data change log records

DB2 provides the mapping and description of specific log record types that allow you to interpret data changes made to DB2 tables from the log. The macros are contained in the data set library *prefix*.SDSNMACS and are documented by comments in the macros themselves.

Log record formats for the record types and subtypes that are listed in "Log record subtype codes" on page 1120 are detailed in the mapping macro DSNDQJ00. DSNDQJ00 provides the mapping of specific data change log records, UR control

log records, and page set control log records that you need to interpret data changes by the UR. DSNDQJ00 also explains the content and usage of the log records.

Reading log records with IFI

You can write a program that uses IFI to capture log records while DB2 is running. You can read the records asynchronously, by starting a trace that reads the log records into a buffer and then issuing an IFI call to read those records out of the buffer. Alternatively, you can read those log records synchronously, by using an IFI call that returns those log records directly to your IFI program.

This section describes both methods, in the following topics:
"Reading log records into a buffer"
"Reading specific log records (IFCID 0129)"
"Reading complete log data (IFCID 0306)" on page 1123

Either the primary or one of the secondary authorization IDs must have MONITOR2 privilege. For details on how to code an IFI program, see Appendix E, "Programming for the Instrumentation Facility Interface (IFI)," on page 1151.

Reading log records into a buffer

To begin gathering active log records into a buffer, issue the following command in your IFI program:

```
-START TRACE(P) CLASS(30) IFCID(126) DEST(OPX)
```

where:

- P signifies to start a DB2 performance trace. Any of the DB2 trace types can be used.
- CLASS(30) is a user-defined trace class (31 and 32 are also user-defined classes).
- IFCID(126) activates DB2 log buffer recording.
- DEST(OPX) starts the trace to the next available DB2 online performance (OP) buffer. The size of this OP buffer can be explicitly controlled by the BUFSIZE keyword of the START TRACE command. Valid sizes range from 256 KB to 16 MB. The number must be evenly divisible by 4.

When the START TRACE command takes effect, from that point forward until DB2 terminates, DB2 begins writing 4-KB log buffer VSAM control intervals (CIs) to the OP buffer as well as to the active log. As part of the IFI COMMAND invocation, the application specifies an ECB to be posted and a threshold to which the OP buffer is filled when the application is posted to obtain the contents of the buffer. The IFI READA request is issued to obtain OP buffer contents.

Reading specific log records (IFCID 0129)

IFCID 129 can be used with an IFI READS request to return a specific range of log records from the active log into the return area your program has initialized. Enter the following command into your IFI program:

```
CALL DSNWLI(READS,ifca,return_area,ifcid_area,qual_area)
```

IFCID 129 must appear in the IFCID area.

To retrieve the log control interval, your program must initialize certain fields in the qualification area:

WQALLTYP
This is a 3-byte field in which you must specify CI (with a trailing blank), which stands for “control interval”.

WQALLMOD
In this 1-byte field, you specify whether you want the first log CI of the restarted DB2 subsystem, or whether you want a specific control interval as specified by the value in the RBA field.

F The “first” option is used to retrieve the first log CI of this DB2 instance. This option ignores any value in WQALLRBA and WQALLNUM.

P The“partial” option is used to retrieve partial log CIs for the log capture exit which is described in Appendix B, “Writing exit routines,” on page 1051. DB2 places a value in field IFCAHLRS of the IFI communication area, as follows:

- The RBA of the log CI given to the log capture exit routine, if the last CI written to the log was not full.
- 0, if the last CI written to the log was full.

When you specify option P, DB2 ignores values in WQALLRBA and WQALLNUM.

R The “read” option is used to retrieve a set of up to 7 continuous log CIs. If you choose this option, you must also specify the WQALLRBA and WQALLNUM options, which the following text details.

WQALLRBA
In this 8-byte field, you specify the starting log RBA of the control intervals to be returned. This value must end in X'000' to put the address on a valid boundary. This field is ignored when using the WQALLMOD=F option.

If you specify an RBA that is not in the active log, reason code 00E60854 is returned in the field IFCARC2, and the RBA of the first CI of the active log is returned in field IFCAFCI of the IFCA. These 6 bytes contain the IFCAFCI field.

WQALLNUM
In this 2-byte field, specify the number of control intervals you want returned. The valid range is from X'0001' through X'0007', which means that you can request and receive up to seven 4-KB log control intervals. This field is ignored when using the WQALLMOD=F option. For a complete description of the qualification area, see Table 302 on page 1159.

If you specify a range of log CIs, but some of those records have not yet been written to the active log, DB2 returns as many log records as possible. You can find the number of CIs returned in field QWT02R1N of the self-defining section of the record. For information about interpreting trace output, see Appendix D, “Interpreting DB2 trace output,” on page 1135.

Reading complete log data (IFCID 0306)

The major benefits for using IFCID 0306 are:

- IFCID 0306 can request DB2 to decompress log records if compressed, before passing them to the return area of your IFI program.
- In a data sharing environment, DB2 merges log records if the value of the IFI READS qualification WQALFLTR is X'00'. If WQALFLTR is X'01', log records are not merged.
- IFCID can retrieve log records from the archive data sets.

- Complete log records are always returned.

To use this IFCID, use the same call as described in "Reading specific log records (IFCID 0129)" on page 1122. IFCID 0306 must appear in the IFCID area. IFCID 0306 returns complete log records and the spanned record indicators in bytes 2 will have no meaning, if present. Multi-segmented control interval log records are combined for a complete log record.

Specifying the return area

For IFCID 0306 requests, your program's return area must reside in ECSA key 7 storage. The IFI application program must set the eye-catcher to 'I306' at offset 4 in the Return Area before making the IFCID 0306 call. There is an additional 60 byte area that must be included after the 4-byte length indicator and the 'I306' eye-catcher. This area is used by DB2 between successive application calls and must not be modified by the application. The return area mapping is documented later in this section.

The IFI application program needs to run in supervisor state to request the ECSA key 7 return area. The return area storage size need be a minimum of the largest DB2 log record returned plus the additional area defined in DSNDQW04. Minimize the number of IFI calls required to get all log data but do not over use ECSA by the IFI program. The other IFI storage areas can remain in user storage key 8. The IFI application must be in supervisor state and key 0 when making IFCID 0306 calls.

Qualifying log records

To retrieve IFCID 0306 log records, your program must initialize certain fields in the qualification area mapped by DSNDWQAL. These qualification fields are described here:

WQALLMOD

In this 1-byte field, specify one of the following modes:

D Retrieves the single log record whose RBA value and member id is specified in WQALLRBA. Issuing a D request while holding a position in the log, causes the request to fail and terminates the log position held.

F Used as a first call to request log records beyond the LRSN or RBA specified in WQALLRBA that meet the criteria specified in WQALLCRI.

H Retrieves the highest LRSN or log RBA in the active log. The value is returned in field IFCAHLRS of the IFI communications area (IFCA). There is no data returned in the return area and the return code for this call will indicate that no data was returned.

N Used following mode F or N calls to request any remaining log records that meet the criteria specified in WQALLCRI. * and any option specified in WQALLOPT. As many log records as fit in the program's return area are returned.

T Terminates the log position that was held by any previous F or N request. This allows held resources to be released.

Mode R is not used for IFCID 0306.

For both F or N requests, each log record returned contains a record-level feedback area recorded in QW0306L. The number of log records retrieved is in QW0306CT. The ending log RBA or LRSN of the log records to be returned is in QW0306ES.

WQALLRBA
In this 8-byte field, specify the starting log RBA or LRSN of the control records to be returned. For IFCID 0306, this is used on the "first" option (F) request to request log records beyond the LRSN or RBA specified in this field. Determine the RBA or LRSN value from the H request. For RBAs, the value plus one should be used. For IFCID 0306 with D request of WQALLMOD, the high-order 2 bytes must specify member id and the low order 6 bytes contain the RBA.

WQALLCRI
In this 1-byte field, indicate what types of log records you want:

X'00'
Tells DB2 to retrieve only log records for changed data capture and unit of recovery control.

X'FF'
Tells DB2 to retrieve all types of log records. Use of this option can retrieve large data volumes and degrade DB2 performance.

WQALLOPT
In this 1-byte field, indicate whether you want the returned log records to be decompressed.

X'01'
Tells DB2 to decompress the log records before they are returned.

X'00'
Tells DB2 to leave the log records in the compressed format.

A typical sequence of IFCID 0306 calls is:

WQALLMOD=H
This is only necessary if you want to find the current position in the log. The LRSN or RBA is returned in IFCAHLRS. The return area is not used.

WQALLMOD=F
The WQALLRBA, WQALLCRI and WQALLOPT should be set. If 00E60812 is returned, you have all the data for this scope. You should wait a while before issuing another WQALLMOD=F call. In data sharing, log buffers are flushed when the F request is issued.

WQALLMOD=N
If the 00E60812 has not been returned, you issue this call until it is. You should wait a while before issuing another WQALLMOD=F call.

WQALLMOD=T
This should only be used if you do not want to continue with the WQALLMOD=N before the end is reached. It has no use if a position is not held in the log.

IFCID 0306 return area mapping: IFCID 0306 has a unique return area format. The first section is mapped by QW0306OF instead of the write header DSNDQWIN. See Appendix E, "Programming for the Instrumentation Facility Interface (IFI)," on page 1151 for details.

Reading log records with OPEN, GET, and CLOSE

DB2 provides three stand-alone log services that user-written application programs can use to read DB2 recovery log records and control intervals even when DB2 is not running:

- OPEN initializes stand-alone log services.
- GET returns a pointer to the next log record or log record control interval.
- CLOSE deallocates data sets and frees storage.

To invoke these services, use the assembler language macro, DSNJSLR, specifying one of the preceding functions.

These log services use a *request block*, which contains a feedback area in which information for all stand-alone log GET calls is returned. The request block is created when a stand-alone log OPEN call is made. The request block must be passed as input to all subsequent stand-alone log calls (GET and CLOSE). The request block is mapped by the DSNDSLRB macro and the feedback area is mapped by the DSNDSLRF macro.

See Figure 153 on page 1133 for an example of an application program that includes these various stand-alone log calls.

When you issue an OPEN request, you can indicate whether you want to get log records or log record control intervals. Each GET request returns a single logical record or control interval depending on which you selected with the OPEN request. If neither is specified, the default, RECORD, is used. DB2 reads the log in the forward direction of ascending relative byte addresses or log record sequence numbers (LRSNs).

If a bootstrap data set (BSDS) is allocated before stand-alone services are invoked, appropriate log data sets are allocated dynamically by z/OS. If the bootstrap data set is not allocated before stand-alone services are invoked, the JCL for your user-written application to read a log must specify and allocate the log data sets to be read.

Table 290 lists and describes the JCL DD statements used by stand-alone services.

Table 290. JCL DD statements for DB2 stand-alone log services

JCL DD statement	Explanation
JOBCAT or STEPCAT	Specifies the catalog in which the BSDS and the active log data sets are cataloged. Required if the BSDS or any active log data set is to be accessed, unless the data sets are cataloged in the system master catalog.
BSDS	Specifies the bootstrap data set (BSDS). Optional. Another ddname can be used for allocating the BSDS, in which case the ddname must be specified as a parameter on the FUNC=OPEN (see "Stand-alone log OPEN request" on page 1129 for more information). Using the ddname in this way causes the BSDS to be used. If the ddname is omitted on the FUNC=OPEN request, the processing uses DDNAME=BSDS when attempting to open the BSDS.
ARCHIVE	Specifies the archive log data sets to be read. Required if an archive data set is to be read and the BSDS is not available (the BSDS DD statement is omitted). Should not be present if the BSDS DD statement is present. If multiple data sets are to be read, specify them as concatenated data sets in ascending log RBA order.

Table 290. JCL DD statements for DB2 stand-alone log services (continued)

JCL DD statement	Explanation
ACTIVE*n*	(Where *n* is a number from 1 to 7). Specifies an active log data set that is to be read. Should not be present if the BSDS DD statement is present. If only one data set is to be read, use ACTIVE1 as the ddname. If multiple active data sets are to be read, use DDNAMEs ACTIVE1, ACTIVE2, ... ACTIVE*n* to specify the data sets. Specify the data sets in ascending log RBA order with ACTIVE1 being the lowest RBA and ACTIVE*n* being the highest.
DD statements for data sharing users	
GROUP	If you are reading logs from every member of a data sharing group in LRSN sequence, you can use this statement to locate the BSDSs and log data sets needed. You must include the data set name of one BSDS in the statement. DB2 can find the rest of the information from that one BSDS. All members' logs and BSDS data sets must be available. If you use this DD statement, you must also use the LRSN and RANGE parameters on the OPEN request. The GROUP DD statement overrides any M*xx*BSDS statements that are used. (DB2 searches for the BSDS DD statement first, then the GROUP statement, and then the M*xx*BSDS statements. If you want to use a particular member's BSDS for your own processing, you must call that DD statement something other than BSDS.)
M*xx*BSDS	Names the BSDS data set of a member whose log must participate in the read operation and whose BSDS is to be used to locate its log data sets. Use a separate M*xx*BSDS DD statement for each DB2 member. *xx* can be any two valid characters. Use these statements if logs from selected members of the data sharing group are required and the BSDSs of those members are available. These statements are ignored if you use the GROUP DD statement. For one M*xx*BSDS statement, you can use either RBA or LRSN values to specify a range. If you use more than one M*xx*BSDS statement, you must use the LRSN to specify the range.
M*yy*ARCHV	Names the archive log data sets of a member to be used as input. *yy* can be any two valid characters that do not duplicate any *xx* used in an M*xx*BSDS DD statement. Concatenate all required archived log data sets of a given member in time sequence under one DD statement. Use a separate M*yy*ARCHV DD statement for each member. You must use this statement if the BSDS data set is unavailable or if you want only some of the log data sets from selected members of the group. If you name the BSDS of a member by a M*xx*BSDS DD statement, do not name the log of the same member by an M*yy*ARCHV statement. If both M*yy*ARCHV and M*xx*BSDS identify the same log data sets, the service request fails. M*yy*ARCHV statements are ignored if you use the GROUP DD statement.

Table 290. JCL DD statements for DB2 stand-alone log services (continued)

JCL DD statement	Explanation
M*yy*ACT*n*	Names the active log data set of a member to be used as input. *yy* can be any two valid characters that do not duplicate any *xx* used in an M*xx*BSDS DD statement. Use the same characters that identify the M*yy*ARCHV statement for the same member; do *not* use characters that identify the M*yy*ARCHV statement for any other member. *n* is a number from 1 to 16. Assign values of *n* in the same way as for ACTIVE*n* DD statements. You can use this statement if the BSDS data sets are unavailable or if you want only some of the log data sets from selected members of the group. If you name the BSDS of a member by a M*xx*BSDS DD statement, do not name the log of the same member by an M*yy*ACT*n* statement. M*yy*ACT*n* statements are ignored if you use the GROUP DD statement.

The DD statements must specify the log data sets in ascending order of log RBA (or LRSN) range. If both ARCHIVE and ACTIVE*n* DD statements are included, the first archive data set must contain the lowest log RBA or LRSN value. If the JCL specifies the data sets in a different order, the job terminates with an error return code with a GET request that tries to access the first record breaking the sequence. If the log ranges of the two data sets overlap, this is not considered an error; instead, the GET function skips over the duplicate data in the second data set and returns the next record. The distinction between out-of-order and overlap is as follows:

- **Out-of-order condition** occurs when the log RBA or LRSN of the first record in a data set is greater than that of the first record in the following data set.
- **Overlap condition** occurs when the out-of-order condition is not met but the log RBA or LRSN of the last record in a data set is greater than that of the first record in the following data set.

Gaps within the log range are permitted. A gap is created when one or more log data sets containing part of the range to be processed are not available. This can happen if the data set was not specified in the JCL or is not reflected in the BSDS. When the gap is encountered, an exception return code value is set, and the next complete record after the gap is returned.

Normally, the BSDS DD name is supplied in the JCL, rather than a series of ACTIVE DD names or a concatenated set of data sets for the ARCHIVE ddname. This is commonly referred to as “running in BSDS mode”.

Data sharing members that participate in a read

The number of members whose logs participate in a particular read request is determined by:

- The number of members in the group, if you use the GROUP DD statement
- Otherwise, the number of different *xx*s and *yy*s used in the M*xx* and M*yy* type DD statements.

For example, assume you need to read log records from members S1, S2, S3, S4, S5 and S6.

S1 and S2 locate their log data sets by their BSDSs.
S3 and S4 need both archive and active logs.

S4 has two active log data sets.
S5 needs only its archive log.
S6 needs only one of its active logs.

Then you need the following DD statements to specify the required log data sets:

MS1BSDS	MS2BSDS	MS3ARCHV MS3ACT1	MS4ARCHV MS4ACT1 MS4ACT2	MS5ARCHV	MS6ACT1

The order of the DD statements in the JCL stream is not important.

Registers and return codes

The request macro invoking these services can be used by reentrant programs. The macro requires that register 13 point to an 18-word save area at invocation. In addition, registers 0, 1, 14, and 15 are used as work and linkage registers. A return code is passed back in register 15 at the completion of each request. When the return code is nonzero, a reason code is placed in register 0. Return codes identify a class of errors, while the reason code identifies a specific error condition of that class. The stand-alone log return codes are shown in Table 291.

Table 291. Stand-alone log return codes

Return code	Explanation
0	Successful completion.
4	Exception condition (for example, end of file), not an error. This return code is not applicable for OPEN and CLOSE requests.
8	Unsuccessful completion due to improper user protocol.
12	Unsuccessful completion. Error encountered during processing of a valid request.

The stand-alone log services invoke executable macros that can execute only in 24-bit addressing mode and reference data below the 16-MB line. User-written applications should be link-edited as AMODE(24), RMODE(24).

Stand-alone log OPEN request

Issue this request when you want to initialize the stand-alone log services. The syntax for the stand-alone log OPEN request is:

```
{label} DSNJSLR   FUNC=OPEN
                  ,LRSN=YES³NO
                  ,DDNAME= address or (Reg. 2-12) optional
                  ,RANGE= address or (Reg. 2-12) optional
                  ,PMO=CI or RECORD
```

Keyword **Explanation**

FUNC=OPEN Requests the stand-alone log OPEN function.

LRSN Tells DB2 how to interpret the log range:
NO: the log range is specified as RBA values. This is the default.
YES: the log range is specified as LRSN values.

DDNAME Specifies the address of an 8-byte area which contains the ddname to be used as an alternate to a ddname of the BSDS when the BSDS is opened, or a register that contains that address.

RANGE Specifies the address of a 12-byte area containing the log range to be processed by subsequent GET requests against the request block generated by this request, or a register that contains that address.

If LRSN=NO, then the range is specified as RBA values. If LRSN=YES, then the range is specified as LRSN values.

The first 6 bytes contain the low RBA or LRSN value. The first complete log record with an RBA or LRSN value equal to or greater than this value is the record accessed by the first log GET request against the request block. The last 6 bytes contain the end of the range or high RBA or LRSN value. An end-of-data condition is returned when a GET request tries to access a record with a starting RBA or LRSN value greater than this value. A value of 6 bytes of X'FF' indicates that the log is to be read until either the end of the log (as specified by the BSDS) or the end of the data in the last JCL-specified log data set is encountered.

If BSDS, GROUP, or MxxBSDS DD statements are used for locating the log data sets to be read, the RANGE parameter is required. If the JCL determines the log data sets to be read, the RANGE parameter is optional.

PMO Specifies the processing mode. You can use OPEN to retrieve either log records or control intervals in the same manner. Specify PMO=CI or RECORD, then use GET to return the data you have selected. The default is RECORD.

The rules remain the same regarding control intervals and the range specified for the OPEN function. Control intervals must fall within the range specified on the RANGE parameter.

Output	Explanation
GPR 1	General-purpose register 1 contains the address of a request block on return from this request. This address must be used for subsequent stand-alone log requests. When no more log GET operations are required by the program, this request block should be used by a FUNC=CLOSE request.
GPR 15	General-purpose register 15 contains a return code upon completion of a request. For nonzero return codes, a corresponding reason code is contained in register 0. The return codes are listed and explained in Table 291 on page 1129.
GPR 0	General-purpose register 0 contains a reason code associated with a nonzero return code in register 15.

See Part 3 of *DB2 Codes* for reason codes that are issued with the return codes.

Log control interval retrieval: You can use the PMO option to retrieve log control intervals from archive log data sets. DSNJSLR also retrieves log control intervals from the active log if the DB2 system is not active. During OPEN, if DSNJSLR detects that the control interval range is not within the archive log range available (for example, the range purged from BSDS), an error condition is returned.

Specify CI and use GET to retrieve the control interval you have chosen. The rules remain the same regarding control intervals and the range specified for the OPEN function. Control intervals must fall within the range specified on the RANGE parameter.

Log control interval format: A field in the last 7 bytes of the control interval, offset 4090, contains a 7-byte timestamp. This field reflects the time at which the control interval was written to the active log data set. The timestamp is in store clock (STCK) format and is the high-order 7 bytes of the 8-byte store clock value.

Stand-alone log GET request

This request returns a pointer to a buffer containing the next log record based on position information in the request block.

A log record is available in the area pointed to by the request block until the next GET request is issued. At that time, the record is no longer available to the requesting program. If the program requires reference to a log record's content after requesting a GET of the next record, the program must move the record into a storage area that is allocated by the program.

The first GET request, after a FUNC=OPEN request that specified a RANGE parameter, returns a pointer in the request feedback area. This points to the first record with a log RBA value greater than or equal to the low log RBA value specified by the RANGE parameter. If the RANGE parameter was not specified on the FUNC=OPEN request, then the data to be read is determined by the JCL specification of the data sets. In this case, a pointer to the first complete log record in the data set that is specified by the ARCHIVE, or by ACTIVE1 if ARCHIVE is omitted, is returned. The next GET request returns a pointer to the next record in ascending log RBA order. Subsequent GET requests continue to move forward in log RBA sequence until the function encounters the end of RANGE RBA value, the end of the last data set specified by the JCL, or the end of the log as determined by the bootstrap data set.

The syntax for the stand-alone log GET request is:

```
{label} DSNJSLR   FUNC=GET
                  ,RBR=(Reg. 1-12)
```

Keyword	Explanation
FUNC=GET	Requests the stand-alone log GET function.
RBR	Specifies a register that contains the address of the request block this request is to use. Although you can specify any register between 1 and 12, using register 1 (RBR=(1)) avoids the generation of an unnecessary load register and is therefore more efficient. The pointer to the request block (that is passed in register *n* of the RBR=(n) keyword) must be used by subsequent GET and CLOSE function requests.

Output

Explanation

GPR 15
General-purpose register 15 contains a return code upon completion of a request. For nonzero return codes, a corresponding reason code is contained in register 0. Return codes are listed and explained in Table 291 on page 1129.

GPR 0 General-purpose register 0 contains a reason code associated with a nonzero return code in register 15.

See Part 3 of *DB2 Codes* for reason codes that are issued with the return codes.

Reason codes 00D10261 - 00D10268 reflect a damaged log. In each case, the RBA of the record or segment in error is returned in the stand-alone feedback block field (SLRFRBA). A damaged log can impair DB2 restart; special recovery procedures are required for these circumstances. For recovery from these errors, refer to Chapter 22, “Recovery scenarios,” on page 517.

Information about the GET request and its results is returned in the request feedback area, starting at offset X'00'. If there is an error in the length of some record, the control interval length is returned at offset X'0C' and the address of the beginning of the control interval is returned at offset X'08'.

On return from this request, the first part of the request block contains the feedback information that this function returns. Mapping macro DSNDSLRF defines the feedback fields which are shown in Table 292. The information returned is status information, a pointer to the log record, the length of the log record, and the 6-byte log RBA value of the record.

Table 292. Stand-alone log get feedback area contents

Field name	Hex offset	Length (bytes)	Field contents
SLRFRC	00	2	Log request return code
SLRFINFO	02	2	Information code returned by dynamic allocation. Refer to the z/OS SPF job management publication for information code descriptions
SLRFERCD	04	2	VSAM or dynamic allocation error code, if register 15 contains a nonzero value.
SLRFRG15	06	2	VSAM register 15 return code value.
SLRFFRAD	08	4	Address of area containing the log record or CI
SLRFRCLL	0C	2	Length of the log record or RBA
SLRFRBA	0E	6	Log RBA of the log record
SLRFDDNM	14	8	ddname of data set on which activity occurred

Stand-alone log CLOSE request

This request deallocates any log data sets that were dynamically allocated by previous processing. Also, all storage that was obtained by previous functions, including the request block specified on this request, is freed.

The syntax for the stand-alone log CLOSE request is:

```
{label} DSNJSLR   FUNC=CLOSE
                  ,RBR=(Reg. 1-12)
```

Keyword **Explanation**

FUNC=CLOSE
Requests the CLOSE function.

RBR
Specifies a register that contains the address of the request block that this function uses. Although you can specify any register between 1 and 12, using register 1 (RBR=(1)) avoids the generation of an unnecessary load register and is therefore more efficient.

Output **Explanation**

GPR 15
Register 15 contains a return code upon completion of a request.

For nonzero return codes, a corresponding reason code is contained in register 0. The return codes are listed and explained in Table 291 on page 1129.

GPR 0 Register 0 contains a reason code that is associated with a nonzero return code that is contained in register 15. The only reason code used by the CLOSE function is 00D10030.

See Part 3 of *DB2 Codes* for reason code details.

Sample application that uses stand-alone log services

Figure 153 shows sample segments of an assembler program that uses the three stand-alone log services (OPEN, GET, and CLOSE) to process one log record.

```
TSTJSLR5 CSECT
   ⋮
OPENCALL EQU   *
         LA    R2,NAME                GET BSDS DDNAME ADDRESS
         LA    R3,RANGER              GET ADDRESS OF RBA RANGE
         DSNJSLR FUNC=OPEN,DDNAME=(R2),RANGE=(R3)
         LTR   R15,R15                CHECK RETURN CODE FROM OPEN
         BZ    GETCALL                OPEN OK, DO GET CALLS
   ⋮
*****************************************************************
*        HANDLE ERROR FROM OPEN FUNCTION AT THIS POINT          *
*****************************************************************
   ⋮
GETCALL  EQU   *
         DSNJSLR FUNC=GET,RBR=(R1)
         C     R0,=X'00D10020'        END OF RBA RANGE ?
         BE    CLOSE                  YES, DO CLEANUP
         C     R0,=X'00D10021'        RBA GAP DETECTED ?
         BE    GAPRTN                 HANDLE RBA GAP
         LTR   R15,R15                TEST RETURN CODE FROM GET
         BNZ   ERROR
   ⋮
   ⋮
*****************************************************************
*       PROCESS RETURNED LOG RECORD AT THIS POINT. IF LOG RECORD *
*       DATA MUST BE KEPT ACROSS CALLS, IT MUST BE MOVED TO A    *
*       USER-PROVIDED AREA.                                      *
*****************************************************************
         USING SLRF,1                 BASE SLRF DSECT
         L     R8,SLRFFRAD            GET LOG RECORD START ADDR
         LR    R9,R8
         AH    R9,SLRFRCLL            GET LOG RECORD END ADDRESS
         BCTR  R9,R0
   ⋮
CLOSE    EQU   *
         DSNJSLR FUNC=CLOSE,RBR=(1)
   ⋮
NAME     DC    C'DDBSDS'
RANGER   DC    X'00000000000000000005FFFF'
   ⋮
         DSNDSLRB
         DSNDSLRF
         EJECT
R0       EQU   0
R1       EQU   1
R2       EQU   2
   ⋮
R15      EQU   15
         END
```

Figure 153. Excerpts from a sample program using stand-alone log services

Reading log records with the log capture exit routine

You can use the log capture exit routine to capture DB2 log data in real time. This installation exit routine presents log data to a log capture exit routine when the data is written to the DB2 active log. This exit routine is not intended to be used for general purpose log auditing or tracking. The IFI interface (see "Reading log records with IFI" on page 1122) is designed (and is more appropriate) for this purpose.

The log capture exit routine executes in an area of DB2 that is critical for performance. As such, it is primarily intended as a mechanism to capture log data for recovery purposes. In addition, the log capture exit routine operates in a very restrictive z/OS environment, which severely limits its capabilities as a stand-alone routine.

To capture log records with this exit routine, you must first write an exit routine (or use the one provided by the preceding program offering) that can be loaded and called under the various processing conditions and restrictions required of this exit routine. See "Log capture routines" on page 1101 and refer to the previous sections of this appendix, "Contents of the log" on page 1111 and "The physical structure of the log" on page 1116.

Appendix D. Interpreting DB2 trace output

The information in this appendix is Product-sensitive Programming Interface and Associated Guidance Information, as defined in “Notices” on page 1301.

When you activate a DB2 trace, it produces trace records based on the parameters you specified for the START TRACE command. Each record identifies one or more significant DB2 events. You can use OMEGAMON to format, print, and interpret DB2 trace output. If you do not have OMEGAMON, or you want to do your own analysis of the trace output, you can use the information in this appendix and the trace field descriptions that are shipped with DB2. By examining a DB2 trace record, you can determine the type of trace that produced the record (statistics, accounting, audit, performance, monitor, or global) and the event the record reports.

This appendix contains the following sections:

- “The sections of the trace output”
- “Trace field descriptions” on page 1150

Note that when the trace output indicates a particular release level, 'xx'varies according to the actual release of DB2.

The sections of the trace output

Trace records can be written to SMF or GTF. In both cases, the record contains up to four basic sections:

- An SMF or GTF writer header section
- A self-defining section
- A product section
- Zero or more data sections

Figure 154 shows the format of DB2 trace records.

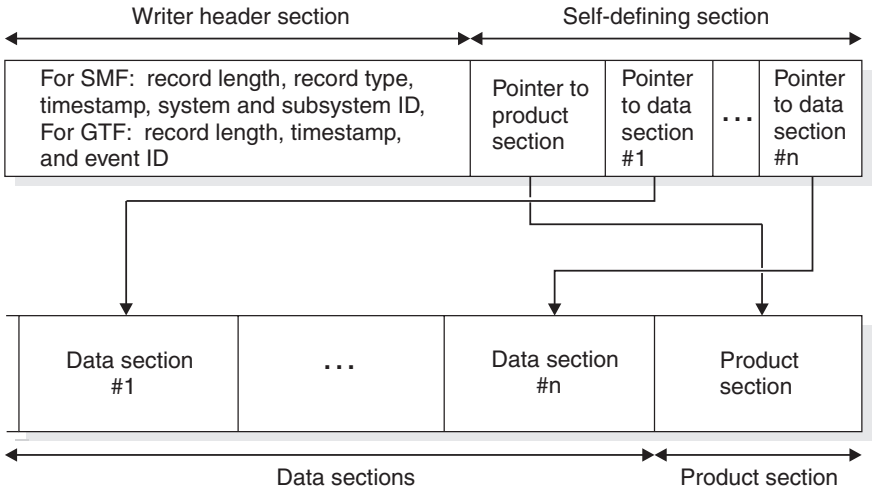


Figure 154. General format of trace records written by DB2

© Copyright IBM Corp. 1982, 2008

The writer header section begins at the first byte of the record and continues for a fixed length. (The GTF writer header is longer than the SMF writer header.)

The self-defining section follows the writer header section (both GTF and SMF) and is further described in "Self-defining section" on page 1143. The first self-defining section always points to a special data section called the product section. Among other things, the product section contains an instrumentation facility component identifier (IFCID). Descriptions of the records differ for each IFCID. For a list of records, by IFCID, for each class of a trace, see the description of the START TRACE command in *DB2 Command Reference*.

The product section also contains field QWHSNSDA, which indicates how many self-defining data sections the record contains. You can use this field to keep from trying to access data sections that do not exist. In trying to interpret the trace records, remember that the various keywords you specified when you started the trace determine whether any data is collected. If no data has been collected, field QWHSNSDA shows a data length of zero.

SMF writer header section

In SMF, writer headers for statistics records are mapped by macro DSNDQWST, for accounting records by DSNDQWAS, and for performance, audit, and monitor records by DSNDQWSP. When these macros are assembled, they include the other macros necessary to map the remainder of the trace records sent to SMF.

The SMF writer header section begins at the first byte of the record. After establishing addressability, you can examine the header fields. The fields are described in Table 293.

Table 293. Contents of SMF writer header section

Hex Offset	DSNDQWST	DSNDQWAS	DSNDQWSP	Description
0	SM100LEN	SM101LEN	SM102LEN	Total length of SMF record
2	SM100SGD	SM101SGD	SM102SGD	Segment descriptor
4	SM100FLG	SM101FLG	SM102FLG	System indicator
5	SM100RTY	SM101RTY	SM102RTY	SMF record type: • Statistics=100(dec) • Accounting=101(dec) • Monitor=102(dec) • Audit=102(dec) • Performance=102(dec)
6	SM100TME	SM101TME	SM102TME	SMF record timestamp, time portion
A	SM100DTE	SM101DTE	SM102DTE	SMF record timestamp, date portion
E	SM100SID	SM101SID	SM102SID	System ID
12	SM100SSI	SM101SSI	SM102SSI	Subsystem ID
16	SM100STF	SM101STF	SM102STF	Reserved
17	SM100RI	SM101RI	SM102RI	Reserved
18	SM100BUF	SM101BUF	SM102BUF	Reserved
1C	SM100END	SM101END	SM102END	End of SMF header

Figure 155 is a sample of the first record of the DB2 performance trace output sent to SMF.

```
          A        B  C      D       E         F                  G  H
000000   01240000 0E660030  9EEC0093 018FF3F0   F9F0E2E2 D6D70000  00000000 0000008C
         I   J    K         L   M    N
000020   00980001 0000002C  005D0001 00550053   4DE2E3C1 D9E340E3  D9C1C3C5 404DE2E3
000040   C1E3405D C3D3C1E2  E2404D5C 405DD9D4   C9C4404D 5C405DD7  D3C1D540 4D5C405D
000060   C1E4E3C8 C9C4404D  5C405DC9 C6C3C9C4   404D5C40 5DC2E4C6  E2C9E9C5 404D5C40
                                    O           P   Q  R
000080   5D000000 01000101  01000000 004C0110   000402xx 00B3AB78  E2E2D6D7 A6E9BACB
                                                S
0000A0   F6485E02 00000003  00000021 00000001   E2C1D5E3 C16DE3C5  D9C5E2C1 6DD3C1C2
0000C0   C4C2F2D5 C5E34040  D3E4D5C4 F0404040   A6E9BACB F4570001  004C0200 E2E8E2D6
0000E0   D7D94040 F0F2F34B  C7C3E2C3 D5F6F0F2   E2E2D6D7 40404040  40404040 40404040
000100   E2E8E2D6 D7D94040  00000000 00000000   00000000 00000000  00000000 00000000
000120   00000000T
```

Figure 155. DB2 trace output sent to SMF (printed with DFSERA10 print program of IMS)

Key to Figure 155	Description
A 0124	Record length (field SM102LEN); beginning of SMF writer header section
B 66	Record type (field SM102RTY)
C 0030 9EEC	Time (field SM102TME)
D 0093 018F	Date (field SM102DTE)
E F3F0 F9F0	System ID (field SM102SID)
F E2E2 D6D7	Subsystem ID (field SM102SSI)
G	End of SMF writer header section
H 0000008C	Offset to product section; beginning of self-defining section
I 0098	Length of product section
J 0001	Number of times the product section is repeated
K 0000002C	Offset to first (in this case, only) data section
L 005D	Length of data section
M 0001	Number of times the data section is repeated
N 00550053	Beginning of data section
O	Beginning of product section
P 0004	IFCID (field QWHSIID)
Q 02	Number of self-defining sections in the record (field QWHSNSDA)
R xx	Release indicator number (field QWHSRN); this varies according to the actual level of DB2 you are using.
S E2C1D5E3...	Local location name (16 bytes)
T	End of first record

GTF writer header section

The length and content of the writer header section differs between SMF and GTF records, but the other sections of the records are the same for SMF and GTF.

The GTF writer header section begins at the first byte of the record. After establishing addressability, you can examine the fields of the header. The writer headers for trace records sent to GTF are always mapped by macro DSNDQWGT. The fields are described in Table 294 on page 1138.

Table 294. Contents of GTF writer header section

Offset	Macro DSNDQWGT field	Description
0	QWGTLEN	Length of Record
2		Reserved
4	QWGTAID	Application identifier
5	QWGTFID	Format ID
6	QWGTTIME	Timestamp; you must specify TIME=YES when you start GTF.
14	QWGTEID	Event ID: X'EFB9'
16	QWGTASCB	ASCB address
20	QWGTJOBN	Job name
28	QWGTHDRE	Extension to header
28	QWGTDLEN	Length of data section
30	QWGTDSCC	Segment control code 0=Complete 2=Last 1=First 3=Middle
31	QWGTDZZ2	Reserved
32	QWGTSSID	Subsystem ID
36	QWGTWSEQ	Sequence number
40	QWGTEND	End of GTF header

Figure 156 on page 1139 contains trace output sent to GTF.

```
DFSERA10 - PRINT PROGRAM

000000   001A0000 0001FFFF  94B6A6E9 BD6636FA   5C021000 00010000  0000
         A          B                C            D                E  F
000000   011C0000 FF00A6E9  C33E28F7 DD03EFB9   00F91400 E2E2D6D7  D4E2E3D9 01000100
         G             H  I       J   K     L        M   N     O
000020   E2E2D6D7 00000001  000000A0 00980001   00000038 00680001  0060005E 4DE2E3C1
000040   D9E340E3 D9C1C3C5  404DE2E3 C1E3405D   C3D3C1E2 E2404D5C  405DD9D4 C9C4404D
000060   5C405DC4 C5E2E340  4DC7E3C6 405DD7D3   C1D5404D 5C405DC1  E4E3C8C9 C4404D5C
000080   405DC9C6 C3C9C440  4D5C405D C2E4C6E2   C9E9C540 4D5C405D  FFFFFFFF 00040101
         P        Q  R S
0000A0   004C0110 000402xx  00B3ADB8 E2E2D6D7   A6E9C33E 28EF4403  00000006 00000001
                T
0000C0   00000001 E2C1D5E3  C16DE3C5 D9C5E2C1   6DD3C1C2 C4C2F2D5  C5E34040 D3E4D5C4
0000E0   F0404040 A6E9C33E  271F0001 004C0200   E2E8E2D6 D7D94040  F0F2F34B C7C3E2C3
000100   D5F6F0F2 E2E2D6D7  40404040 40404040   40404040 E2E8E2D6  D7D94040
                                                                                U
000000   00440000 FF00A6E9  C33E2901 1303EFB9   00F91400 E2E2D6D7  D4E2E3D9 00280200
000020   E2E2D6D7 00000001  00000000 00000000   00000000 00000000  00000000 00000000
000040   00000000 V
         W                                                                      X
000000   011C0000 FF00A6E9  C33E2948 E203EFB9   00F91400 E2E2D6D7  D4E2E3D9 01000100
000020   E2E2D6D7 00000002  000006D8 004C0001   00000090 001C0004  00000100 001C000E
000040   00000288 0018000E  00000590 00400001   000005D0 00740001  00000480 00440001
000060   000003D8 00800001  00000458 00280001   00000644 00480001  000004E4 00AC0001
000080   0000068C 004C0001  000004C4 00200001   D4E2E3D9 00000001  762236F2 00000000
0000A0   59F48900 001E001E  00F91400 C4C2D4F1   00000001 1A789573  00000000 95826100
0000C0   001F001F 00F90E00  C4C9E2E3 00000000   3413C60E 00000000  1C4D0A00 00220022
0000E0   00F90480 C9D9D3D4  00000000 0629E2BC   00000000 145CE000  001D001D 00F91600
000100   E2D4C640 00000046  00000046 00000000   00000000 00000000  00000000
                                                                                Y
000000   011C0000 FF00A6E9  C33E294B 1603EFB9   00F91400 E2E2D6D7  D4E2E3D9 01000300
000020   E2E2D6D7 00000002  D9C5E240 00000000   00000000 00000000  00000000 00000000
000040   00000000 C7E3C640  00000001 00000001   00000000 00000000  00000000 00000000
000060   E2D9E540 00000000  00000000 00000000   00000000 00000000  00000000 E2D9F140
000080   00000156 000000D2  00000036 00000036   00000000 00000004  E2D9F240 00000000
0000A0   00000000 00000000  00000000 00000000   00000000 D6D7F140  00000000 00000000
0000C0   00000000 00000000  00000000 00000000   D6D7F240 00000000  00000000 00000000
0000E0   00000000 00000000  00000000 D6D7F340   00000000 00000000  00000000 00000000
000100   00000000 00000000  D6D7F440 00000000   00000000 00000000  00000000
                                                                                Y
000000   011C0000 FF00A6E9  C33E294D 3C03EFB9   00F91400 E2E2D6D7  D4E2E3D9 01000300
000020   E2E2D6D7 00000002  00000000 00000000   D6D7F540 00000000  00000000 00000000
000040   00000000 00000000  00000000 D6D7F640   00000000 00000000  00000000 00000000
000060   00000000 00000000  D6D7F740 00000000   00000000 00000000  00000000 00000000
000080   00000000 D6D7F840  00000000 00000000   00000000 00000000  00000000 00000000
0000A0   00010000 0000000E  0000000D 00000000   00000000 00000000  00020000 0000000D
0000C0   0000000D 00000000  00000000 00000000   00030000 00000003  00000003 00000000
0000E0   00000000 00000000  00040000 00000006   00000006 00000000  00000000 00000000
000100   00050000 00000005  00000005 00000000   00000000 00000000  006A0000
                                                                                Y
000000   011C0000 FF00A6E9  C33E294F 6103EFB9   00F91400 E2E2D6D7  D4E2E3D9 01000300
000020   E2E2D6D7 00000002  00000005 00000005   00000000 00000000  00000000 008C0000
000040   00000000 00000000  00000000 00000000   00000000 008D0000  00000000 00000000

  ⋮
                                                                                Z
000000   00780000 FF00A6E9  C33E2957 D103EFB9   00F91400 E2E2D6D7  D4E2E3D9 005C0200
                                     AA
000020   E2E2D6D7 00000002  00000000 004C011A   00010D31 02523038  E2E2D6D7 A6E9C33E
000040   29469A03 0000000E  00000002 00000001   E2C1D5E3 C16DE3C5  D9C5E2C1 6DD3C1C2
000060   40404040 40404040  40404040 40404040   A6E9B6B4 9A2B0001
```

Figure 156. DB2 trace output sent to GTF (spanned records printed with DFSERA10 print program of IMS)

Key to Figure 156	Description
A 011C	Record length (field QWGTLEN); beginning of GTF writer header section
B A6E9 C33E28F7 DD03	Timestamp (field QWGTTIME)
C EFB9	Event ID (field QWGTEID)
D E2E2D6D7 D4E2E3D9	Job name (field QWGTJOBN)
E 0100	Length of data section
F 01	Segment control code (01 = first segment of the first record)

Key to Figure 156 on page 1139	Description
G E2E2D6D7	Subsystem ID (field QWGTSSID)
H	End of GTF writer header section
I 000000A0	Offset to product section; beginning of self-defining section
J 0098	Length of product section
K 0001	Number of times the product section is repeated
L 00000038	Offset to first (in this case, only) data section
M 0068	Length of data section
N 0001	Number of times the data section is repeated
O 0060005E	Beginning of data section
P 004C0110...	Beginning of product section
Q 0004	IFCID (field QWHSIID)
R 02	Number of self-defining sections in the record (field QWHSNSDA)
S xx	Release indicator number (field QWHSRN); this varies according to the actual release level of DB2 you are using.
T E2C1D5E3...	Local location name (16 bytes)
U 02	Last segment of the first record
V	End of first record
W	Beginning of GTF header for new record
X 01	First segment of a spanned record (QWGTDSCC = QWGTDS01)
Y 03	Middle segment of a spanned record (QWGTDSCC = QWGTDS03)
Z 02	Last segment of a spanned record (QWGTDSCC = QWGTDS02)
AA 004C	Beginning of product section

GTF records are blocked to 256 bytes. Because some of the trace records exceed the GTF limit of 256 bytes, they have been blocked by DB2. Use the following logic to process GTF records:

1. Is the GTF event ID of the record equal to the DB2 ID (that is, does QWGTEID = X'*x*FB9')?

   If it is *not* equal, get another record.

   If it is equal, continue processing.
2. Is the record spanned?

   If it is spanned (that is, QWGTDSCC ¬ = QWGTDS00), test to determine whether it is the first, middle, or last segment of the spanned record.

   a. If it is the *first* segment (that is, QWGTDSCC = QWGTDS01), save the entire record including the sequence number (QWGTWSEQ) and the subsystem ID (QWGTSSID).

   b. If it is a *middle* segment (that is, QWGTDSCC = QWGTDS03), find the first segment matching the sequence number (QWGTSEQ) and on the subsystem ID (QWTGSSID). Then move the data portion immediately after the GTF header to the end of the previous segment.

c. If it is the *last* segment (that is, QWGTDSCC = QWGTDS02), find the first segment matching the sequence number (QWGTSEQ) and on the subsystem ID (QWTGSSID). Then move the data portion immediately after the GTF header to the end of the previous record.

   Now process the completed record.

If it is **not** spanned, process the record.

Figure 157 on page 1142 shows the same output after it has been processed by a user-written routine, which follows the logic that was outlined previously.

```
000000   01380000 FF00A6E9  DCA7E275 1204EFB9   00F91400 E2E2D6D7  D4E2E3D9 011C0000
000020   E2E2D6D7 00000019  000000A0 00980001   00000038 00680001  0060005E 4DE2E3C1
000040   D9E340E3 D9C1C3C5  404DE2E3 C1E3405D   C3D3C1E2 E2404D5C  405DD9D4 C9C4404D
000060   5C405DC4 C5E2E340  4DC7E3C6 405DD7D3   C1D5404D 5C405DC1  E4E3C8C9 C4404D5C
000080   405DC9C6 C3C9C440  4D5C405D C2E4C6E2   C9E9C540 4D5C405D  00000001 00040101
0000A0   004C0110 000402xx  00B3ADB8 E2E2D6D7   0093018F 11223310  0000000C 00000019
0000C0   00000001 E2C1D5E3  C16DE3C5 D9C5E2C1   6DD3C1C2 C4C2F2D5  C5E34040 D3E4D5C4
0000E0   F0404040 A6E9DCA7  DF960001 004C0200   E2E8E2D6 D7D94040  F0F2F34B C7C3E2C3
000100   D5F6F0F2 E2E2D6D7  40404040 40404040   40404040 E2E8E2D6  D7D94040 00000000
000120   00000000 00000000  00000000 00000000   00000000 00000000
         A                                B
000000   07240000 FF00A6E9  DCA8060C 2803EFB9   00F91400 E2E2D6D7  D4E2E3D9 07080000
                       C  D                     E
000020   E2E2D6D7 0000001A  000006D8 004C0001   00000090 001C0004  00000100 001C000E
000040   00000288 0018000E  00000590 00400001   000005D0 00740001  00000480 00440001
000060   000003D8 00800001  00000458 00280001   00000644 00480001  000004E4 00AC0001
                            F
000080   0000068C 004C0001  000004C4 00200001   D4E2E3D9 00000003  27BCFDBC 00000000
0000A0   AB000300 001E001E  00F91400 C4C2D4F1   00000001 1DE8AEE2  00000000 DB0CB200
0000C0   001F001F 00F90E00  C4C9E2E3 00000000   4928674B 00000000  217F6000 00220022
0000E0   00F90480 C9D9D3D4  00000000 07165F79   00000000 3C2EF500  001D001D 00F91600
000100   E2D4C640 0000004D  0000004D 00000000   00000000 00000000  00000000 D9C5E240
000120   00000000 00000000  00000000 00000000   00000000 00000000  C7E3C640 00000019
000140   00000019 00000000  00000000 00000000   00000000 E2D9E540  00000000 00000000
000160   00000000 00000000  00000000 00000000   E2D9F140 00000156  000000D2 00000036
000180   00000036 00000000  00000004 E2D9F240   00000092 00000001  00000091 00000091
0001A0   00000000 0000000C  D6D7F140 00000002   00000001 00000001  00000000 00010000
0001C0   20000004 D6D7F240  00000000 00000000   00000000 00000000  00000000 00000000
0001E0   D6D7F340 00000000  00000000 00000000   00000000 00000000  00000000 D6D7F440
000200   00000000 00000000  00000000 00000000   00000000 00000000  D6D7F540 00000000
000220   00000000 00000000  00000000 00000000   00000000 D6D7F640  00000000 00000000
000240   00000000 00000000  00000000 00000000   D6D7F740 00000000  00000000 00000000
000260   00000000 00000000  00000000 D6D7F840   00000000 00000000  00000000 00000000
000280   00000000 00000000  00010000 00000042   00000011 00000030  00000000 00000000
0002A0   00020000 00000041  00000011 00000030   00000000 00000000  00030000 00000003
0002C0   00000003 00000000  00000000 00000000   00040000 0000000C  0000000C 00000000
0002E0   00000000 00000000  00050000 0000000B   0000000A 00000001  00000000 00000000
000300   006A0000 0000000C  0000000B 00000001   00000000 00000000  008C0000 00000000
000320   00000000 00000000  00000000 00000000   008D0000 00000000  00000000 00000000
000340   00000000 00000000  008E0000 00000000   00000000 00000000  00000000 00000000
000360   008F0000 00000000  00000000 00000000   00000000 00000000  00900000 00000000
000380   00000000 00000000  00000000 00000000   00910000 00000000  00000000 00000000
0003A0   00000000 00000000  00920000 00000000   00000000 00000000  00000000 00000000
0003C0   00CA0000 00000041  00000011 00000030   00000000 00000000  00000000 00000000
0003E0   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
000400   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
000420   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
000440   00000000 00000000  00000000 00000004   00000000 00000000  000005D4 00000130
000460   0000000D 0000000A  00000029 00000009   000000C3 00000000  00000000 00000000
000480   00000001 0000000C  00000000 04A29740   00000000 00000000  00000001 00000000
0004A0   00000001 00000000  00000000 00000000   00000000 00000000  00000000 00000000
0004C0   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
0004E0   00000000 E2C1D56D  D1D6E2C5 40404040   40404040 00000000  00000002 00000003
000500   00000000 000004A8  000005C7 00000000   00000001 00000003  00000003 00000000
000520   00000001 00000000  00000001 00000000   00000000 00000000  00000000 00000000
000540   00000002 00000001  00000000 00000000   00000000 00000000  00000000 00000000
000560   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
000580   00000000 00000000  00000002 00000000   00000003 00000000  00000003 00000006
0005A0   00000000 00000000  00000000 00000000   00000005 00000003  00000000 00000000
0005C0   00000000 00000003  00000000 00000000   00000000 00000000  00000000 00000000
0005E0   00000000 00000000  0000000C 00000001   00000000 00000007  00000000 00000000
000600   00000000 00000000  00000000 00000001   00000000 00000000  00000000 00000000
```

Figure 157. DB2 trace output sent to GTF (assembled with a user-written routine and printed with DFSERA10 print program of IMS) (Part 1 of 2)

```
000620   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
000640   00000000 003C0048  D8E2E2E3 00000035   00000006 00000002  0000009E 0000002B
000660   00000078 00000042  00000048 000000EE   0000001B 0000007B  0000004B 00000000
000680   00000000 00000000  00000000 0093004C   D8D1E2E3 00000000  000000FC 0000000E
0006A0   00000000 00000000  0000009D 00000000   00000000 00000016  0000000F 00000018
                                                                    G
0006C0   00000000 00000000  00000000 00000000   00000000 00000000  004C011A 00010Dxx
0006E0   02523038 E2E2D6D7  0093018F 11223324   00000042 0000001A  00000001 E2C1D5E3
000700   C16DE3C5 D9C5E2C1  6DD3C1C2 40404040   40404040 40404040  40404040 A6E9B6B4
000720   9A2B0001 H
```

Figure 157. DB2 trace output sent to GTF (assembled with a user-written routine and printed with DFSERA10 print program of IMS) (Part 2 of 2)

Key to Figure 157 on page 1142	Description
A 0724	Length of assembled record; beginning of GTF writer header section of second record (field QWGTLEN)
B EFB9	GTF event ID (field QWGTEID)
C	End of GTF writer header section of second record
D 000006D8	Offset to product section
E 00000090	Offset to first data section
F 000004C4	Offset to last data section
G 004C011A	Beginning of product section
H	End of second record

Self-defining section

The self-defining section following the writer header contains pointers that enable you to find the product and data sections, which contain the actual trace data.

Each “pointer” is a descriptor that contains three fields, which are:

1. A fullword that contains the offset from the beginning of the record to the data section.
2. A halfword that contains the length of each item in the data section. If this value is zero, the length of the items are varied.
3. A halfword that contains the number of times that the data section is repeated. If the field contains “0”, the data section is not in the record. If it contains a number greater than 1, multiple data items are stored contiguously within that data section. To find the second data item, add the length of the first data item to the address of the first data item (and so forth).

Pointers occur in a fixed order, and their meanings are determined by the IFCID of the record. Different sets of pointers can occur, and each set is described by a separate DSECT. Therefore, to examine the pointers, you must first establish addressability by using the DSECT that provides the appropriate description of the self-defining section. To do this, perform the following steps:

1. Compute the address of the self-defining section.

   The self-defining section begins at label “SM100END” for statistics records, “SM101END” for accounting records, and “SM102END” for performance and audit records. It does not matter which mapping DSECT you use because the length of the SMF writer header is always the same.

   For GTF, use QWGTEND.
2. Determine the IFCID of the record.

   Use the first field in the self-defining section; it contains the offset from the beginning of the record to the product section. The product section contains the IFCID.

The product section is mapped by DSNDQWHS; the IFCID is mapped by QWHSIID.

For statistics records that have IFCID 0001, establish addressability using label “QWS0”; for statistics records having IFCID 0002, establish addressability using label “QWS1”. For accounting records, establish addressability using label “QWA0”. For performance and audit records, establish addressability using label “QWT0”.

After establishing addressability using the appropriate DSECT, use the pointers in the self-defining section to locate the record’s data sections.

Reading the self-defining section for same-length data items

If the length of each item in a data section is the same, the self-defining section indicates the length in a halfword that follows the fullword offset. If this value is “0”, the length of the data items varies. For more information about variable-length data items, see “Reading the self-defining section for variable-length data items.”

The relationship between the contents of the self-defining section “pointers” and the items in a data section for same-length data items is shown in Figure 158.

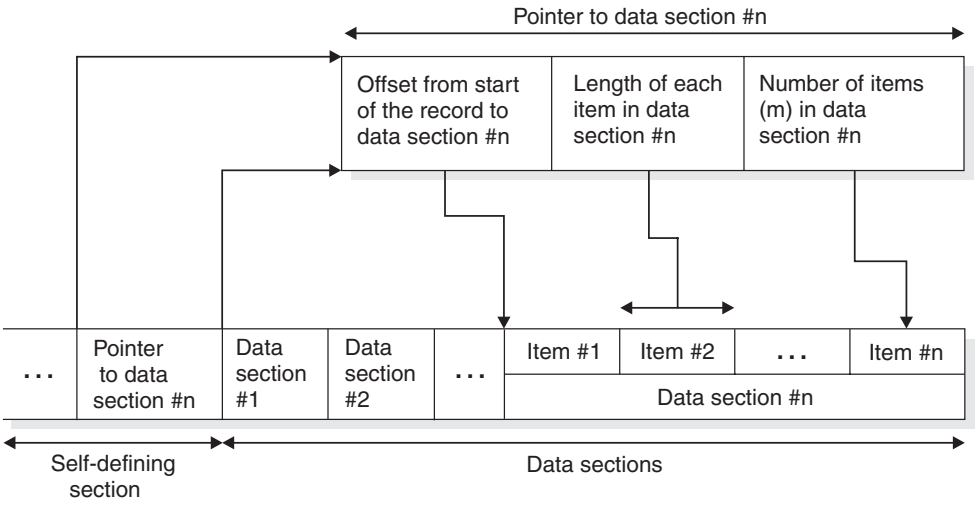


Figure 158. Relationship between self-defining section and data sections for same-length data items

Reading the self-defining section for variable-length data items

If the length of each item in a data section varies, the self-defining section indicates the value “0” in the halfword that follows the fullword offset. The length of each variable-length data item is indicated by two bytes that precede each data item.

The relationship between the contents of the self-defining section “pointers” and the items in a data section for variable-length data items is shown in .

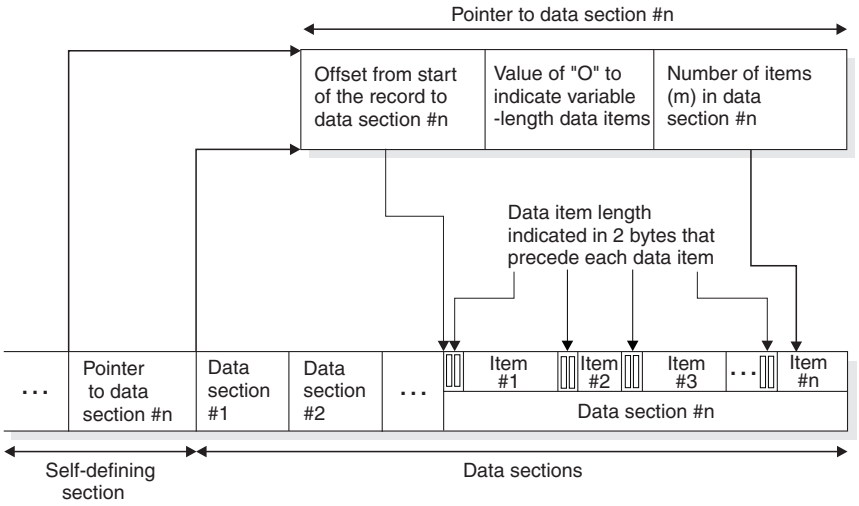


Figure 159. Relationship between self-defining section and data sections for variable-length data items

Product section

The product section for all record types contains the standard header. The other headers (correlation, CPU, distributed, and data sharing data) might also be present. Table 295 shows the contents of the product section standard header.

Table 295. Contents of product section standard header

Hex Offset	Macro DSNDQWHS field	Description
0	QWHSLEN	Length of standard header
2	QWHSTYP	Header type
3	QWHSRMID	RMID
4	QWHSIID	IFCID
6	QWHSRELN	Release number section
6	QWHSNSDA	Number of self-defining sections
7	QWHSRN	DB2 release identifier
8	QWHSACE	ACE address
C	QWHSSSID	Subsystem ID
10	QWHSSTCK	Timestamp—STORE CLOCK value assigned by DB2
18	QWHSISEQ	IFCID sequence number
1C	QWHSWSEQ	Destination sequence number
20	QWHSMTN	Active trace number mask
24	QWHSLOCN	Local location Name
34	QWHSLWID	Logical unit of work ID
34	QWHSNID	Network ID
3C	QWHSLUNM	LU name
44	QWHSLUUV	Uniqueness value

Table 295. Contents of product section standard header (continued)

Hex Offset	Macro DSNDQWHS field	Description
4A	QWHSLUCC	Commit count
4C	QWHSEND	End of product section standard header

Table 296 shows the contents of the product section correlation header.

Table 296. Contents of product section correlation header

Hex Offset	Macro DSNDQWHC field	Description
0	QWHCLEN	Length of correlation header
2	QWHCTYP	Header type
3		Reserved
4	QWHCAID	Authorization ID
C	QWHCCV	Correlation ID
18	QWHCCN	Connection name
20	QWHCPLAN	Plan name
28	QWHCOPID	Original operator ID
30	QWHCATYP	The type of system that is connecting
34	QWHCTOKN	Trace accounting token field
4A		Reserved
4C	QWHCEUID	User ID of at the workstation for the end user
5C	QWHCEUTX	Transaction name for the end user
7C	QWHCEUWN	Workstation name for the end user
8E	QWHCEND	End of product section correlation header

Table 297 shows the contents of the CPU header.

Table 297. Contents of CPU header

Hex Offset	Macro DSNDQWHU field	Description
0	QWHULEN	Length of CPU header
2	QWHUTYP	Header type
3		Reserved
4	QWHUCPU	CPU time of MVS TCB or SRB dispatched
C	QWHUCNT	Count field reserved
E	QWHUEND	End of header

Table 298 shows the contents of the distributed data header.

Table 298. Contents of distributed data header

Hex Offset	Macro DSNDQWHD field	Description
0	QWHDLEN	Length of the distributed header

Table 298. Contents of distributed data header (continued)

Hex Offset	Macro DSNDQWHD field	Description
2	QWHDTYP	Header type
3		Reserved
4	QWHDRQNM	Requester location name
14	QWHDTSTP	Timestamp for DBAT trace record
1C	QWHDSVNM	EXCSAT SRVNAM parameter
2C	QWHDPRID	ACCRDB PRDID parameter
30	QWHDEND	End of distributed header

Table 299 shows the contents of the trace header.

Table 299. Contents of trace header

Hex Offset	Macro DSNDQWHT field	Description
0	QWHTLEN	Length of the trace header
2	QWHTTYP	Header type
3		Reserved
4	QWHTTID	Event ID
6	QWHTTAG	ID specified on DSNWTRC macro
7	QWHTFUNC	Resource manager function code. Default is 0.
8	QWHTEB	Execution block address
C	QWHTPASI	Prior address space ID - EPAR
E	QWHTR14A	Register 14 address space ID
10	QWHTR14	Contents of register 14
14	QWHTR15	Contents of register 15
18	QWHTR0	Contents of register 0
1C	QWHTR1	Contents of register 1
20	QWHTEXU	Address of MVS execution unit
24	QWHTDIM	Number of data items
26	QWHTHASI	Home address space ID
28	QWHTDATA	Address of the data
2C	QWHTFLAG	Flags in the trace list
2E	QWHTDATL	Length of the data list
30	QWHTEND	End of header

Table 300 shows the contents of the data sharing header.

Table 300. Contents of data sharing header

Hex Offset	Macro DSNDQWHA field	Description
0	QWHALEN	Length of the data sharing header
2	QWHATYP	Header type
3		Reserved

Table 300. Contents of data sharing header (continued)

Hex Offset	Macro DSNDQWHA field	Description
4	QWHAMEMN	DB2 member name
C	QWHADSGN	DB2 data sharing group name
14	QWHAEND	End of header

Figure 160 is a sample accounting trace for a distributed transaction sent to SMF.

```
                                                                                A
000000  07000000 1E650059 A8B50104 019FF3F0  F9F0E5F8 F1C10000 00000000 0000061A
        B  C     D        E  F     G                   H                 I
000020  00E60001 00000084 01F00001 000003CE  01CC0001 0000059A 00400002 00000376
                 J                 K                   L                 M
000040  00580001 00000000 00000000 00000274  01020001 00000000 00000000 00000000
                 N
000060  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
000080  00000000 BAA5E4D7 8373558E BAA5E4E2  B6DB4F46 00000000 0349EA39 00000000
0000A0  05578AFD 00000000 00000000 00000000  00000000 0000000C 40404040 40404040
0000C0  00000000 00000000 00000001 00000002  00000000 006349C0 00000000 005735A0
0000E0  00000000 00000000 00000000 00000000  00000000 00000000 00000006 00000000
000100  00000000 00000000 00000000 00000000  00000000 005536E0 00000000 00000000
000120  00000000 00000000 00000000 00000006  00000000 00000000 00000000 00000000
                          O
000140  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
000160  00000000 00030004 00000000 00000000  00000000 00000000 00000000 00000000
000180  00000000 03BDA0A1 00000000 0131DD20  0000000C 00000000 0C90ADB5 00000006
0001A0  00000000 00000000 00001816 00000000  00000000 00000000 00000000 00000000
0001C0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
0001E0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
000200  00432BFA 00000000 00010F80 00000009  7C68DEB2 00000000 033B2BAB 00000000
                 P
000220  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
000240  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
000260  00000000 00000000 00000000 00000000  00000000 5FC4E2D5 F0F8F0F1 F5E2E3D3
000280  C5C3F140 40404040 40404040 40E4E2C9  C2D4E2E8 40E2E8C5 C3F1C4C2 F2C2C1E3
0002A0  C3C84040 40C2C1E3 C3C84040 40E6D5C5  E2E3D5D7 40404040 40E2E8E2 C1C4D440
0002C0  40D7D3D5 C1D7D7D3 F8E4E2C5 D97EE2E8  E2C1C4D4 00000000 00000000 00000000
0002E0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
000300  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
                 Q
000320  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
000340  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
000360  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
000380  00000000 00000000 00000000 00010000  00000000 00000000 00000000 00000000
0003A0  00000000 00000000 00000000 00170000  00100000 00000000 00040000 00000000
0003C0  00060000 00000000 00000000 00002095  01CCD8E7 E2E30000 00020000 00000000
0003E0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
                                                     R
000400  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
000420  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
```

Figure 160. DB2 distributed data trace output sent to SMF (printed with IMS DFSERA10 print program) (Part 1 of 2). In this example there is one accounting record (IFCID 0003) from the server site (SILICON_VALLEY_LAB). DSNDQWA0 maps the self-defining section for IFCID 0003.

```
000440  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
                                  S
000460  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
000480  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
0004A0  00000000 00000000 00000000 00000000  00000000 00000000 00030000 00000000
0004C0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
0004E0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
000500  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
000520  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
                                            T
000540  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
000560  00000000 00000000 00030000 00000000  00000000 00000000 00000000 00000000
                                            U
000580  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
0005A0  00080000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
                                                                      V
0005C0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00640000
0005E0  000A0000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
000600  00000000 00000000 00000000 00000000  00000000 00000000 00000052 011A0003
000620  0D8106E0 D7B8E5F8 F1C1BAA5 E4E2B6E5  85A60000 00180000 00230000 0004E2E3
000640  D3C5C3F1 40404040 40404040 4040E4E2  C9C2D4E2 E840E2E8 C5C3F1C4 C2F2BAA5
                                W
000660  E4D78307 00010000 00000000 00940200  E2E8E2C1 C4D44040 E6D5C5E2 E3D5D740
000680  40404040 C2C1E3C3 C8404040 D7D3D5C1  D7D7D3F8 E2E8E2C1 C4D44040 00000001
0006A0  00000000 00000000 00000000 00000000  00000000 00000000 40404040 40404040
0006C0  40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040
```

Figure 160. DB2 distributed data trace output sent to SMF (printed with IMS DFSERA10 print program) (Part 2 of 2). In this example there is one accounting record (IFCID 0003) from the server site (SILICON_VALLEY_LAB). DSNDQWA0 maps the self-defining section for IFCID 0003.

Key to Figure 160 on page 1148	Description
A 0000061A	Offset to product section; beginning of self-defining section
B 00E6	Length of product section
C 0001	Number of times product section is repeated
D 00000084	Offset to accounting section
E 01F0	Length of accounting section
F 0001	Number of times accounting section is repeated
G 000003CE	Offset to SQL accounting section
H 0000059A	Offset to buffer manager accounting section
I 00000376	Offset to locking accounting section
J 00000000	Offset to distributed section
K 00000274	Offset to MVS/DDF accounting section
L 00000000	Offset to IFI accounting section
M 00000000	Offset to package/DBRM accounting section
N 00000000...	Beginning of accounting section (DSNDQWAC)
O 00000000...	Beginning of distributed section (DSNDQLAC)
P 00000000...	Beginning of MVS/DDF accounting section (DSNDQMDA)
Q 00000000...	Beginning of package/DBRM accounting section (DSNDQPAC)
R 00000000...	Beginning of locking accounting section (DSNDQTXA)
S 00000000...	Beginning of SQL accounting section (DSNDQXST)
T 00000000...	Beginning of buffer manager accounting section (DSNDQBAC)
U 00000000...	Beginning of product section (DSNDQWHS); beginning of standard header
V 00640000...	Beginning of correlation header (DSNDQWHC)
W 00000094...	Beginning of distributed header (DSNDQWHD)

Trace field descriptions

If you intend to write a program to read DB2 trace records, use the assembler mapping macros in *prefix*.SDSNMACS.

You can use the TSO or ISPF browse function to look at the field descriptions in the trace record mapping macros online, even when DB2 is down. If you prefer to look at the descriptions in printed form, you can use ISPF to print a listing of the data set.

Appendix E. Programming for the Instrumentation Facility Interface (IFI)

The information in this appendix is Product-sensitive Programming Interface and Associated Guidance Information, as defined in “Notices” on page 1301.

The DB2 instrumentation facility gathers trace data that can be written to one or more destinations that you specify. The instrumentation facility interface (IFI) is designed for a program needing *online* trace information. IFI can be accessed through any of the DB2 attachment facilities.

IFI uses the standard security mechanisms that DB2 uses: connection authorization, plan authorization, and so forth. For more information about security, see Part 3, “Security and auditing,” on page 123. Security checks specifically related to IFI are included in the descriptions of the functions.

Before using IFI, you should be familiar with the material in “DB2 trace” on page 1189, which includes information on the DB2 trace facility and instrumentation facility component identifiers (IFCIDs).

Note that where the trace output indicates a particular release level, you will see 'xx' to show that this information varies according to the actual release of DB2 that you are using.

You can use IFI in a monitor program (a program or function outside of DB2 that receives information about DB2) to perform the following tasks:
- “Submitting DB2 commands through IFI”
- “Obtaining trace data through IFI” on page 1152
- “Passing data to DB2 through IFI” on page 1152

When a DB2 trace is active, internal events trigger the creation of trace records. The records, identified by *instrumentation facility component identifiers* (IFCIDs), can be written to buffers, and you can read them later with the IFI READA function. This means you are collecting the data *asynchronously*; you are not reading the data at the time it was written.

You can trigger the creation of certain types of trace records by using the IFI READS function. The records, identified as usual by IFCIDs, do not need a buffer; they are passed immediately to your monitor program through IFI. This means you are collecting the data *synchronously*. The data is collected at the time of the request for the data.

Submitting DB2 commands through IFI

You can submit any DB2 command through IFI, but this capability is most useful for submitting DB2 trace commands to start, stop, display, and modify traces.

Using specified trace classes and IFCIDs, a monitor program can control the amount and type of its data. You can design your monitor program to:
- Activate and deactivate pre-defined trace classes.
- Activate and deactivate a trace record or group of records (identified by IFCIDs).

© Copyright IBM Corp. 1982, 2008

- Activate and deactivate predefined trace classes and trace records (identified by IFCIDs) restricting tracing to a set of DB2 identifiers (plan name, authorization ID, resource manager identifier (RMID), and so on).

Obtaining trace data through IFI

You might want to collect trace data from DB2:

- To obtain accounting information for online billing.
- To periodically obtain system-wide information about DB2, highlight exceptional conditions, or provide throughput information.

  The following illustrates the logic flow:

  1. Initialize.
  2. Set a timer.
  3. Wait for the timer to expire.
  4. Call IFI to obtain statistics data via a READS request.
  5. Do delta calculations to determine activity.

     This step is not necessary for IFCID 0199 because DB2 resets statistics at the beginning of every collection interval.
  6. Display the information on a terminal.
  7. Loop back to the timer.
- To learn which processes have been connected to DB2 the longest, or which processes have used the most CPU time in DB2.
- To obtain accounting records as transactions terminate.
- To determine the access and processing methods for an SQL statement. Start a trace, issue a PREPARE statement, and then use the resulting trace data as an alternative to using EXPLAIN.
- To capture log buffers online for use in remote recovery, as described in Appendix C, “Reading log records,” on page 1111.
- To retrieve SQL changes synchronously from the log for processing in an application. See “Reading log records with IFI” on page 1122 for more information.

Passing data to DB2 through IFI

You can use IFI to pass data to the destination of a DB2 trace. For example, you can:

- Extend accounting data collected within DB2. For example, a monitor program can collect batch file I/O counts, store them in a user-defined trace record, and process them along with standard DB2 accounting data.
- Include accounting data from QMF, IMS, or CICS.
- Permit CICS users to write the CICS accounting token and task number into the DB2 trace, assuming ACCOUNTREC in the DB2ENTRY RDO definition is neither UOW nor TASK .

IFI functions

A monitor program can use the following IFI functions:

COMMAND To submit DB2 commands. For more information, see “COMMAND: Syntax and usage with IFI” on page 1154.

READS To obtain monitor trace records synchronously. The READS request

causes those records to be returned immediately to the monitor program. For more information, see "READS: Syntax and usage with IFI" on page 1156.

READA To obtain trace records of any trace type asynchronously. DB2 records trace events as they occur and places that information into a buffer; a READA request moves the buffered data to the monitor program. For more information, see "READA: Syntax and usage with IFI" on page 1171.

WRITE To write information to a DB2 trace destination that was previously activated by a START TRACE command. For more information, see "WRITE: Syntax and usage with IFI" on page 1173.

Invoking IFI from your program

IFI can be used by assembler and PL/I programs. To use IFI, include a call to DSNWLI in your monitor program.

The following example depicts an IFI call in an assembler program. All examples in this appendix are given for assembler.

```
CALL DSNWLI,(function,ifca,parm-1,...parm-n),VL
```

The parameters that are passed on the call indicate the desired function (as described in "IFI functions" on page 1152), point to communication areas used by the function, and provide other information that depends on the function specified. Because the parameter list may vary in length, the high-order bit of the last parameter must be on to signal that it is the last parameter in the list.

Example: To turn on the bit in assembler, use the VL option to signal a variable length parameter list.

The communication areas that are used by IFI are described in "Common communication areas for IFI calls" on page 1174.

After you insert this call in your monitor program, you must link-edit the program with the correct language interface. Each of the following language interface modules has an entry point of DSNWLI for IFI:

- CAF DSNALI
- TSO DSNELI
- CICS DSNCLI
- IMS DFSLI000
- RRSAF DSNRLI

CAF DSNALI, the CAF (call attachment facility) language interface module, includes a second entry point of DSNWLI2. The monitor program that link-edits DSNALI with the program can make IFI calls directly to DSNWLI. The monitor program that loads DSNALI must also load DSNWLI2 and remember its address. When the monitor program calls DSNWLI, the program must have a dummy entry point to handle the call to DSNWLI and then call the real DSNWLI2 routine. See Part 6 of *DB2 Application Programming and SQL Guide* for additional information about using CAF.

Considerations for writing a monitor program: A monitor program issuing IFI requests must be connected to DB2 at the thread level. If the program contains SQL statements, you must precompile the program and create a DB2 plan using the BIND process. If the monitor program does not contain any SQL statements, it

does not have to be precompiled. However, as is the case in all the attachment environments, even though an IFI only program (one with no SQL statements) does not have a plan of its own, it can use any plan to get the thread level connection to DB2.

The monitor program can run in either 24- or 31-bit mode.

Monitor trace classes: Monitor trace classes 2 through 8 can be used to collect information related to DB2 resource usage. Use monitor trace class 5, for example, to find out how much time is spent processing IFI requests. Monitor trace classes 2, 3, and 5 are identical to accounting trace classes 2, 3, and 5. For more information about these traces, see "Monitor trace" on page 1192.

Monitor authorization: On the first READA or READS call from a user, an authorization is checked to determine if the primary authorization ID or one of the secondary authorization IDs of the plan executor has MONITOR1 or MONITOR2 privilege. If your installation uses the access control authorization exit routine, that exit routine might control the privileges that can use the monitor trace. If you have an authorization failure, an audit trace (class 1) record is generated that contains the return and reason codes from the exit. This is included in IFCID 0140. See "Access control authorization exit routine" on page 1061for more information on the access control authorization exit routine.

Using IFI from stored procedures

You can use the IFI interface from a stored procedure. The output of the trace can be returned to the client. You can also issue DB2 commands, such as "DISPLAY THREAD", from a stored procedure and return the results to the client.

COMMAND: Syntax and usage with IFI

A DB2 command resides in the output area; a monitor program can submit that command by issuing a COMMAND request to IFI. The DB2 command is processed and the output messages are returned to the monitor program in the return area.

You can submit any DB2 command, including START TRACE, STOP TRACE, DISPLAY TRACE, and MODIFY TRACE. Because the program can also issue other DB2 commands, you should be careful about which commands you use. For example, do **not** use STOP DB2.

Authorization for DB2 commands through IFI

For an application program to submit a command, the primary authorization ID or one of the secondary authorization IDs of the process must have the appropriate DB2 command authorization. Otherwise the request is denied. An application program might have the authorization to issue DB2 commands, but not the authorization to issue READA requests. For more information about authorization, see Part 3, "Security and auditing," on page 123.

Syntax for DB2 commands through IFI

DB2 commands that are issued through IFI have the following syntax:

```
CALL DSNWLI,('COMMAND ',ifca,return-area,output-area,buffer-info .),VL
```

ifca

IFCA (instrumentation facility communication area) is an area of storage that contains the return code and reason code. IFCA indicates the following information:

- The success or failure of the request
- Diagnostic information from the DB2 component that executed the command
- The number of bytes moved to the return area
- The number of bytes of the message segments that did not fit in the return area

Some commands might return valid information despite a non-zero return code or reason code. For example, the DISPLAY DATABASE command might indicate that more information could have been returned than was allowed.

If multiple errors occur, the last error is returned to the caller. For example, if the command is in error and the error message does not fit in the area, the error return code and reason code indicate that the return area is too small.

If a monitor program issues START TRACE, the ownership token (IFCAOWNR) in the IFCA determines the owner of the asynchronous buffer. The owner of the buffer is the only process that can obtain data through a subsequent READA request. See "Instrument facility communications area (IFCA)" on page 1174 for a description of the IFCA.

return-area

When the issued command finishes processing, it places messages (if any) in the return area. The messages are stored as varying-length records, and the total number of bytes in the records is placed in the IFCABM (bytes moved) field of the IFCA. If the return area is too small, as many message records as will fit are placed into the return area.

The monitor program should analyze messages that are returned by the command function. See "Return area" on page 1178 for a description of the return area.

output-area

Contains the varying-length command. See "Output area" on page 1179 for a description of the output area.

buffer-info

This parameter is required for starting traces to an OP buffer. Otherwise, it is not needed. This parameter is used only on COMMAND requests. It points to an area that contains information about processing options when a trace is started by an IFI call to an unassigned OP*n* destination buffer. An OP*n* destination buffer is considered unassigned if it is not owned by a monitor program.

If the OP*n* destination buffer is assigned, then the buffer information area is not used on a later START or MODIFY TRACE command to that OP*n* destination. For more information about using OP*n* buffers, see "Usage notes for READA requests through IFI" on page 1171.

When you use *buffer-info* on START TRACE, you can specify the number of bytes that can be buffered before the monitor program ECB is posted. The ECB is posted when the amount of trace data collected has reached the value that is specified in the byte count field. The byte count field is also specified in the buffer information area.

Table 301 on page 1156 summarizes the fields in the buffer information area.

Table 301. Buffer information area fields. This area is mapped by assembler mapping macro DSNDWBUF.

Name	Hex offset	Data type	Description
WBUFLEN	0	Signed two-byte integer	Length of the buffer information area, plus 4. A zero indicates the area does not exist.
	2	Signed two-byte integer	Reserved.
WBUFEYE	4	Character, 4 bytes	Eye catcher for block, WBUF.
WBUFECB	8	Address	The ECB address to post when the buffer has reached the byte count specification (WBUFBC). The ECB must reside in monitor key storage. A zero indicates not to post the monitor program. In this case, the monitor program should use its own timer to determine when to issue a READA request.
WBUFBC	C	Signed four-byte integer	The records placed into the instrumentation facility must reach this value before the ECB will be posted. If the number is zero, and an ECB exists, posting occurs when the buffer is full.

Example of DB2 command through IFI

This example issues a DB2 START TRACE command for MONITOR Class 1.

```
CALL DSNWLI,('COMMAND ',IFCAAREA,RETAREA,OUTAREA,BUFAREA),VL
   ⋮
COMMAND DC CL8 'COMMAND '
************************************************************************
* Function parameter declaration                                       *
************************************************************************
* Storage of LENGTH(IFCA) and properly initialized                     *
************************************************************************
IFCAAREA  DS       0CL180
   ⋮
************************************************************************
* Storage for length and returned info.                                *
************************************************************************
RETAREA   DS      CL608
************************************************************************
* Storage for length and DB2 Command                                   *
************************************************************************
OUTAREA   DS      0CL42
OUTLEN    DC      X'002A0000'
OUTCMD    DC      CL37'-STA TRAC(MON) DEST(OPX) BUFSIZE(256)'
************************************************************************
* Storage of LENGTH(WBUF) and properly initialized                     *
************************************************************************
BUFAREA  DS       0CL16
   ⋮
```

Figure 161. Starting a trace using IFI

READS: Syntax and usage with IFI

READS allows your monitor program to read DB2 status information that is collected at the time of the IFI call. Monitor class 1 must be activated prior to any READS requests. The records available are for IFCIDs 0001, 0002, 0106, 0124, 0129, 0147, 0148, 0149, 0150, 0185, 0199, 0202, 0230, 0234, 0254, 0306, 0316, and 0317. For a description of the data that these records provide, see “Synchronous data and

READS requests through IFI" on page 1167. IFCID 0124, 0129, 0147, 148, 149, 0150, 0199, 234, 0254, 0316, and 0317 can be obtained only through the IFI READS interface.

The program that issues the READS request does not need to start monitor class 1 because no ownership of an OP buffer is involved when you obtain data from the READS interface. Data is written directly to the application program's return area, bypassing the OP buffer. This bypass is in direct contrast to the READA interface where the application that issues READA must first issue a START TRACE command to obtain ownership of an OP buffer and start the appropriate traces.

Authorization for READS requests through IFI

On a READS request, a check is made to see if monitor class 1 is active; if it is not active, the request is denied. The primary authorization ID or one of the secondary authorization IDs of the process running the application program must have MONITOR1 or MONITOR2 privilege. If neither the primary authorization ID nor one of the secondary authorization IDs has authorization, the request is denied.

Exception: IFCID 185 requests do not require the MONITOR1 or MONITOR2 privilege.

READS requests are checked for authorization once for each user (ownership token) of the thread. (Several users can use the same thread, but an authorization check is performed each time the user of the thread changes.)

If you use READS to obtain your own data (IFCID 0124, 0147, 0148, or 0150 not qualified), no authorization check is performed.

Syntax for READS requests through IFI

DB2 commands that are issued through IFI have the following syntax:

```
CALL DSNWLI,('READS   ',ifca,return-area,ifcid-area,qual-area),VL
```

ifca
: Contains information about the success of the call. See "Instrument facility communications area (IFCA)" on page 1174 for a description of the IFCA.

return-area
: Contains the varying-length records that are returned by the instrumentation facility. IFI monitor programs might require large enough READS return areas to accommodate the following information:

- Larger IFCID 0147 and 0148 records that contain distributed thread data (both allied and database access).
- Additional records that are returned when database access threads exist that satisfy the specified qualifications on the READS request.
- Log record control intervals with IFCID 129. For more information about using IFI to return log records, see "Reading specific log records (IFCID 0129)" on page 1122.
- Log records based on user-specified criteria with IFCID 306. For example, the user can retrieve compressed or decompressed log records. For more information about reading log records, see Appendix C, "Reading log records," on page 1111.
- Data descriptions and changed data returned with IFCID 185.

If the return area is too small to hold all the records returned, it contains as many records as will fit. The monitor program obtains the return area for READS requests in its private address space. See "Return area" on page 1178 for a description of the return area.

ifcid-area
Contains the IFCIDs of the desired information. The number of IFCIDs can be variable. If the length specification of the IFCID area is exceeded or an IFCID of X'FFFF' is encountered, the list is terminated. If an invalid IFCID is specified no data is retrieved. See "IFCID area" on page 1179 for a description of the IFCID area.

qual-area
This parameter is optional, and is used only on READS requests. It points to the qualification area, where a monitor program can specify constraints on the data that is to be returned. If the qualification area does not exist (length of binary zero), information is obtained from all active allied threads and database access threads. Information is not obtained for any inactive database access threads that might exist.

The length constants for the qualification area are provided in the DSNDWQAL mapping macro. If the length is not equal to the value of one of these constants, IFI considers the call invalid.

The following trace records, identified by IFCID, cannot be qualified: 0001, 0002, 0106, 0202, 217, 225, 0230. If you attempt to qualify them, the qualification is ignored.

The rest of the synchronous records can be qualified. See "Synchronous data and READS requests through IFI" on page 1167 for information about these records. However, not all the qualifications in the qualification area can be used for these IFCIDs. See "Which qualifications are used for READS requests issued through IFI?" on page 1166 for qualification restrictions. Unless the qualification area has a length of binary zero (in which case the area does not exist), the address of *qual-area* supplied by the monitor program points to an area that is formatted by the monitor program, as shown in Table 302 on page 1159.

Table 302. Qualification area fields. This area is mapped by the assembler mapping macro DSNDWQAL.

Name	Hex offset	Data type	Description
WQALLEN	0	Signed two-byte integer	Length of the qualification area, plus 4. The following constants set the qualification area length field: **WQALLN4** When specified, the location name qualifications (WQALLOCN and WQALLUWI), the group buffer pool qualifier (WQALGBPN) and the read log fields are used. **WQALLN5** When specified, the dynamic statement cache fields (WQALFFLD, WQALFVAL, WQALSTNM, and WQALSTID) are used for READS calls for IFCID 0316 and 0317. **WQALLN6** When specified, the end-user identification fields (WQALEUID, WQALEUTX, and WQALEUWS) are used for READS calls for IFCID 0124, 0147, 0148, 0149, and 0150. **WQALLN21** When specified, the location name qualifications (WQALLOCN and WQALLUWI) are ignored. **WQALLN22** When specified, the location name qualifications (WQALLOCN and WQALLUWI) are used. **WQALLN23** When specified, the log data access fields (WQALLTYP, WQALLMOD, WQALLRBA, and WQALLNUM) are used for READS calls that use IFCID 129.
	2	Signed two-byte integer	Reserved.
WQALEYE	4	Character, 4 bytes	Eye catcher for block, WQAL.
WQALACE	8	Address	Thread identification token value. This value indicates the specific thread wanted; binary zero if it is not to be used.
WQALAIT2	C	Address	Reserved.
WQALPLAN	10	Character, 8 bytes	Plan name; binary zero if it is not to be used.
WQALAUTH	18	Character, 8 bytes	The current primary authorization ID; binary zero if it is not to be used.
WQALOPID	20	Character, 8 bytes	The original authorization ID; binary zero if it is not to be used.
WQALCONN	28	Character, 8 bytes	Connection name; binary zero if it is not to be used.
WQALCORR	30	Character, 12 bytes	Correlation ID; binary zero if it is not to be used.
WQALREST	3C	Character, 32 bytes	Resource token for a specific lock request when IFCID 0149 is specified. The field must be set by the monitor program. The monitor program can obtain the information from a previous READS request for IFCID 0150 or from a READS request for IFCID 0147 or 0148.
WQALHASH	5C	Hex, 4 bytes	Resource hash value that specifies the resource token for a specific lock request when IFCID 0149 is specified. The field must be set by the monitor program. The monitor program can obtain the information from a previous READS request for IFCID 0150 or possibly from a READS request for IFCID 0147 or 0148.
WQALASID	60	Hex, 2 bytes	ASID that specifies the address space of the desired process.

Table 302. Qualification area fields (continued). This area is mapped by the assembler mapping macro DSNDWQAL.

Name	Hex offset	Data type	Description
WQALFOPT	62	Hex, 1 byte	Filtering options for IFCID 0150: **X'20'** Return lock information for resources that have local or global waiters. **X'40'** Return lock information only for resources that have one or more interested agents. **X'80'** Return lock information only for resources that have waiters.
WQALFLGS	63	Hex, 1 byte	Options for 147/148 records: **X'40'** Active allied agent 147/148 records are **not** written for this READS request. DDF/RRSFA rollup records are written if WQAL148NR = OFF. **X'80'** DDF/RRSAF rollup 147/148 records are **not** written for this READS request. Active allied agent records are written if WQAL148NA = OFF.[1]
	64	Character, 24 bytes	LUWID (logical unit of work ID) of the thread wanted; binary zero if it is not to be used
	7C	Character, 16 bytes	Location name. If specified, data is returned only for distributed agents that originate at the specified location. **Example:** If site A is located where the IFI program is running and SITE A is specified in the WQALLOCN, database access threads and distributed allied agents that execute at SITE A are reported. Local non-distributed agents are not reported. **Example:** If site B is specified in the WQALLOCN and the IFI program is still executing at site A, information on database access threads that execute in support of a distributed allied agent at site B are reported. **Example:** If WQALLOCN is not specified, information on all threads that execute at SITE A (the site where the IFI program executes) is returned. This includes local non-distributed threads, local database access agents, and local distributed allied agents.
WQALLTYP	8C	Character, 3 bytes	Specifies the type of log data access. 'CI ' must be specified to obtain log record control intervals (CIs).

Table 302. Qualification area fields (continued). This area is mapped by the assembler mapping macro DSNDWQAL.

Name	Hex offset	Data type	Description
WQALLMOD	8F	Character, 1 byte	The mode of log data access: **'D'** Return the direct log record specified in WQALLRBA if the IFCID is 0306. **'F'** Access the first log CI of the restarted DB2 system if the IFCID is 0129. One CI is returned, and the WQALLNUM and WQALLRBA fields are ignored. It indicates to return the first set of qualified log records if the IFCID is 0306. **'H'** Return the highest LRSN or log RBA in the active log. The value is returned in the field IFCAHLRS in the IFCA. **'N'** Return the next set of qualified log records. **'P'** the last partial CI written to the active log is given to the Log Capture Exit. If the last CI written to the log was not full, the RBA of the log CI given to the Log Exit is returned in the IFCAHLRS field of the IFI communication area (IFCA). Otherwise, an RBA of zero is returned in IFCAHLRS. This option ignores WQALLRBA and WQALLNUM. **'R'** Access the CIs specified by the value in the WQALLRBA field: • If the requested number of complete CIs (as specified in WQALLNUM) are currently available, those CIs are returned. If fewer than the requested number of complete CIs are available, IFI returns as many complete CIs as are available. • If the WQALLRBA value is beyond the end of the active log, IFI returns a return code of X'0000000C' and a reason code of X'00E60855'. No records are returned. • If no complete CIs exist beyond the WQALLRBA value, IFI returns a return code of X'0000000C' and a reason code of X'00E60856'. No records are returned. **'T'** Terminate the log position that is held to anticipate a future mode 'N' call.
WQALLNUM	90	Hex, 2 bytes	The number of log CIs to be returned. The valid range is X'0001' to X'0007'.
WQALCDCD	92	Character, 1 byte	Data description request flag: **'A'** Indicates that a data description will only be returned the first time a DATA request is issued from the region or when it was changed for a given table. This is the default. **'N'** Indicates that a data description is not returned. **'Y'** Indicates that a data description will be returned for each table in the list for every new request.
	93	Hex, 1 byte	Reserved.
WQALLRBA	94	Hex, 8 bytes	If the IFCID is 0129, this is the starting log RBA of the CI to be returned. The CI starting log RBA value must end in X'000'. The RBA value must be right-justified. If the IFCID is 0306, this is the log RBA or LRSN to be used in mode 'F'.

Table 302. Qualification area fields (continued). This area is mapped by the assembler mapping macro DSNDWQAL.

Name	Hex offset	Data type	Description
WQALGBPN	9C	Character, 8 bytes	Group buffer pool name for IFCID 0254. Buffer pool name for IFCID 0199. To specify a single buffer pool or group buffer pool, specify the buffer pool name in hexadecimal, followed by hexadecimal blanks. **Example:** To specify buffer pool BP1, put X'C2D7F14040404040' in this field. To specify more than one buffer pool or group buffer pool, use the pattern-matching character X'00' in any position in the buffer pool name. X'00' indicates that any character can appear in that position, and in all positions that follow. **Example:** If you put X'C2D7F10000000000' in this field, you indicate that you want data for all buffer pools whose names begin with BP1, so IFI collects data for BP1, BP10 through BP19, and BP16K0 through BP16K9. **Example:** If you put X'C2D700F100000000' in this field, you indicate that you want data for all buffer pools whose names begin with BP, so IFI collects data for all buffer pools. IFI ignores X'F1' in position four because it occurs after the first X'00'.
WQALLCRI	A4	Hex, 1 byte	Log Record Selection Criteria: **'00'** Indicates the return DB2CDC and UR control log records.
WQALLOPT	A5	Hex, 1 byte	Processing Options relating to decompression: **'00'** Indicates that decompression should not occur. **'01'** Indicates to decompress the log records if they are compressed.

Table 302. Qualification area fields (continued). This area is mapped by the assembler mapping macro DSNDWQAL.

Name	Hex offset	Data type	Description
WQALFLTR	A6	Hex, 1 byte	For an IFCID 0316 request, WQALFLTR identifies the filter method: **X'00'** Indicates no filtering. This value tells DB2 to return information for as many cached statements as fit in the return area. **X'01'** Indicates that DB2 returns information about the cached statements that have the highest values for a particular statistics field. The statistics field is specified in WQALFFLD. DB2 returns information for as many statements as fit in the return area. **Example:** If the return is large enough for information about 10 statements, the statements with the ten highest values for the specified statistics field are reported. **X'02'** Indicates that DB2 returns information about the cached statements that exceed a threshold value for a particular statistics field. The name of the statistics field is specified in WQALFFLD. The threshold value is specified in WQALFVAL. DB2 returns information for as many qualifying statements as fit in the return area. **X'04'** Indicates that DB2 returns information about a single cached statement. The application provides the four-byte cached statement identifier in field WQALSTID. An IFCID 0316 request with this qualifier is intended for use with IFCID 0172 or IFCID 0196, to obtain information about the statements that are involved in a timeout or deadlock. For an IFCID 0317 request, WQALFLTR identifies the filter method: **X'04'** Indicates that DB2 returns information about a single cached statement. The application provides the four-byte cached statement identifier in field WQALSTID. An IFCID 0317 request with this qualifier is intended for use with IFCID 0172 or IFCID 0196, to obtain information about the statements that are involved in a timeout or deadlock. For an IFCID 0306 request, WQALFLTR indicates whether DB2 merges log records in a data sharing environment: **X'00'** Indicates that DB2 merges log records from data sharing members. **X'03'** Indicates that DB2 does not merge log records from data sharing members.

Table 302. Qualification area fields (continued). This area is mapped by the assembler mapping macro DSNDWQAL.

Name	Hex offset	Data type	Description
WQALFFLD	A7	Character, 1 byte	For an IFCID 0316 request, when WQALFLTR is X'01' or X'02', this field specifies the statistics field that is used to determine the cached statements about which DB2 reports. You can enter the following values: **'A'** The accumulated elapsed time (QW0316AE). This option is valid only when QWALFLTR=X'01'. **'B'** The number of buffer reads (QW0316NB). **'C'** The accumulated CPU time (QW0316CT). This option is valid only when QWALFLTR=X'01'. **'E'** The number of executions of the statement (QW0316NE). **'G'** The number of GETPAGE requests (QW0316NB). **'I'** The number of index scans (QW0316NI). **'L'** The number of parallel groups (QW0316NL). **'P'** The number of rows processed (QW0316NP). **'R'** The number of rows examined (QW0316NR). **'S'** The number of sorts performed (QW0316NS). **'T'** The number of table space scans (QW0316NT). **'W'** The number of buffer writes (QW0316NW). **'X'** The number of times that a RID list was not used because the number of RIDs would have exceeded one or more internal DB2 limits (QW0316RT). **'Y'** The number of times that a RID list was not used because not enough storage was available (QW0316RS). **'1'** The accumulated wait time for synchronous I/O (QW0316W1). This option is valid only when QWALFLTR=X'01'. **'2'** The accumulated wait time for lock and latch requests (QW0316W2). This option is valid only when QWALFLTR=X'01'. **'3'** The accumulated wait time for a synchronous execution unit switch (QW0316W3). This option is valid only when QWALFLTR=X'01'. **'4'** The accumulated wait time for global locks (QW0316W4). This option is valid only when QWALFLTR=X'01'. **'5'** The accumulated wait time for read activity by another thread (QW0316W5). This option is valid only when QWALFLTR=X'01'. **'6'** The accumulated wait time for write activity by another thread (QW0316W6). This option is valid only when QWALFLTR=X'01'.

Table 302. Qualification area fields (continued). This area is mapped by the assembler mapping macro DSNDWQAL.

Name	Hex offset	Data type	Description
WQALFVAL	A8	Signed 4-byte integer	For an IFCID 0316 request, when WQALFLTR is X'02', this field and WQALFFLD determine the cached statements about which DB2 reports. To be eligible for reporting, a cached statement must have a value for WQALFFLD that is no smaller than the value that you specify in WQALFVAL. DB2 reports information on as many eligible statements as fit in the return area.
WQALSTNM	AC	Character, 16 bytes	For an IFCID 0317 request, when WQALFLTR is not X'04', this field specifies the name of a cached statement about which DB2 reports. This is a name that DB2 generates when it caches the statement. To obtain this name, issue a READS request for IFCID 0316. The name is in field QW0316NM. This field and WQALSTID uniquely identify a cached statement.
WQALSTID	BC	Unsigned 4-byte integer	For an IFCID 0316 or IFCID 0317 request, this field specifies the ID of a cached statement about which DB2 reports. DB2 generates this ID when it caches the statement. To obtain the ID, use the following options: • For an IFCID 0317 request, when WQALFLTR is not X'04', obtain this ID by issuing a READS request for IFCID 0316. The ID is in field QW0316TK. This field and WQALSTNM uniquely identify a cached statement. • For an IFCID 0316 or IFCID 0317 request, when WQALFLTR is X'04', obtain this ID by issuing a READS request for IFCID 0172 or IFCID 0196. The ID is in field QW0172H9 (cached statement ID for the holder in a deadlock), QW0172W9 (cached statement ID for the waiter in a deadlock), or QW0196H9 (cached statement ID of the holder in a timeout). This field uniquely identifies a cached statement.
WQALEUID	C0	Character, 16 bytes	The end user's workstation user ID. This value can be different from the authorization ID that is used to connect to DB2. This field contains binary zeroes if the client does not supply this information.
WQALEUTX	D0	Character, 32 bytes	The name of the transaction or application that the end user is running. This value identifies the application that is currently running, not the product that is used to run the application. This field contains binary zeroes if the client does not supply this information.
WQALEUWS	F0	Character, 18 bytes	The end user's workstation name. This value can be different from the authorization ID used to connect to DB2. This field contains binary zeroes if the client does not supply this information.

Note: 1. The only valid filters for DDF/RRSAF 147/148 rollup records are WQALEUID, WQALEUTX, and
WQALEUWN. For a 147/148 request, DDF/RRSAF records are not processed if any of the following WQAL fields
are not X'00':
• WQALACE
• WQALAUTH
• WQALOPID
• WQALPLAN
• WQALCORR
• WQALLUWI
• WQALLOCN
• WQALASID
• WQALCONN

Important: If your monitor program does not initialize the qualification area, the READS request is denied.

Which qualifications are used for READS requests issued through IFI?

Not all qualifications are used for all IFCIDs. Table 303 lists the qualification fields that are used for each IFCID.

Table 303. Qualification fields for IFCIDs

These IFCIDs...	Are allowed to use these qualification fields
0124, 0147, 0148, 0150	WQALACE WQALAIT2 WQALPLAN[1] WQALAUTH[1] WQALOPID[1] WQALCONN[1] WQALCORR[1] WQALASID WQALLUWI[1] WQALLOCN[1] WQALEUID WQALEUTX WQALEUWS
0129	WQALLTYP WQALLMOD WQALLRBA WQALLNUM
0149	WQALREST WQALHASH
0150	WQALFOPT
0185	WQALCDCD
0199, 0254	WQALGBPN[2]
0306	WQALFLTR WQALLMOD WQALLRBA WQALLCRI WQALLOPT
0316	WQALFLTR WQALFFLD WQALFVAL WQALSTID
0317	WQALFLTR WQALSTNM WQALSTID

Table 303. Qualification fields for IFCIDs (continued)

These IFCIDs...	Are allowed to use these qualification fields

Note:

1. DB2 allows you to partially qualify a field and fill the rest of the field with binary zero. For example, the 12-byte correlation value for a CICS thread contains the 4-character CICS transaction code in positions 5-8. Assuming a CICS transaction code of AAAA, the following hexadecimal *qual-area* correlation qualification can be used to find the first transaction with a correlation value of AAAA in positions 5-8: X'00000000C1C1C1C100000000'.
2. X'00' in this field indicates a pattern-matching character. X'00' in any position of the field indicates that IFI collects data for buffer pools whose names contain any character in that position and all following positions.

Usage notes for READS requests through IFI

Due to performance considerations, the majority of data that is obtained by a monitor program probably comes over the synchronous interface. Summarized DB2 information is easier for a monitor program to process, and the monitor program logic is simpler because a smaller number of records are processed.

Start monitor classes 2, 3, 5, 7, and 8 to collect summary and status information for later probing. In this case, an instrumentation facility trace is started and information is summarized by the instrumentation facility, but not returned to the caller until it is requested by a READS call.

The READS request can reference data that is updated during the retrieval process. You might need to do reasonability tests on data that is obtained through READS. Because the READS function does not suspend activity that takes place under referenced structures, an abend can occur. If an abend occurs, the READS function is terminated without a dump and the monitor program is notified through the return code and reason code information in the IFCA. However, the return area can contain valid trace records, even if an abend occurred; therefore, your monitor program should check for a non-zero value in the IFCABM (bytes moved) field of the IFCA.

When you use a READS request with a query parallelism task, remember that each parallel task is a separate thread. Each parallel thread has a separate READS output. See Chapter 35, “Parallel operations and query performance,” on page 987 for more information on tracing the parallel tasks. A READS request might return thread information for parallel tasks on a DB2 data sharing member without the thread information for the originating task in a Sysplex query parallelism case. See *DB2 Data Sharing: Planning and Administration* for more information.

The qual-area parameter, mapped by DSNDWQAL, is the only means of qualifying the trace records to be returned on IFI READS requests.

Synchronous data and READS requests through IFI

There are certain types of records that you can read synchronously. Identified by IFCID, these records are:

0001 Statistical data on the systems services address space, including task control block (TCB) and service request block (SRB) times for system services, database services, including DDF statistics, and Internal Resource Lock Manager (IRLM) address spaces.

0002 Statistical data on the database services address space.

0106 Static system parameters.

0124 An active SQL snapshot that provides status information about:
- The process
- The SQL statement text
- The relational data system input parameter list (RDI) block
- Certain bind and locking information

You can obtain a varying amount of data because the request requires the process to be connected to DB2, have a cursor table allocated (RDI and status information is provided), and be active in DB2 (SQL text is provided if available). The SQL text that is provided does not include the SQL host variables.

For dynamic SQL, IFI provides the original SQL statement. The RDISTYPE field contains the actual SQL function taking place. For example, for a SELECT statement, the RDISTYPE field can indicate that an open cursor, fetch, or other function occurred. For static SQL, you can see the DECLARE CURSOR statement, and the RDISTYPE indicates the function. The RDISTYPE field is mapped by mapping macro DSNXRDI.

0129 Returns one or more VSAM control intervals (CIs) that contain DB2 recovery log records. For more information about using IFI to return these records for use in remote site recovery, see Appendix C, “Reading log records,” on page 1111.

0147 An active thread snapshot that provides a status summary of processes at a DB2 thread or non-thread level.

0148 An active thread snapshot that provides more detailed status of processes at a DB2 thread or non-thread level.

0149 Information that indicates who (the thread identification token) is holding locks and waiting for locks on a particular resource and hash token. The data is in the same format as IFCID 0150.

0150 All the locks held and waited on by a given user or owner (thread identification token).

0185 Data descriptions for each table for which captured data is returned on this DATA request. IFCID 0185 data is only available through a propagation exit routine that is triggered by DB2.

0199 Information about buffer pool usage by DB2 data sets. DB2 reports this information for an interval that you specify in the DATASET STATS TIME field of installation panel DSNTIPN. At the beginning of each interval, DB2 resets these statistics to 0.

0202 Dynamic system parameters.

0217 Storage detail record for the DBM1 address space.

0225 Storage summary record for the DBM1 address space.

0230 Global statistics for data sharing.

0234 User authorization information.

0254 Group buffer pool usage in the data sharing group.

0306 Returns compressed or decompressed log records in both a data sharing or non data-sharing environment. For IFCID 306 requests, your program's return area must reside in ECSA key 7 storage with

the IFI application program running in key 0 supervisor state. The IFI application program must set the eye catcher to "I306" before making the IFCID 306 call. See "Instrument facility communications area (IFCA)" on page 1174 for more information about the instrumentation facility communication area (IFCA) and what is expected of the monitor program.

0316 Returns information about the contents of the dynamic statement cache. The IFI application can request information for all statements in the cache, or provide qualification parameters to limit the data returned. DB2 reports the following information about a cached statement:

- A statement name and ID that uniquely identify the statement
- If IFCID 0318 is active, performance statistics for the statement
- The first 60 bytes of the statement text

0317 Returns the complete text of an SQL statement in the dynamic statement cache and the PREPARE attributes string. You must provide the statement name and statement ID from IFCID 0316 output. For more information about using IFI to obtain information about the dynamic statement cache, see "Using READS calls to monitor the dynamic statement cache."

For more information about IFCID field descriptions, see the mapping macros in *prefix*.SDSNMACS. See also "DB2 trace" on page 1189 and Appendix D, "Interpreting DB2 trace output," on page 1135 for additional information.

Using READS calls to monitor the dynamic statement cache

You can use READS requests from an IFI application to monitor the contents of the dynamic statement cache, and optionally, to see some accumulated statistics for those statements. This strategy can help you detect and diagnose performance problems for those cached dynamic SQL statements.

An IFI program that monitors the dynamic statement cache should include these steps:

1. Acquire and initialize storage areas for common IFI communication areas.
2. Issue an IFI COMMAND call to start performance trace class 30 for IFCID 0318. This step enables statistics collection for statements in the dynamic statement cache. See "Controlling collection of dynamic statement cache statistics with IFCID 0318" on page 1170 for information on when you should start a trace for IFCID 0318.
3. Put the IFI program into a wait state. During this time, SQL applications in the subsystem execute dynamic SQL statements by using the dynamic statement cache.
4. Resume the IFI program after enough time has elapsed for a reasonable amount of activity to occur in the dynamic statement cache.
5. Set up the qualification area for a READS call for IFCID 0316 as described in Table 302 on page 1159.
6. Set up the IFCID area to request data for IFCID 0316.
7. Issue an IFI READS call to retrieve the qualifying cached SQL statements.
8. Examine the contents of the return area.

   For a statement with unexpected statistics values:

   a. Obtain the statement name and statement ID from the IFCID 0316 data.

b. Set up the qualification area for a READS call for IFCID 0317, as described in Table 302 on page 1159.
c. Set up the IFCID area to request data for IFCID 0317.
d. Issue a READS call for IFCID 0317 to get the entire text of the statement.
e. Obtain the statement text from the return area.
f. Use the statement text to execute an SQL EXPLAIN statement.
g. Fetch the EXPLAIN results from the PLAN_TABLE.

9. Issue an IFI COMMAND call to stop monitor trace class 1.
10. Issue an IFI COMMAND call to stop performance trace class 30 for IFCID 0318.

An IFI program that monitors deadlocks and timeouts of cached statements should include these steps:

1. Acquire and initialize storage areas for common IFI communication areas.
2. Issue an IFI COMMAND call to start monitor trace class 1. This step lets you make READS calls for IFCID 0316 and IFCID 0317.
3. Issue an IFI COMMAND call to start performance trace class 30 for IFCID 0318. This step enables statistics collection for statements in the dynamic statement cache. See "Controlling collection of dynamic statement cache statistics with IFCID 0318" for information on when you should start a trace for IFCID 0318.
4. Start performance trace class 3 for IFCID 0172 to monitor deadlocks, or performance trace class 3 for IFCID 0196 to monitor timeouts.
5. Put the IFI program into a wait state. During this time, SQL applications in the subsystem execute dynamic SQL statements by using the dynamic statement cache.
6. Resume the IFI program when a deadlock or timeout occurs.
7. Issue a READA request to obtain IFCID 0172 or IFCID 0196 trace data.
8. Obtain the cached statement ID of the statement that was involved in the deadlock or timeout from the IFCID 0172 or IFCID 0196 trace data. Using the statement ID, set up the qualification area for a READS call for IFCID 0316 or IFCID 0317, as described in Table 302 on page 1159.
9. Set up the IFCID area to request data for IFCID 0316 or IFCID 0317.
10. Issue an IFI READS call to retrieve the qualifying cached SQL statement.
11. Examine the contents of the return area.
12. Issue an IFI COMMAND call to stop monitor trace class 1.
13. Issue an IFI COMMAND call to stop performance trace class 30 for IFCID 0318 and performance trace class 3 for IFCID 0172 or IFCID 0196.

Controlling collection of dynamic statement cache statistics with IFCID 0318

The collection of statistics for statements in the dynamic statement cache can increase the processing cost for those statements. To minimize this increase, use IFCID 0318 to enable and disable the collection of dynamic statement cache statistics. When IFCID 0318 is inactive, DB2 does not collect those statistics. DB2 tracks the statements in the dynamic statement cache, but does not accumulate the statistics as those statements are used. When you are not actively monitoring the cache, you should turn off the trace for IFCID 0318.

If you issue a READS call for IFCID 0316 while IFCID 0318 is inactive, DB2 returns identifying information for all statements in the cache, but returns 0 in all the IFCID 0316 statistics counters.

When you stop or start the trace for IFCID 0318, DB2 resets the IFCID 0316 statistics counters for all statements in the cache to 0.

READA: Syntax and usage with IFI

The READA function allows a monitor program to asynchronously read data that has accumulated in an OP*n* buffer.

Authorization for READA requests through IFI

On a READA request the application program must own the specified destination buffer, or the request is denied. You can obtain ownership of a storage buffer by issuing a START TRACE to an OP*n* destination. If the primary authorization ID or one of the secondary authorization IDs of the process does not have MONITOR1 or MONITOR2 privilege, the request is denied. READA requests are checked for authorization once for each user of the thread. (Several users can use the same thread, but an authorization check is performed each time the user of the thread changes.)

Syntax for READA requests through IFI

READA requests that are issued through IFI have the following syntax:

```
CALL DSNWLI,('READA   ',ifca,return-area),VL
```

ifca
: Contains information about the OP*n* destination and the ownership token value (IFCAOWNR) at call initiation. After the READA call completes, the IFCA contains the return code, reason code, the number of bytes moved to the return area, the number of bytes not moved to the return area if the area was too small, and the number of records lost. See “Common communication areas for IFI calls” on page 1174 for a description of the IFCA.

return-area
: Contains the varying-length records that are returned by the instrumentation facility. If the return area is too small, as much of the output as will fit is placed into the area (a complete varying-length record). Reason code 00E60802 is returned in cases where the monitor program's return area is not large enough to hold the returned data. See “Return area” on page 1178 for a description of the return area.

IFI allocates up to eight OP buffers upon request from storage above the line in extended CSA. IFI uses these buffers to store trace data until the owning application performs a READA request to transfer the data from the OP buffer to the application's return area. An application becomes the owner of an OP buffer when it issues a START TRACE command and specifies a destination of
OPN or OPX. Each buffer can be of size 256 KB to 16 MB. IFI allocates a
maximum of 16 MB of storage for each of the eight OP buffers. The default monitor buffer size is determined by the MONSIZE parameter in the DSNZPARM module.

Usage notes for READA requests through IFI

You can use a monitor trace that uses any one of eight online performance monitor destinations, OP*n*, (where *n* is equal to a value from 1 to 8). Typically, the destination of OP*n* is only used with commands issued from a monitor program.

For example, the monitor program can pass a specific online performance monitor destination (OP1, for example) on the START TRACE command to start asynchronous trace data collection.

If the monitor program passes a generic destination of OPX, the instrumentation facility assigns the next available buffer destination slot and returns the OP*n* destination name to the monitor program. To avoid conflict with another trace or program that might be using an OP buffer, you should use the generic OPX specification when you start tracing. You can then direct the data to the destination specified by the instrumentation facility with the START or MODIFY TRACE commands.

There are times, however, when you should use a specific OP*n* destination initially:

- When you plan to start numerous asynchronous traces to the same OP*n* destination. To do this, you must specify the OP*n* destination in your monitor program. The OP*n* destination started is returned in the IFCA.
- When the monitor program specifies that a particular monitor class (defined as available) together with a particular destination (for example OP7) indicates that certain IFCIDs are started. An operator can use the DISPLAY TRACE command to determine which monitors are active and what events are being traced.

Buffering data: To have trace data go to the OP*n* buffer, you must start the trace from within the monitor program. After the trace is started, DB2 collects and buffers the information as it occurs. The monitor program can then issue a read asynchronous (READA) request to move the buffered data to the monitor program. The buffering technique ensures that the data is not being updated by other users while the buffer is being read by the READA caller. For more information, see "Data integrity and IFI" on page 1183.

Possible data loss: You can activate all traces and have the trace data buffered. However, this plan is definitely not recommended because performance might suffer and data might be lost.

Data loss occurs when the buffer fills before the monitor program can obtain the data. DB2 does not wait for the buffer to be emptied, but, instead, informs the monitor program on the next READA request (in the IFCARLC field of the IFCA) that the data has been lost. The user must have a high enough dispatching priority that the application can be posted and then issue the READA request before significant data is lost.

Asynchronous data and READA requests through IFI

DB2 buffers all IFCID data that is activated by the START TRACE command and passes it to a monitor program on a READA request. The IFCID events include all of the following:

- Serviceability
- Statistics
- Accounting
- Performance
- Audit data
- IFCIDs defined for the IFI write function

IFCID events are discussed in "DB2 trace" on page 1189.

Your monitor program can request an asynchronous buffer, which records trace data as trace events occur. The monitor program is then responsible for unloading

the buffer on a timely basis. One method is to set a timer to wake up and process the data. Another method is to use the buffer information area on a START TRACE command request, shown in Table 301 on page 1156, to specify an ECB address to post when a specified number of bytes have been buffered.

Example of READA requests through IFI

The following depicts the logic flow for monitoring DB2 accounting and for displaying the information on a terminal:

1. Initialize.
2. Use GETMAIN to obtain a storage area equal to BUFSIZE.
3. Start an accounting trace by issuing a DB2 START TRACE=ACCTG DEST=OPX command through IFI indicating to wake up this routine by a POST whenever the buffer is 20% full.
4. Check the status in the IFCA to determine if the command request was successful.
5. WAIT for the buffer to be posted.
6. Clear the post flag.
7. Call IFI to obtain the buffer data via a READA request.
8. Check the status of the IFCA to determine if the READA request was successful.
9. De-block the information provided.
10. Display the information on a terminal.
11. Loop back to the WAIT.

WRITE: Syntax and usage with IFI

A monitor program can write information to a DB2 trace destination by issuing a write (WRITE) request for a specific IFCID.

Authorization for WRITE requests through IFI

WRITE requests are not checked for authorization, but a DB2 trace must be active for the IFCID being written. If the IFCID is not active, the request is denied. For a WRITE request, no other authorization checks are made.

Syntax for WRITE requests through IFI

WRITE requests that are issued through IFI have the following syntax:

```
CALL DSNWLI,('WRITE   ',ifca,output-area,ifcid-area),VL
```

The write function must specify an IFCID area. The data that is written is defined and interpreted by your site.

ifca
: Contains information regarding the success of the call. See “Instrument facility communications area (IFCA)” on page 1174 for a description of the IFCA.

output-area
: Contains the varying-length of the monitor program's data record to be written. See “Output area” on page 1179 for a description of the output area.

ifcid-area
: Contains the IFCID of the record to be written. Only the IFCIDs that are defined to the write function (see Table 304 on page 1174) are allowed. If

an invalid IFCID is specified or the IFCID is not active (not started by a TRACE command), no data is written. See Table 304 for IFCIDs that can be used by the write function.

Table 304. Valid IFCIDs for WRITE function

IFCID (decimal)	IFCID (hex)	Trace type	Class	Comment
0146	0092	Auditing	9	Write to IFCID 146
0151	0097	Accounting	4	Write to IFCID 151
0152	0098	Statistics	2	Write to IFCID 152
0153	0099	Performance	1	Background events and write to IFCID 153
0154	009A	Performance	15	Write to IFCID 154
0155	009B	Monitoring	4	Write to IFCID 155
0156	009C	Serviceability	6	Reserved for user-defined serviceability trace

See “IFCID area” on page 1179 for a description of the IFCID area.

Usage notes for WRITE requests through IFI

A WRITE request is written to a destination that is activated by a START TRACE command.

Recommendation: If your site uses the IFI WRITE function, establish usage procedures and standards. Procedures ensure that the correct IFCIDs are active when DB2 is performing the WRITE function. Standards determine the records and record formats that a monitor program sends to DB2.

Recommendation: Because your site can use one IFCID to contain many different records, place your site's record type and sub-type in the first fields in the data record .

Common communication areas for IFI calls

The following communication areas are used on all IFI calls:
- “Instrument facility communications area (IFCA)”
- “Return area” on page 1178
- “IFCID area” on page 1179
- “Output area” on page 1179

Instrument facility communications area (IFCA)

The program's instrumentation facility communication area (IFCA) is a communications area between the monitor program and IFI. A required parameter on all IFI requests, the IFCA contains information about the success of the call in its return code and reason code fields.

The monitor program is responsible for allocating storage for the IFCA and initializing it. The IFCA must be initialized to binary zeros and the eye catcher, 4-byte owner field, and length field must be set by the monitor program. Failure to properly initialize the IFCA results in denying any IFI requests.

The monitor program is also responsible for checking the IFCA return code and reason code fields to determine the status of the request.

The IFCA fields are described in Table 305.

Table 305. Instrumentation facility communication area. The IFCA is mapped by assembler mapping macro DSNDIFCA.

Name	Hex offset	Data type	Description
IFCALEN	0	Hex, 2 bytes	Length of IFCA.
IFCAFLGS	2	Hex, 1 byte	Processing flags. • IFCAGLBL, X'80' This bit is on if an IFI request is to be processed on all members of a data sharing group.
	3	Hex, 1 byte	Reserved.
IFCAID	4	Character, 4 bytes	Eye catcher for block, IFCA.
IFCAOWNR	8	Character, 4 bytes	Owner field, provided by the monitor program. This value is used to establish ownership of an OP*n* destination and to verify that a requester can obtain data from the OP*n* destination. This is *not* the same as the owner ID of a plan.
IFCARC1	C	4-byte signed integer	Return code for the IFI call. Binary zero indicates a successful call. See Part 3 of *DB2 Codes* for information about reason codes. For a return code of 8 from a COMMAND request, the IFCAR0 and IFCAR15 values contain more information.
IFCARC2	10	4-byte signed integer	Reason code for the IFI call. Binary zero indicates a successful call. See Part 3 of *DB2 Codes* for information about reason codes.
IFCABM	14	4-byte signed integer	Number of bytes moved to the return area. A non-zero value in this field indicates information was returned from the call. Only complete records are moved to the monitor program area.
IFCABNM	18	4-byte signed integer	Number of bytes that did not fit in the return area and still remain in the buffer. Another READA request will retrieve that data. Certain IFI requests return a known quantity of information. Other requests will terminate when the return area is full.
	1C	4-byte signed integer	Reserved.
IFCARLC	20	4-byte signed integer	Indicates the number of records lost prior to a READA call. Records are lost when the OP buffer storage is exhausted before the contents of the buffer are transferred to the application program via an IFI READA request. Records that do not fit in the OP buffer are not written and are counted as records lost.
IFCAOPN	24	Character, 4 bytes	Destination name used on a READA request. This field identifies the buffer requested, and is required on a READA request. Your monitor program must set this field. The instrumentation facility fills in this field on START TRACE to an OP*n* destination from an monitor program. If your monitor program started multiple OP*n* destination traces, the first one is placed in this field. If your monitor program did not start an OP*n* destination trace, the field is not modified. The OP*n* destination and owner ID are used on subsequent READA calls to find the asynchronous buffer.
IFCAOPNL	28	2-byte signed integer	Length of the OP*n* destinations started. On any command entered by IFI, the value is set to X'0004'. If an OP*n* destination is started, the length is incremented to include all OP*n* destinations started.
	2A	2-byte signed integer	Reserved.
IFCAOPNR	2C	Character, 8 fields of 4 bytes each	Space to return 8 OP*n* destination values.

Table 305. Instrumentation facility communication area (continued). The IFCA is mapped by assembler mapping macro DSNDIFCA.

Name	Hex offset	Data type	Description
IFCATNOL	4C	2-byte signed integer	Length of the trace numbers plus 4. On any command entered by IFI the value is set to X'0004'. If a trace is started, the length is incremented to include all trace numbers started.
	4E	2-byte signed integer	Reserved.
IFCATNOR	50	Character, 8 fields of 2 bytes each.	Space to hold up to eight EBCDIC trace numbers that were started. The trace number is required if the MODIFY TRACE command is used on a subsequent call.
IFCADL	60	Hex, 2 bytes	Length of diagnostic information.
	62	Hex, 2 bytes	Reserved.

Table 305. Instrumentation facility communication area (continued). The IFCA is mapped by assembler mapping macro DSNDIFCA.

Name	Hex offset	Data type	Description
IFCADD	64	Character, 80 bytes	Diagnostic information. • IFCAFCI, offset 64, 6 bytes This contains the RBA of the first CI in the active log if IFCARC2 is 00E60854. See "Reading specific log records (IFCID 0129)" on page 1122 for more information. • IFCAR0, offset 6C, 4 bytes For COMMAND requests, this field contains -1 or the return code from the component that executed the command. • IFCAR15, offset 70, 4 bytes For COMMAND requests, this field contains one of the following values: **0** The command completed successfully. **4** Internal error. **8** The command was not processed because of errors in the command. **12** The component that executed the command returned the return code in IFCAR0. **16** An abend occurred during command processing. Command processing might be incomplete, depending on when the error occurred. See IFCAR0 for more information. **20** Response buffer storage was not available. The command completed, but no response messages are available. See IFCAR0 for more information. **24** Storage was not available in the DSNMSTR address space. The command was not processed. **28** CSA storage was not available. If a response buffer is available, the command might have partially completed. See IFCAR0 for more information. **32** The user is not authorized to issue the command. The command was not processed. • IFCAGBPN, offset 74, 8 bytes This is the group buffer pool name in error if IFCARC2 is 00E60838 or 00E60860 • IFCABSRQ, offset 88, 4 bytes This is the size of the return area required when the reason code is 00E60864. • IFCAHLRS, offset 8C, 6 bytes This field can contain the highest LRSN or log RBA in the active log (when WQALLMOD is 'H'). Or, it can contain the RBA of the log CI given to the Log Exit when the last CI written to the log was not full, or an RBA of zero (when WQALLMOD is 'P').
IFCAGRSN	98	Four-byte signed integer	Reason code for the situation in which an IFI calls requests data from members of a data sharing group, and not all the data is returned from group members. See Part 3 of *DB2 Codes* for information about reason codes.

Table 305. Instrumentation facility communication area (continued). The IFCA is mapped by assembler mapping macro DSNDIFCA.

Name	Hex offset	Data type	Description
IFCAGBM	9C	Four-byte signed integer	Total length of data that was returned from other data sharing group members and fit in the return area.
IFCAGBNM	A0	Four-byte signed integer	Total length of data that was returned from other data sharing group members and did not fit in the return area..
IFCADMBR	A4	Character, 8 bytes	Name of a single data sharing group member on which an IFI request is to be executed. Otherwise, this field is blank. If this field contains a member name, DB2 ignores field IFCAGLBL.
IFCARMBR	AC	Character, 8 bytes	Name of the data sharing group member from which data is being returned. DB2 sets this field in each copy of the IFCA that it places in the return area, not in the IFCA of the application that makes the IFI request.

Return area

You must specify a return area on all READA, READS, and COMMAND requests. IFI uses the return area to return command responses, synchronous data, and asynchronous data to the monitor program. Table 306describes the return area.

Table 306. Return area

Hex offset	Data type	Description
0	Signed 4-byte integer	The length of the return area, plus 4. This must be set by the monitor program. The valid range for READA requests is 100 to 1048576 (X'00000064' to X'00100000'). The valid range for READS requests is 100 to 2147483647 (X'00000064' to X'7FFFFFFF').
4	Character, varying-length	DB2 places as many varying-length records as it can fit into the area following the length field. The monitor program's length field is not modified by DB2. Each varying-length trace record has a 2-byte or 4-byte length field, depending on the high-order bit. if the high-order bit is on, the length field is 4 bytes. If the high-order bit is off, the length field is the first 2 bytes. In this case, the third byte indicates whether the record is spanned, and the fourth byte is reserved. After a COMMAND request, the last character in the return area is a new-line character (X'15').

Table 307 shows the return area for IFCID 0306.

Table 307. Return area using IFCID 0306

Hex	Data type	Description
0	Signed four-byte integer	The length of the return area.
4	Character, 4 bytes	The eye catcher, a constant, I306. Beginning of QW0306OF mapping.
8	Character, 60 bytes	Reserved.
44	Signed four-byte integer	The length of the returned data.

Note: For more information about reading log records, see Appendix C, "Reading log records," on page 1111

The destination header for data that is returned on a READA or READS request is mapped by macro DSNDQWIW or the header QW0306OF for IFCID 306 requests.

Please refer to *prefix*.SDSNIVPD(DSNWMSGS) for the format of the trace record and its header. The size of the return area for READA calls should be as large the size specified with the BUFSIZE keyword on the START TRACE command.

Data returned on a COMMAND request consists of varying-length segments (X'xxxxrrrr' where the length is 2 bytes and the next 2 bytes are reserved), followed by the message text. More than one record can be returned. The last character in the return area is a new-line character (X'15').

The monitor program must compare the number of bytes moved (IFCABM in the IFCA) to the sum of the record lengths to determine when all records have been processed.

IFCID area

You must specify the IFCID area on READS and WRITE requests. The IFCID area contains the IFCIDs to process. Table 308 shows the IFCID area.

Table 308. IFCID area

Hex Offset	Data type	Description
0	Signed two-byte integer	Length of the IFCID area, plus 4. The length can range from X'0006' to X'0044'. For WRITE requests, only one IFCID is allowed, so the length must be set to X'0006'. For READS requests, you can specify multiple IFCIDs. If so, you must be aware that the returned records can be in a different sequence than requested and some records can be missing.
2	Signed two-byte integer	Reserved.
4	Hex, *n* fields of 2 bytes each	The IFCIDs to be processed. Each IFCID is placed contiguous to the previous IFCID for a READS request. The IFCIDs start at X'0000' and progress upward. You can use X'FFFF' to signify the last IFCID in the area to process.

Output area

The output area is used on command and WRITE requests. The first two bytes contain the length of the monitor program's record to write or the DB2 command to be issued, plus 4 additional bytes. The next two bytes are reserved. You can specify any length from 10 to 4096 (X'000A0000' to X'10000000'). The rest of the area is the actual command or record text.

Example: In an assembler program, a START TRACE command is formatted in the following way:

```
DC   X'002A0000'       LENGTH INCLUDING LL00 + COMMAND
DC   CL37'-STA TRACE(MON) DEST(OPX) BUFSIZE(256)'
```

Using IFI in a data sharing group

You can use IFI READA and READS calls in an application that runs on one member of a data sharing group to gather trace data from other members of the data sharing group. You can also use an IFI COMMAND call to execute a command at another member of a data sharing group. In addition to the IFCA fields that you use for an IFI request for a single subsystem, you need to set or read the following fields for an IFI request for a data sharing group:

IFCAGLBL
Set this flag on to indicate that the READS or READA request should be sent to all members of the data sharing group.

IFCADMBR
If you want an IFI READS, READA, or COMMAND request to be executed at a single member of the data sharing group, assign the name of the group member to this field. If you specify a name in this field, DB2 ignores IFCAGLBL. If the name that you specify is not active when DB2 executes the IFI request, DB2 returns an error.

Recommendation: To issue a DB2 command that does not support SCOPE(GROUP) at another member of a data sharing group, set the IFCADMBR field and issue an IFI COMMAND.

IFCARMBR
The name of the data sharing member that generated the data that follows the IFCA. DB2 sets this value in the copy of the IFCA that it places in the requesting program's return area.

IFCAGRSN
A reason code that DB2 sets when not all data is returned from other data sharing group members. See Part 3 of *DB2 Codes* for specific reason codes.

IFCAGBM
The number of bytes of data that other members of the data sharing group return and that the requesting program's return area **can** contain.

IFCAGBNM
The number of bytes of data that members of the data sharing group return but that the requesting program's return area **cannot** contain.

As with READA or READS requests for single DB2 subsystems, you need to issue a START TRACE command before you issue the READA or READS request. You can issue START TRACE with the parameter SCOPE(GROUP) to start the trace at all members of the data sharing group. For READA requests, specify DEST(OPX) in the START TRACE command. DB2 collects data from all data sharing members and returns it to the OPX buffer for the member from which you issue the READA request.

If a new member joins a data sharing group while a trace with SCOPE(GROUP) is active, the trace starts at the new member.

After you issue a READS or READA call for all members of a data sharing group, DB2 returns data from all members in the requesting program's return area. Data from the local member is first, followed by the IFCA and data for all other members.

Example: If the local DB2 is called DB2A, and the other two members in the group are DB2B and DB2C, the return area looks like this:

```
Data for DB2A
IFCA for DB2B (DB2 sets IFCARMBR to DB2B)
Data for DB2B
IFCA for DB2C (DB2 sets IFCARMBR to DB2C)
Data for DB2C
```

If an IFI application requests data from a single other member of a data sharing group (IFCADMBR contains a member name), the requesting program's return area

contains the data for that member but no IFCA for the member. All information about the request is in the requesting program's IFCA.

Because a READA or READS request for a data sharing group can generate much more data than a READA or READS request for a single DB2, you need to increase the size of your return area to accommodate the additional data.

Interpreting records returned by IFI

This section describes the format of the records returned by IFI as a result of READA, READS, and COMMAND requests.

Trace data record format

Trace records that are returned from READA and READS requests contain the following sections:

- A writer header that reports the length of the entire record, whether the record was in the first, middle, or last section of data, and other specific information for the writer.

  The writer header for IFI is mapped by DSNDQWIW or the header QW0306OF for IFCID 306 requests. See the mapping macros in *prefix*.SDSNMACS for the formats.
- A self-defining section
- A product section containing specific DB2 information based on the active trace
- Data areas containing the actual recorded data are mapped by multiple mapping macros described in *prefix*.SDSNMACS.

For detailed information about the format of trace records and their mapping macros, see Appendix D, “Interpreting DB2 trace output,” on page 1135, or see the mapping macros in *prefix*.SDSNMACS.

Figure 162 on page 1182 shows the return area after a READS request successfully executed.

```
DFSERA10 - PRINT PROGRAM
   ⋮
          A        B                  C
000000   05A80000 00000510  00980001 00000054   00B80001 0000010C  01000001 0000020C
000020   01160001 00000324  01B00001 000004D4   00000000 000004D4  00080001 000004DC
                                                          D
000040   00010001 000004E0  00000000 000004E0   00300001 80000018  00000010 000003E8
000060   00640064 000A0028  003D0000 0000A000   00033000 00033000  00010000 E0000000
000080   00000000 00000000  00000000 C1C4D4C6   F0F0F140 F0F20080  00003084 00000000
0000A0   00002000 0005003C  0028F040 40404040   40404040 40404040  40404040 40404040

   ⋮
000320   B0000000 202701D0  E2D7D9D4 D2C4C4F0   F0F1F940 01980064  00000000 E7C14000
000340   00400280 C5E2E8E2  C1C4D440 40000000   000E1000 000001BC  000001B0 C9C2D4E4
000360   E2C5D940 C9D9D3D4  D7D9D6C3 C9D9D3D4   0000003C 0000012C  0000000A 8080008C
000380   00FA0000 00007D00  000A0014 00050028   000E0002 00080008  00400077 00000514
0003A0   000003E8 012C0000  0000000E 000A01F4   00FA0000 00000032  000003E8 00002710
0003C0   E2E8E2C1 C4D44040  E2E8E2D6 D7D94040   E2E8E2D6 D7D94040  000A0080 00140000
0003E0   00000080 0005000A  13880078 0008000A   00040004 00040005  0001000A 00020005
000400   00003000 00007800  00000001 000007D0   00040400 00780078  00010003 00019000
000420   0000000A 00000020  00000019 00000000   0005000A 0006000A  00640064 00040063
000440   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
000460   30C4E2D5 D9C7C3D6  D3C4E2D5 6DD9C5C7   C9E2E3C5 D96DC1D7  D7D3C4E2 D56DD9C5
000480   C7C9E2E3 C5D96DD6  C2D1E380 C4E2D5D9   C7C6C4C2 000009FD  C5000000 00001060
0004A0   00020000 00001000  40000000 00000000   00000000 00000000  00000000 00000000
0004C0   00000000 00000000  00000000 F1F161F1   F361F9F2 C4E2D5C3  F3F1F040 80000000
0004E0   00160030 C6C1C340  00010000 C4C4C640   40404040 C1800002  00000000 C1C3E3C9
                                                E        F
000500   E5C54040 00000000  00000000 00000000   004C011A 006A0A31  00B45B78 E2E2D6D7
000520   A6E9C7D5 EBDB1104  00000008 00000002   00000001 E2C1D5E3  C16DE3C5 D9C5E2C1
000540   6DD3C1C2 C4C2F2D5  C5E34040 D3E4D5C4   F0404040 A6E9C7D2  E73C0001 004C0200
000560   E2E8E2C1 C4D44040  D4D7C9E3 E2F14040   40404040 C2C1E3C3  C8404040 C4E2D5C5
000580   C4C3D340 E2E8E2C1  C4D44040 00000001   00000000 00000000  00000000 00000000
0005A0   00000000 00000000
```

Figure label	Description
A 05A8	Length of record. The next two bytes are reserved.
B 00000510	Offset to product section standard header.
C 00000054	Offset to first data section.
D 80000018	Beginning of first data section.
E 004C011A	Beginning of product section standard header.
F 006A	IFCID (decimal 106).

Figure 162. Example of IFI return area after READS request (IFCID 106). This output was assembled by a user-written routine and printed with the DFSERA10 print program of IMS.

For more information about IFCIDs and mapping macros, see “DB2 trace” on page 1189 and Appendix D, “Interpreting DB2 trace output,” on page 1135.

Command record format

The record that is returned from a command request can contain none or many message text segments. Each segment is a varying-length message (LLZZ, where LL is the 2-byte length and ZZ is a 2-byte reserved area) followed by message text. The IFCA’s IFCABM field contains the total number of bytes moved.

Figure 163 on page 1183 shows the return area after a START TRACE command successfully executes.

```
DFSERA10 - PRINT PROGRAM
   ⋮
          A        B         C        D
 000000   007E0000 0000007A  003C0001 C4E2D5E6   F1F3F0C9 406F40D4  D6D540E3 D9C1C3C5
 000020   40E2E3C1 D9E3C5C4  6B40C1E2 E2C9C7D5   C5C440E3 D9C1C3C5  40D5E4D4 C2C5D940
                   E         F
 000040   F0F24015 003A0001  C4E2D5F9 F0F2F2C9   406F40C4 E2D5E6E5  C3D4F140 7D60E2E3
 000060   C1D9E340 E3D9C1C3  C57D40D5 D6D9D4C1   D340C3D6 D4D7D3C5  E3C9D6D5 4015
```

Figure 163. Example of IFI return area after a START TRACE command. This output was assembled with a user-written routine and printed with DFSERA10 program of IMS.

Figure label	Description
A 007E0000	Field entered by print program
B 0000007A	Length of return area
C 003C	Length of record (003C). The next two bytes are reserved.
D C4E2D5E6	Beginning of first message
E 003A	Length of record. The next two bytes are reserved.
F C4E2D5F9	Beginning of second message

The IFCABM field in the IFCA would indicate that X'00000076' (C + E) bytes have been moved to the return area.

Data integrity and IFI

Although IFI displays DB2 statistics, agent status, and resource status data, it does not change or display DB2 database data. When a process retrieves data, information is moved from DB2 fetch-protected storage to the user's address space, or from the address space to DB2 storage, in the storage key of the requester. Data that is moved by the READA request is serialized so that only *clean data* is moved to the address space of the requester.

The serialization techniques used to obtain data for a given READA request might minimally degrade performance on processes that simultaneously store data into the instrumentation facility buffer. Failures during the serialization process are handled by DB2.

The DB2 structures that are searched on a READS request are validated before they are used. If the DB2 structures are updated while being searched, inconsistent data might be returned. If the structures are deleted while being searched, users might access invalid storage areas, causing an abend. If an abend does occur, the functional recovery routine of the instrumentation facility traps the abend and returns information about it to the application program's IFCA.

Auditing data and IFI

Starting, stopping, or modifying trace through IFI might cause changes to the events being traced for audit. Each time these trace commands are processed a record is sent to the destination processing the trace type. In the case of audit, the audit destination receives a record indicating a trace status has been changed. These records are IFCID 0004 and 0005.

Locking considerations for IFI

When you design your application to use IFI, consider the potential for locking delays, deadlocks, and time-out conflicts. Locks are obtained for IFI in the following situations:

- When READS and READA requests are checked for authorization, short duration locks on the DB2 catalog are obtained. When the check is made, subsequent READS or READA requests are not checked for authorization. Remember, if you are using the access control exit routine, then that routine might be controlling the privileges that the monitor trace can use.
- When DB2 commands are submitted, each command is checked for authorization. DB2 database commands obtain additional locks on DB2 objects.

A program can issue SQL statements through an attachment facility and DB2 commands through IFI. This environment creates the potential for an application to deadlock or time-out with itself over DB2 locks acquired during the execution of SQL statements and DB2 database commands. You should ensure that all DB2 locks acquired by preceding SQL statements are no longer held when the DB2 database command is issued. You can do this by:

- Binding the DB2 plan with ACQUIRE(USE) and RELEASE(COMMIT) bind parameters
- Initiating a commit or rollback to free any locks your application is holding, before issuing the DB2 command

If you use SQL in your application, the time between commit operations should be short. For more information on locking, see Chapter 31, “Improving concurrency,” on page 809.

Recovery considerations for IFI

When an application program issues an IFI call, the requested function is immediately performed. If the application program subsequently abends, the IFI request is **not** backed out. In contrast, requests that do not use IFI are committed and abended as usual. For example, if an IFI application program also issues SQL calls, a program abend causes the SQL activity to be backed out.

Errors and IFI

While using IFI, you might encounter any of these types of error:

- Connection failure, because the user is not authorized to connect to DB2
- Authorization failure, because the process is not authorized to access the DB2 resources specified

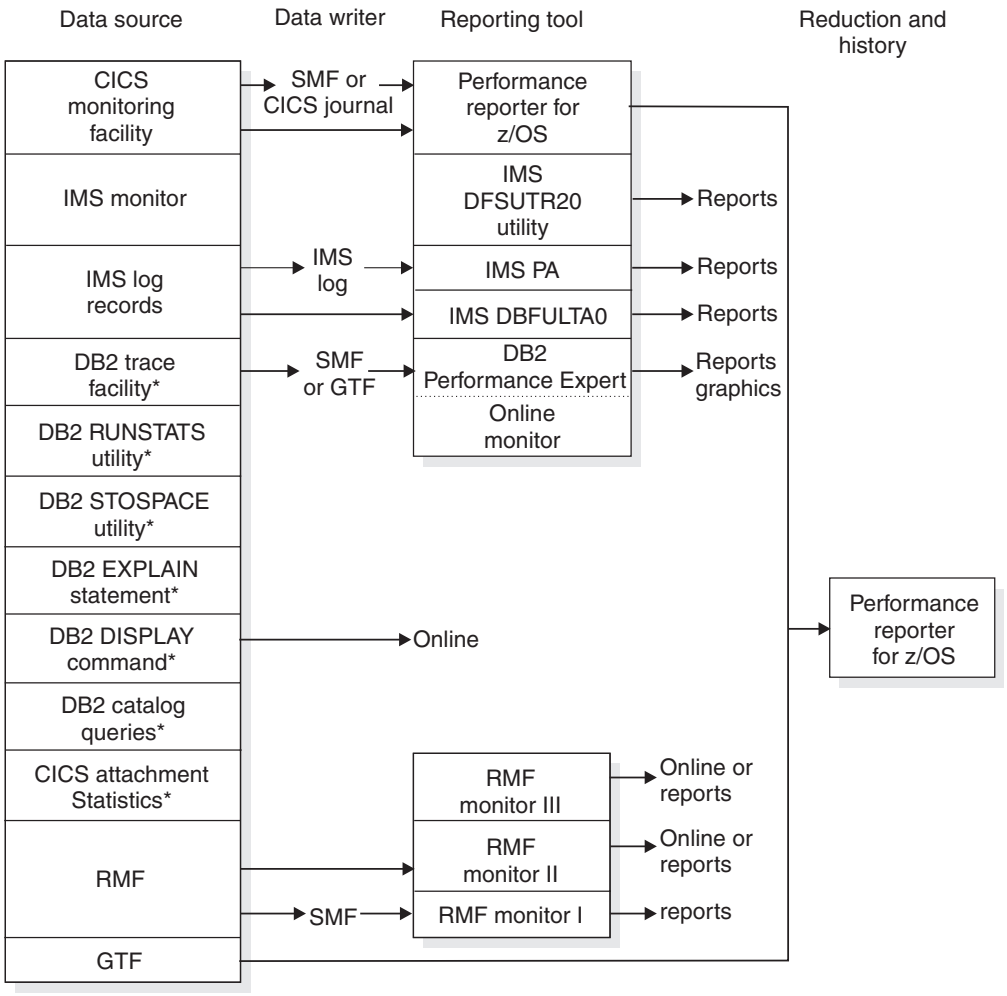
Requests sent through IFI can fail for a variety of reasons, including:

- One or more parameters are invalid.
- The IFCA area is invalid.
- The specified OP*n* is in error.
- The requested information is not available.
- The return area is too small.

Return code and reason code information is stored in the IFCA in fields IFCARC1 and IFCARC2. Further return and reason code information is contained in Part 3 of *DB2 Codes*.

Appendix F. Using tools to monitor performance

This appendix describes the various facilities for monitoring DB2 activity and performance. It includes information on facilities within the DB2 product as well as tools available outside of DB2. Figure 164 shows various monitoring tools that can be used in a DB2 environment.



*Facilities available with the DB2 product

Figure 164. Monitoring tools in a DB2 environment

Table 310 describes these monitoring tools.

Table 310. Monitoring tools in a DB2 environment

Monitoring tool	Description
CICS Attachment Facility statistics	Provide information about the use of CICS threads. This information can be displayed on a terminal or printed in a report.

© Copyright IBM Corp. 1982, 2008

Table 310. Monitoring tools in a DB2 environment (continued)

Monitoring tool	Description
CICS Monitoring Facility (CMF)	Provides performance information about each CICS transaction executed. It can be used to investigate the resources used and the time spent processing transactions. Be aware that overhead is significant when CMF is used to gather performance information.
DB2 catalog queries	Help you determine when to reorganize table spaces and indexes. See the description of the REORG utility in Part 2 of *DB2 Utility Guide and Reference*.
DB2 Connect	Can monitor and report DB2 server-elapsed time for client applications that access DB2 data. See "Reporting server-elapsed time" on page 1017.
DB2 DISPLAY command	Gives you information about the status of threads, databases, buffer pools, traces, allied subsystems, applications, and the allocation of tape units for the archive read process. For information about the DISPLAY BUFFERPOOL command, see "Monitoring and tuning buffer pools using online commands" on page 675. For information about using the DISPLAY command to monitor distributed data activity, see "The DISPLAY command" on page 1013. For the detailed syntax of each command, refer to Chapter 2 of *DB2 Command Reference*.
DB2 EXPLAIN statement	Provides information about the access paths used by DB2. See Chapter 34, "Using EXPLAIN to improve SQL performance," on page 927 and Chapter 5 of *DB2 SQL Reference*.
OMEGAMON XE for DB2 Performance Expert	A licensed program that integrates the function of DB2 Buffer Pool Analyzer and DB2 Performance Monitor. OMEGAMON provides performance monitoring, reporting, buffer pool analysis, and a performance warehouse, all in one tool. OMEGAMON monitors all subsystem instances across many different platforms in a consistent way. You can use OMEGAMON to analyze DB2 trace records and optimize buffer pool usage. See "OMEGAMON" on page 1196 for more information.
OMEGAMON XE DB2 Performance Monitor (DB2 PM)	An orderable feature of DB2 that you can use to analyze DB2 trace records. As indicated previously, OMEGAMON includes the function of DB2 PM. OMEGAMON is described under "OMEGAMON" on page 1196.
DB2 RUNSTATS utility	Can report space use and access path statistics in the DB2 catalog. See "Gathering monitor statistics and update statistics" on page 912 and Part 2 of *DB2 Utility Guide and Reference*.
DB2 STOSPACE utility	Provides information about the actual space allocated for storage groups, table spaces, table space partitions, index spaces, and index space partitions. See Part 2 of *DB2 Utility Guide and Reference*.
DB2 trace facility	Provides DB2 performance and accounting information. It is described under "DB2 trace" on page 1189.
Generalized Trace Facility (GTF)	A z/OS service aid that collects information to analyze particular situations. GTF can also be used to analyze seek times and Supervisor Call instruction (SVC) usage, and for other services. See "Recording GTF trace data" on page 1195 for more information.

Table 310. Monitoring tools in a DB2 environment (continued)

Monitoring tool	Description
IMS DFSUTR20 utility	A print utility for IMS Monitor reports.
IMS Fast Path Log Analysis utility (DBFULTA0)	An IMS utility that provides performance reports for IMS Fast Path transactions.
IMS Performance Analyzer (IMS PA)	A separately licensed program that can be used to produce transit time information based on the IMS log data set. It can also be used to investigate response-time problems of IMS DB2 transactions.
Resource Measurement Facility (RMF)	An optional feature of z/OS that provides system-wide information on processor utilization, I/O activity, storage, and paging. There are three basic types of RMF sessions: Monitor I, Monitor II, and Monitor III. Monitor I and Monitor II sessions collect and report data primarily about specific system activities. Monitor III sessions collect and report data about overall system activity in terms of work flow and delay.
System Management Facility (SMF)	A z/OS service aid used to collect information from various z/OS subsystems. This information is dumped and reported periodically, such as once a day. Refer to "Recording SMF trace data" on page 1193 for more information.
Tivoli Decision Support for OS/390	Formerly known as Tivoli Performance Reporter for OS/390, is a licensed program that collects SMF data into a DB2 database and allows you to create reports on the data. See "Tivoli Decision Support for OS/390" on page 1197.

Using z/OS, CICS, and IMS tools

To monitor DB2 and CICS, you can use:
- RMF Monitor II for physical resource utilizations
- GTF for detailed I/O monitoring when needed
- Tivoli Decision Support for OS/390 for application processor utilization, transaction performance, and system statistics.

You can use RMF Monitor II to dynamically monitor system-wide physical resource utilizations, which can show queuing delays in the I/O subsystem.

In addition, the CICS attachment facility DSNC DISPLAY command allows any authorized CICS user to dynamically display statistical information related to thread usage and situations when all threads are busy. For more information about the DSNC DISPLAY command, see Chapter 2 of *DB2 Command Reference*.

Be sure that the number of threads reserved for specific transactions or for the pool is large enough to handle the actual load. You can dynamically modify the value specified in the CICS resource definition online (RDO) attribute ACCOUNTREC with the DSNC MODIFY TRANSACTION command. You might also need to modify the maximum number of threads specified for the MAX USERS field on installation panel DSNTIPE.

To monitor DB2 and IMS, you can use:
- RMF Monitor II for physical resource utilizations
- GTF for detailed I/O monitoring when needed

- IMS Performance Analyzer, or its equivalent, for response-time analysis and tracking all IMS-generated requests to DB2
- IMS Fast Path Log Analysis Utility (DBFULTA0) for performance reports for IMS Fast Path transactions.

In addition, the DB2 IMS attachment facility allows you to use the DB2 command DISPLAY THREAD command to dynamically observe DB2 performance.

Monitoring system resources

Monitor system resources to:
- Detect resource constraints (processor, I/O, storage)
- Determine how resources are consumed
- Check processor, I/O, and paging rate to detect a bottleneck in the system
- Detect changes in resource use over comparable periods.

Figure 165 shows an example of a suggested system resources report.

```
                        SYSTEM RESOURCES REPORT         DATE xx/xx/xx
                                                        FROM xx:xx:xx
                                                        TO xx:xx:xx

 TOTAL CPU Busy        74.3 %

   DB2 & IRLM           9.3 %
   IMS/CICS            45.3 %
   QMF Users            8.2 %
   DB2 Batch & Util     2.3 %
   OTHERS               9.2 %

   SYSTEM AVAILABLE    98.0 %

 TOTAL I/Os/sec.       75.5

 TOTAL Paging/sec.      6.8
                              Short          Medium        Long
                              Transaction    Transaction   Transaction

 Average Response Time        3.2 secs       8.6 secs      15.0 secs

 MAJOR CHANGES:
               DB2 application DEST07 moved to production
```

Figure 165. User-created system resources report

The RMF reports used to produce the information in Figure 165 were:
- The RMF CPU activity report, which lists TOTAL CPU busy and the TOTAL I/Os per second.
- RMF paging activity report, which lists the TOTAL paging rate per second for real storage.
- The RMF work load activity report, which is used to estimate where resources are spent. Each address space or group of address spaces to be reported on separately must have different SRM reporting or performance groups. The following SRM reporting groups are considered:
  - DB2 address spaces:
    - DB2 database address space (*ssnm*DBM1)
    - DB2 system services address space (*ssnm*MSTR)
    - Distributed data facility (*ssnm*DIST)
    - IRLM (IRLMPROC)
  - IMS or CICS
  - TSO-QMF

– DB2 batch and utility jobs

The CPU for each group is obtained using the ratio (A/B) × C, where:

A is the sum of CPU and service request block (SRB) service units for the specific group

B is the sum of CPU and SRB service units for all the groups

C is the total processor utilization.

The CPU and SRB service units must have the same coefficient.

You can use a similar approach for an I/O rate distribution.

MAJOR CHANGES shows the important environment changes, such as:

- DB2 or any related software-level change
- DB2 changes in the load module for system parameters
- New applications put into production
- Increase in the number of DB2 QMF users
- Increase in batch and utility jobs
- Hardware changes

MAJOR CHANGES is also useful for discovering the reason behind different monitoring results.

Monitoring transaction manager throughput

Use IMS or CICS monitoring facilities to determine throughput, in terms of transactions processed, and transaction response times. Depending on the transaction manager, you can use the following reports:

- IMS Performance Analyzer
- Fast Path Log Analysis Utility (DBFULTA0)
- Tivoli Decision Support for OS/390

In these reports:

- The transactions processed include DB2 and non-DB2 transactions.
- The transaction processor time includes the DB2 processor time for IMS but not for CICS.
- The transaction transit response time includes the DB2 transit time.

A historical database is useful for saving monitoring data from different periods. Such data can help you track the evolution of your system. You can use Tivoli Decision Support for OS/390 or write your own application based on DB2 and QMF when creating this database.

DB2 trace

The information under this heading, up to “Recording SMF trace data” on page 1193, is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page 1301.

DB2’s instrumentation facility component (IFC) provides a trace facility that you can use to record DB2 data and events. With the IFC, however, analysis and reporting of the trace records must take place outside of DB2. You can use OMEGAMON to format, print, and interpret DB2 trace output. You can view an online snapshot from trace records by using OMEGAMON or other online monitors. For more information on OMEGAMON, see *Using IBM Tivoli OMEGAMON XE on z/OS*. For the exact syntax of the trace commands see Chapter 2 of *DB2 Command Reference*.

If you do not have OMEGAMON, or if you want to do your own analysis of the DB2 trace output, refer to Appendix D, "Interpreting DB2 trace output," on page 1135. Also consider writing your own program using the instrumentation facility interface (IFI). Refer to Appendix E, "Programming for the Instrumentation Facility Interface (IFI)," on page 1151 for more information on using IFI.

Each *trace class* captures information on several subsystem events. These events are identified by many instrumentation facility component identifiers (IFCIDs). The IFCIDs are described by the comments in their mapping macros, contained in *prefix*.SDSNMACS, which is shipped to you with DB2.

Types of traces

DB2 trace can record six types of data: statistics, accounting, audit, performance, monitor, and global. The description of the START TRACE command in Chapter 2 of *DB2 Command Reference* indicates which IFCIDs are activated for the different types of trace and the classes within those trace types. For details on what information each IFCID returns, see the mapping macros in *prefix*.SDSNMACS.

The trace records are written using GTF or SMF records. See "Recording SMF trace data" on page 1193 and "Recording GTF trace data" on page 1195 before starting any traces. Trace records can also be written to storage, if you are using the monitor trace class.

Statistics trace

The statistics trace reports information about how much the DB2 system services and database services are used. It is a system-wide trace and should not be used for chargeback accounting. Use the information the statistics trace provides to plan DB2 capacity, or to tune the entire set of active DB2 programs.

Statistics trace classes 1, 3, 4, 5, and 6 are the default classes for the statistics trace if statistics is specified YES in panel DSNTIPN. If the statistics trace is started using the START TRACE command, then class 1 is the default class.

- Class 1 provides information about system services and database statistics. It also includes the system parameters that were in effect when the trace was started.
- Class 3 provides information about deadlocks and timeouts.
- Class 4 provides information about exceptional conditions.
- Class 5 provides information about data sharing.
- Class 6 provides storage statistics for the DBM1 address space.

If you specified YES in the SMF STATISTICS field on installation panel DSNTIPN, the statistics trace starts automatically when you start DB2, sending class 1, 3, 4 and 5 statistics data to SMF. SMF records statistics data in both SMF type 100 and 102 records. IFCIDs 0001, 0002, 0202, and 0230 are of SMF type 100. All other IFCIDs in statistics trace classes are of SMF type 102. From installation panel DSNTIPN, you can also control the statistics collection interval (STATISTICS TIME field).

The statistics trace is written on an interval basis, and you can control the exact time that statistics traces are taken.

Accounting trace

The DB2 accounting trace provides information related to application programs, including such things as:

Start and stop times
Number of commits and aborts
The number of times certain SQL statements are issued
Number of buffer pool requests
Counts of certain locking events
Processor resources consumed
Thread wait times for various events
RID pool processing
Distributed processing
Resource limit facility statistics

DB2 trace begins collecting this data at successful thread allocation to DB2, and writes a completed record when the thread terminates or when the authorization ID changes.

During CICS thread reuse, a change in the authid or transaction code initiates the sign-on process, which terminates the accounting interval and creates the accounting record. TXIDSO=NO eliminates the sign-on process when only the transaction code changes. When a thread is reused without initiating sign-on, several transactions are accumulated into the same accounting record, which can make it very difficult to analyze a specific transaction occurrence and correlate DB2 accounting with CICS accounting. However, applications that use ACCOUNTREC(UOW) or ACCOUNTREC(TASK) in the DBENTRY RDO definition initiate a "partial sign-on", which creates an accounting record for each transaction. You can use this data to perform program-related tuning and assess and charge DB2 costs.

Accounting data for class 1 (the default) is accumulated by several DB2 components during normal execution. This data is then collected at the end of the accounting period; it does not involve as much overhead as individual event tracing.

On the other hand, when you start class 2, 3, 7, or 8, many additional trace points are activated. Every occurrence of these events is traced internally by DB2 trace, but these traces are not written to any external destination. Rather, the accounting facility uses these traces to compute the additional total statistics that appear in the accounting record, IFCID 003, when class 2 or class 3 is activated. Accounting class 1 must be active to externalize the information.

To turn on accounting for packages and DBRMs, accounting trace classes 1 and 7 must be active. Though you can turn on class 7 while a plan is being executed, accounting trace information is only gathered for packages or DBRMs executed after class 7 is activated. Activate accounting trace class 8 with class 1 to collect information about the amount of time an agent was suspended in DB2 for each executed package. If accounting trace classes 2 and 3 are activated, there is minimal additional performance cost for activating accounting trace classes 7 and 8.

If you want information from either, or both, accounting class 2 and 3, be sure to activate class 2, class 3, or both classes before your application starts. If these classes are activated during the application, the times gathered by DB2 trace are only from the time the class was activated.

Accounting trace class 5 provides information on the amount of elapsed time and TCB time that an agent spent in DB2 processing instrumentation facility interface (IFI) requests. If an agent did not issue any IFI requests, these fields are not included in the accounting record.

If you specified YES for SMF ACCOUNTING on installation panel DSNTIPN, the accounting trace starts automatically when you start DB2, and sends IFCIDs that are of SMF type 101 to SMF. The accounting record IFCID 0003 is of SMF type 101.

Audit trace

The audit trace collects information about DB2 security controls and is used to ensure that data access is allowed only for authorized purposes. On the CREATE TABLE or ALTER TABLE statements, you can specify whether or not a table is to be audited, and in what manner; you can also audit security information such as any access denials, grants, or revokes for the table. The default causes no auditing to take place. For descriptions of the available audit classes and the events they trace, see "Audit class descriptions" on page 287.

If you specified YES for AUDIT TRACE on installation panel DSNTIPN, audit trace class 1 starts automatically when you start DB2. By default, DB2 will send audit data to SMF. SMF records audit data in type 102 records. When you invoke the -START TRACE command, you can also specify GTF as a destination for audit data. Chapter 13, "Auditing," on page 285 describes the audit trace in detail.

Performance trace

The performance trace provides information about a variety of DB2 events, including events related to distributed data processing. You can use this information to further identify a suspected problem, or to tune DB2 programs and resources for individual users or for DB2 as a whole.

You cannot automatically start collecting performance data when you install or migrate DB2. To trace performance data, you must use the -START TRACE(PERFM) command. For more information about the -START TRACE(PERFM) command, refer to Chapter 2 of *DB2 Command Reference*.

The performance trace defaults to GTF.

Monitor trace

The monitor trace records data for online monitoring with user-written programs. This trace type has several predefined classes; those that are used explicitly for monitoring are listed here:

- Class 1 (the default) allows any application program to issue an instrumentation facility interface (IFI) READS request to the IFI facility. If monitor class 1 is inactive, a READS request is denied. Activating class 1 has a minimal impact on performance.
- Class 2 collects processor and elapsed time information. The information can be obtained by issuing a READS request for IFCID 0147 or 0148. In addition, monitor trace class 2 information is available in the accounting record, IFCID 0003. Monitor class 2 is equivalent to accounting class 2 and results in equivalent overhead. Monitor class 2 times appear in IFCIDs 0147, 0148, and 0003 if either monitor trace class 2 or accounting class 2 is active.
- Class 3 activates DB2 wait timing and saves information about the resource causing the wait. The information can be obtained by issuing a READS request for IFCID 0147 or 0148. In addition, monitor trace class 3 information is available

in the accounting record, IFCID 0003. As with monitor class 2, monitor class 3 overhead is equivalent to accounting class 3 overhead.

When monitor trace class 3 is active, DB2 can calculate the duration of a class 3 event, such as when an agent is suspended due to an unavailable lock. Monitor class 3 times appear in IFCIDs 0147, 0148, and 0003, if either monitor class 3 or accounting class 3 is active.

- Class 5 traces the amount of time spent processing IFI requests.
- Class 7 traces the amount of time an agent spent in DB2 to process each package. If monitor trace class 2 is active, activating class 7 has minimal performance impact.
- Class 8 traces the amount of time an agent was suspended in DB2 for each package executed. If monitor trace class 3 is active, activating class 8 has minimal performance impact.

For more information on the monitor trace, refer to Appendix E, "Programming for the Instrumentation Facility Interface (IFI)," on page 1151.

Effect on DB2 performance

The volume of data DB2 trace collects can be quite large. Consequently, the number of trace records you request will affect system performance. In particular, when you activate a performance trace, you should qualify the START TRACE command with the particular classes, plans, authorization IDs, and IFCIDs you want to trace.

The following recommendations apply:

- When starting a performance trace, be sure that you know what you want to report, for example, I/O only or SQL only. See OMEGAMON for examples of which classes produce which reports. Otherwise, you might have incomplete reports and have to rerun or collect too much data, overloading the data collector.
- When the statistics trace is active, statistics are collected by SMF at all times. Use the default statistics frequency of 30 minutes.
- Decide if the continuous collection of accounting data is needed. If a transaction manager provides enough accounting information, DB2 accounting might not be needed. In environments where the processor is heavily loaded, consider not running accounting on a continuous basis.
- When using accounting on a continuous basis, start classes 1 and 3 to SMF (SMF ACCOUNTING on installation panel DSNTIPN). You might also want to start accounting class 2 because it provides additional information that can be useful in resolving problems. Accounting class 2 does introduce some additional processor cost.
- Use the performance trace for short periods of time (START/STOP TRACE) and restrict it to certain users, applications, and classes. Use the default destination GTF to allow immediate analysis of the trace information.
- Start the global trace only if a problem is under investigation, and IBM service personnel have requested a trace.

For more detailed information about the amount of processor resources consumed by DB2 trace, see "Reducing processor resource consumption" on page 659.

Recording SMF trace data

Each location is responsible for processing the SMF records produced by DB2 trace.

For example, during DB2 execution, you can use the z/OS operator command SETSMF or SS to alter SMF parameters that you specified previously. The following command records statistics (record type 100), accounting (record type 101), and performance (record type 102) data to SMF. To execute this command, specify PROMPT(ALL) or PROMPT(LIST) in the SMFPRM*xx* member used from SYS1.PARMLIB.

```
SETSMF SYS(TYPE(100:102))
```

If you are not using measured usage licensing, do not specify type 89 records or you will incur the overhead of collecting that data.

You can use the SMF program IFASMFDP to dump these records to a sequential data set. You might want to develop an application or use OMEGAMON to process these records. For a sample DB2 trace record sent to SMF, see Figure 155 on page 1137. For more information about SMF, refer to *z/OS JES2 Initialization and Tuning Guide*.

Activating SMF

SMF must be running before you can send data to it. To make it operational, update member SMFPRM*xx* of SYS1.PARMLIB, which indicates whether SMF is active and which types of records SMF accepts. For member SMFPRM*xx*, *xx* are two user-defined alphanumeric characters appended to 'SMFPRM' to form the name of an SMFPRM*xx* member. To update this member, specify the ACTIVE parameter and the proper TYPE subparameter for SYS and SUBSYS.

You can also code an IEFU84 SMF exit to process the records that are produced.

Allocating additional SMF buffers

When you specify a performance trace type, the volume of data that DB2 can collect can be quite large. If you are sending this data to SMF, you must allocate adequate SMF buffers; the default buffer settings will probably be insufficient.

If an SMF buffer shortage occurs, SMF rejects any trace records sent to it. DB2 sends a message (DSNW133I) to the MVS operator when this occurs. DB2 treats the error as temporary and remains active even though data could be lost. DB2 sends another message (DSNW123I) to the z/OS operator when the shortage has been alleviated and trace recording has resumed.

You can determine if trace data has been lost by examining the DB2 statistics records with an IFCID of 0001, as mapped by macro DSNQWST. These records show:

- The number of trace records successfully written
- The number of trace records that could not be written
- The reason for the failure

If your location uses SMF for performance data or global trace data, be sure that:

- Your SMF data sets are large enough to hold the data.
- SMF is set up to accept record type 102. (Specify member SMFPRM*xx*, for which ’xx’ are two user-defined alphanumeric characters.)
- Your SMF buffers are large enough.

Specify SMF buffering on the VSAM BUFSP parameter of the access method services DEFINE CLUSTER statement. Do not use the default settings if DB2 performance or global trace data is sent to SMF. Specify CISZ(4096) and

BUFSP(81920) on the DEFINE CLUSTER statement for each SMF VSAM data set. These values are the minimum required for DB2; you might have to increase them, depending on your z/OS environment.

DB2 runs above the 16MB line of virtual storage in a cross-memory environment.

Reporting data in SMF

There are several ways to report trace records sent to SMF:

- Use Tivoli Decision Support for OS/390 to collect the data and create graphical or tabular reports.
- Write an application program to read and report information from the SMF data set. You can tailor it to fit your exact needs.
- Use OMEGAMON. See "OMEGAMON" on page 1196 for a discussion of OMEGAMON's capabilities.

In any of those ways you can compare any report for a current day, week, or month with an equivalent sample, as far back as you want to go. The samples become more widely spaced but are still available for analysis.

Recording GTF trace data

The default destination for the performance trace classes is the generalized trace facility (GTF). The z/OS operator must start GTF before you can send data to it. When starting GTF, specify TIME=YES, and then TRACE=USRP. Start GTF as indicated in Table 311 to ensure that offsets map correctly. Be sure that no GTF member exists in SYS1.PARMLIB.

Table 311. Recording GTF trace data

You enter...	System responds...
S GTF,,,(TIME=YES)	AHL100A SPECIFY TRACE OPTIONS
TRACE=USRP	AHL101A SPECIFY TRACE EVENT KEYWORDS --USR=
USR=(FB9)	AHL102A CONTINUE TRACE DEFINITION OR REPLY END
END	AHL125A RESPECIFY TRACE OPTIONS OR REPLY U
U	AHL031I GTF INITIALIZATION COMPLETE

Note: To make stopping GTF easier, you can name the GTF session when you start it. For example, you could specify `S GTF.GTF,,,(TIME=YES)`.

If a GTF member exists in SYS1.PARMLIB, the GTF trace option USR might not be in effect. When no other member exists in SYS1.PARMLIB, you are sure to have only the USR option activated, and no other options that might add unwanted data to the GTF trace.

When starting GTF, if you use the JOBNAMEP option to obtain only those trace records written for a specific job, trace records written for other agents are not written to the GTF data set. This means that a trace record that is written by a system agent that is processing for an allied agent is discarded if the JOBNAMEP option is used. For example, after a DB2 system agent performs an IDENTIFY request for an allied agent, an IFCID record is written. If the JOBNAMEP keyword is used to collect trace data for a specific job, however, the record for the IDENTIFY request is not written to GTF, even if the IDENTIFY request was performed for the job named on the JOBNAMEP keyword.

You can record DB2 trace data in GTF using a GTF event ID of X'FB9'.

Trace records longer than the GTF limit of 256 bytes are spanned by DB2. For instructions on how to process GTF records, refer to Appendix D, “Interpreting DB2 trace output,” on page 1135.

OMEGAMON

OMEGAMON provides performance monitoring, reporting, buffer pool analysis, and a performance warehouse all in one tool:

- OMEGAMON includes the function of DB2 Performance Monitor (DB2 PM), which is also available as a stand-alone product. Both products report DB2 instrumentation in a form that is easy to understand and analyze. The instrumentation data is presented in the following ways:
  - The Batch report sets present the data you select in comprehensive reports or graphs containing system-wide and application-related information for both single DB2 subsystems and DB2 members of a data sharing group. You can combine instrumentation data from several different DB2 locations into one report.

    Batch reports can be used to examine performance problems and trends over a period of time.
  - The Online Monitor gives a current “snapshot” view of a running DB2 subsystem, including applications that are running. Its history function displays information about subsystem and application activity in the recent past.

    Both a host-based and Workstation Online Monitor are provided. The Workstation Online Monitor substantially improves usability, simplifies online monitoring and problem analysis, and offers significant advantages. For example, from Workstation Online Monitor, you can launch Visual Explain so you can examine the access paths and processing methods chosen by DB2 for the currently executing SQL statement.

    For more information about the Workstation Online Monitor, see *OMEGAMON Monitoring Performance from Performance Expert Client* or *OMEGAMON for z/OS and Multiplatforms Monitoring Performance from Workstation for z/OS and Multiplatforms.*

  In addition, OMEGAMON contains a Performance Warehouse function that lets you:
  - Save DB2 trace and report data in a performance database for further investigation and trend analysis
  - Configure and schedule the report and load process from the workstation interface
  - Define and apply analysis functions to identify performance bottlenecks.
- OMEGAMON also includes the function of DB2 Buffer Pool Analyzer, which is also available as a stand-alone product. Both products help you optimize buffer pool usage by offering comprehensive reporting of buffer pool activity, including:
  - Ordering by various identifiers such as buffer pool, plan, object, and primary authorization ID
  - Sorting by getpage, sequential prefetch, and synchronous read
  - Filtering capability

In addition, you can simulate buffer pool usage for varying buffer pool sizes and analyze the results of the simulation reports to determine the impact of any changes before making those changes to your current system.

Tivoli Decision Support for OS/390

Tivoli Decision Support for OS/390, formerly known as Tivoli Performance Reporter for OS/390, collects data into a DB2 database and allows you to create graphical and tabular reports to use in managing systems performance. The data can come from different sources, including SMF, the IMS log, the CICS journal, RMF, and DB2.

When considering the use of Tivoli Decision Support for OS/390, consider the following:

- Tivoli Decision Support data collection and reporting are based on user specifications. Therefore, an experienced user can produce more suitable reports than the predefined reports produced by other tools.
- Tivoli Decision Support provides historical performance data that you can use to compare a current situation with previous data.
- Tivoli Decision Support can be used very effectively for reports based on the DB2 statistics and accounting records. When using it for the performance trace consider that:
  - Because of the large number of different DB2 performance records, a substantial effort is required to define their formats to Tivoli Decision Support. Changes in the records require review of the definitions.
  - Tivoli Decision Support does not handle information from paired records, such as “start event” and “end event.” These record pairs are used by OMEGAMON to calculate elapsed times, such as the elapsed time of I/Os and lock suspensions.

The general recommendation for Tivoli Decision Support and OMEGAMON use in a DB2 subsystem is:

- If Tivoli Decision Support is already used or there is a plan to use it at the location:
  - Extend Tivoli Decision Support usage to the DB2 accounting and statistics records.
  - Use OMEGAMON for the DB2 performance trace.
- If Tivoli Decision Support is not used and there is no plan to use it:
  - Use OMEGAMON for the statistics, accounting, and performance trace.
  - Consider extending OMEGAMON with user applications based on DB2 and QMF, to provide historical performance data.

Monitoring application plans and packages

The following statements identify plans and packages that:

- Possibly redo validity checks at run time; if an invalid object or missing authority is found, DB2 issues a warning and checks again for the object or authorization at run time.
- Use repeatable read.
- Are invalid (must be rebound before use), for example, the deleting an index or revoking authority can render a plan or package invalid.
- Are inoperative (require an explicit BIND or REBIND before use). A plan or package can be marked inoperative after an unsuccessful REBIND.

General-use Programming Interface

```
SELECT NAME, VALIDATE, ISOLATION, VALID, OPERATIVE
  FROM SYSIBM.SYSPLAN
  WHERE VALIDATE = 'R' OR ISOLATION = 'R'
    OR VALID = 'N' OR OPERATIVE = 'N';

SELECT COLLID, NAME, VERSION, VALIDATE, ISOLATION, VALID, OPERATIVE
  FROM SYSIBM.SYSPACKAGE
  WHERE VALIDATE = 'R' OR ISOLATION = 'R'
    OR VALID = 'N' OR OPERATIVE = 'N';
```

End of General-use Programming Interface

Appendix G. EXPLAIN tables that are used by optimization tools

Optimization tools such as IBM Optimization Service Center for DB2 for z/OS, IBM DB2 Optimization Expert for z/OS, and the DB2 EXPLAIN function might create and use any of the following EXPLAIN tables to store performance data.

DSN_PREDICAT_TABLE

The predicate table, DSN_PREDICAT_TABLE, contains information about all of the predicates in a query.

Recommendation: Do not manually insert data into or delete data from EXPLAIN tables. The data is intended to be manipulated only by the DB2 EXPLAIN function and various optimization tools.

CREATE TABLE statement

The following figure shows the current format of DSN_PREDICAT_TABLE

```
 CREATE TABLE userid.DSN_PREDICAT_TABLE
  ( QUERYNO            INTEGER      NOT NULL,
    QBLOCKNO           SMALLINT     NOT NULL,
    APPLNAME           VARCHAR(24)      NOT NULL,
    PROGNAME           VARCHAR(128) NOT NULL,
    PREDNO             INTEGER      NOT NULL,
    TYPE               CHAR(8)      NOT NULL,
    LEFT_HAND_SIDE     VARCHAR(128) NOT NULL,
    LEFT_HAND_PNO      INTEGER      NOT NULL,
    LHS_TABNO          SMALLINT     NOT NULL,
    LHS_QBNO           SMALLINT     NOT NULL,
    RIGHT_HAND_SIDE    VARCHAR(128) NOT NULL,
    RIGHT_HAND_PNO     INTEGER      NOT NULL,
    RHS_TABNO          SMALLINT     NOT NULL,
    RHS_QBNO           SMALLINT     NOT NULL,
    FILTER_FACTOR      FLOAT        NOT NULL,
    BOOLEAN_TERM       CHAR(1)      NOT NULL,
    SEARCHARG          CHAR(1)      NOT NULL,
    JOIN               CHAR(1)      NOT NULL,
    AFTER_JOIN         CHAR(1)      NOT NULL,
    ADDED_PRED         CHAR(1)      NOT NULL,
    REDUNDANT_PRED     CHAR(1)      NOT NULL,
    DIRECT_ACCESS      CHAR(1)      NOT NULL,
    KEYFIELD           CHAR(1)      NOT NULL,
    EXPLAIN_TIME       TIMESTAMP    NOT NULL,
    CATEGORY           SMALLINT     NOT NULL,
    CATEGORY_B         SMALLINT     NOT NULL,
    TEXT               VARCHAR(2000) NOT NULL,
    PRED_ENCODE        CHAR(1)      NOT NULL WITH DEFAULT,
    PRED_CCSID         SMALLINT     NOT NULL WITH DEFAULT,
    PRED_MCCSID        SMALLINT     NOT NULL WITH DEFAULT,
    MARKER             CHAR(1)      NOT NULL WITH DEFAULT,
    PARENT_PNO         INTEGER      NOT NULL,
    NEGATION           CHAR(1)      NOT NULL,
    LITERALS           VARCHAR(128) NOT NULL,
    CLAUSE             CHAR(8)      NOT NULL,
    GROUP_MEMBER       VARCHAR(24)      NOT NULL
  ) in database-name.table-space-name
CCSID UNICODE;
```

© Copyright IBM Corp. 1982, 2008

Figure 166. The CREATE TABLE statement for userid.DSN_PREDICAT_TABLE

Column descriptions

The following table describes the columns of the DSN_PREDICAT_TABLE

Table 312. DSN_PREDICAT_TABLE description

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL	The query number, a number used to help identify the query being explained. It is not a unique identifier. Using negative number will cause problems. The possible sources are: **1** The statement line number in the program **2** QUERYNO clause **3** The EXPLAIN statement **4** EDM unique token in the statement cache
QBLOCKNO	SMALLINT NOT NULL	The query block number, a number used to identify each query block within a query.
APPLNAME	VARCHAR(24) NOT NULL	The application plan name.
PROGNAME	VARCHAR(128) NOT NULL	The program name (binding an application) or the package name (binding a package).
PREDNO	INTEGER NOT NULL	The predicate number, a number used to identify a predicate within a query.
TYPE	CHAR(8) NOT NULL	A string used to indicate the type or the operation of the predicate. The possible values are: • 'AND' • 'OR' • 'EQUAL' • 'RANGE' • 'BETWEEN' • 'IN' • 'LIKE' • 'NOT LIKE' • 'EXISTS • 'NOTEXIST' • 'SUBQUERY' • 'HAVING' • 'OTHERS'

Table 312. DSN_PREDICAT_TABLE description (continued)

Column name	Data type	Description
LEFT_HAND_SIDE	VARCHAR(128) NOT NULL	If the LHS of the predicate is a table column (LHS_TABNO > 0), then this column indicates the column name. Other possible values are: • 'VALUE' • 'COLEXP' • 'NONCOLEXP' • 'CORSUB' • 'NONCORSUB' • 'SUBQUERY' • 'EXPRESSION' • Blanks
LEFT_HAND_SIDE	VARCHAR(128)	If the LHS of the predicate is a table column (LHS_TABNO > 0), then this column indicates the column name. Other possible values are: • 'VALUE' • 'COLEXP' • 'NONCOLEXP' • 'CORSUB' • 'NONCORSUB' • 'SUBQUERY' • 'EXPRESSION' • Blanks
LEFT_HAND_PNO	INTEGER NOT NULL	If the LHS of the predicate is a table column (LHS_TABNO > 0), then this column indicates the column name. Other possible values are: • 'VALUE' • 'COLEXP' • 'NONCOLEXP' • 'CORSUB' • 'NONCORSUB' • 'SUBQUERY' • 'EXPRESSION' • Blanks
LHS_TABNO	SMALLINT NOT NULL	If the LHS of the predicate is a table column, then this column indicates a number which uniquely identifies the corresponding table reference within a query.
LHS_QBNO	SMALLINT NOT NULL	If the LHS of the predicate is a table column, then this column indicates a number which uniquely identifies the corresponding table reference within a query.

Table 312. DSN_PREDICAT_TABLE description (continued)

Column name	Data type	Description
RIGHT_HAND_SIDE	VARCHAR(128) NOT NULL	If the RHS of the predicate is a table column (RHS_TABNO > 0), then this column indicates the column name. Other possible values are: • 'VALUE' • 'COLEXP' • 'NONCOLEXP' • 'CORSUB' • 'NONCORSUB' • 'SUBQUERY' • 'EXPRESSION' • Blanks
RIGHT_HAND_PNO	INTEGER NOT NULL	If the predicate is a compound predicate (AND/OR), then this column indicates the second child predicate. However, this column is not reliable when the predicate tree consolidation happens. Use PARENT_PNO instead to reconstruct the predicate tree.
RHS_TABNO	CHAR(1) NOT NULL	If the RHS of the predicate is a table column, then this column indicates a number which uniquely identifies the corresponding table reference within a query.
RHS_QBNO	CHAR(1) NOT NULL	If the RHS of the predicate is a subquery, then this column indicates a number which uniquely identifies the corresponding query block within a query.
FILTER_FACTOR	CHAR(1) NOT NULL	The estimated filter factor.
FILTER_FACTOR	CHAR(1) NOT NULL	Whether this predicate can be used to determine the truth value of the whole WHERE clause.
BOOLEAN_TERM	CHAR(1) NOT NULL	Whether this predicate can be used to determine the truth value of the whole WHERE clause.
SEARCHARG	CHAR(1) NOT NULL	Whether this predicate can be processed by data manager (DM). If it is not, then the relational data service (RDS) needs to be used to take care of it, which is more costly.
AFTER_JOIN	CHAR(1) NOT NULL	Indicates the predicate evaluation phase: **'A'** After join **'D'** During join **blank** Not applicable
ADDED_PRED	CHAR(1) NOT NULL	Whether it is generated by transitive closure, which means DB2 can generate additional predicates to provide more information for access path selection, when the set of predicates that belong to a query logically imply other predicates.
REDUNDANT_PRED	CHAR(1) NOT NULL	Whether it is a redundant predicate, which means evaluation of other predicates in the query already determines the result that the predicate provides.
DIRECT_ACCESS	CHAR(1) NOT NULL	Whether the predicate is direct access, which means one can navigate directly to the row through ROWID.
KEYFIELD	CHAR(1) NOT NULL	Whether the predicate includes the index key column of the involved table.
EXPLAIN_TIME	TIMESTAMP	The EXPLAIN timestamp.
CATEGORY	SMALLINT	IBM internal use only.

Table 312. DSN_PREDICAT_TABLE description (continued)

Column name	Data type	Description
CATEGORY_B	SMALLINT	IBM internal use only.
TEXT	VARCHAR(2000)	The transformed predicate text; truncated if exceeds 2000 characters.
PRED_ENCODE	CHAR(1)	IBM internal use only.
PRED_CCSID	SMALLINT	IBM internal use only.
PRED_MCCSID	SMALLINT	IBM internal use only.
MARKER	CHAR(1)	Whether this predicate includes host variables, parameter markers, or special registers.
PARENT_PNO	INTEGER	The parent predicate number. If this predicate is a root predicate within a query block, then this column is 0.
NEGATION	CHAR(1)	Whether this predicate is negated via NOT.
LITERALS	VARCHAR(128)	This column indicates the literal value or literal values separated by colon symbols.
CLAUSE	CHAR(8)	The clause where the predicate exists: **'HAVING '** The HAVING clause **'ON '** The ON clause **'WHERE '** The WHERE clause
GROUP_MEMBER	CHAR(8)	The member name of the DB2 that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.

DSN_STRUCT_TABLE

The structure table, DSN_STRUCT_TABLE, contains information about all of the query blocks in a query.

Recommendation: Do not manually insert data into or delete data from EXPLAIN tables. The data is intended to be manipulated only by the DB2 EXPLAIN function and various optimization tools.

CREATE TABLE statement

The following figure shows the CREATE TABLE statement for DSN_STRUCT_TABLE.

```
CREATE TABLE userid.DSN_STRUCT_TABLE
  ( QUERYNO            INTEGER      NOT NULL,
    QBLOCKNO           SMALLINT     NOT NULL,
    APPLNAME           VARCHAR(24)      NOT NULL,
    PROGNAME           VARCHAR(128) NOT NULL,
    PARENT             SMALLINT     NOT NULL,
    TIMES              FLOAT        NOT NULL,
    ROWCOUNT           INTEGER      NOT NULL,
    ATOPEN             CHAR(1)      NOT NULL,
    CONTEXT            CHAR(10)     NOT NULL,
    ORDERNO            SMALLINT     NOT NULL,
    DOATOPEN_PARENT    SMALLINT     NOT NULL,
    QBLOCK_TYPE        CHAR(6)      NOT NULL WITH DEFAULT,
    EXPLAIN_TIME       TIMESTAMP    NOT NULL,
    QUERY_STAGE        CHAR(8)      NOT NULL,
```

```
    GROUP_MEMBER       VARCHAR(24)      NOT NULL
  )
   IN database-name.table-space-name
CCSID UNICODE;
```

Figure 167. The CREATE TABLE statement for userid.DSN_STRUCT_TABLE

Column descriptions

The following table describes the columns of DSN_STRUCT_TABLE

Table 313. DSN_STRUCT_TABLE description

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL	The query number, a number used to help identify the query being explained. It is not a unique identifier. Using negative number will cause problems. The possible sources are: **1** The statement line number in the program **2** QUERYNO clause **3** The EXPLAIN statement **4** EDM unique token in the statement cache
QBLOCKNO	SMALLINT NOT NULL	The query block number, a number used to identify each query block within a query.
APPLNAME	VARCHAR(24) NOT NULL	The application plan name.
PROGNAME	VARCHAR(128) NOT NULL	The program name (binding an application) or the package name (binding a package).
PARENT	SMALLINT NOT NULL	The parent query block number of the current query block in the structure of SQL text; this is the same as the PARENT_QBLOCKNO in PLAN_TABLE.
TIMES	FLOAT NOT NULL	The estimated number of rows returned by Data Manager; also the estimated number of times this query block is executed.
ROWCOUNT	INTEGER NOT NULL	The estimated number of rows returned by RDS (Query Cardinality).
ATOPEN	CHAR(1) NOT NULL	Whether the query block is moved up for do-at-open processing; 'Y' if done-at-open; 'N': otherwise.
CONTEXT	CHAR(10) NOT NULL	This column indicates what the context of the current query block is. The possible values are: • 'TOP LEVEL' • 'UNION' • 'UNION ALL' • 'PREDICATE' • 'TABLE EXP' • 'UNKNOWN'
ORDERNO	SMALLINT NOT NULL	Not currently used.
DOATOPEN_PARENT	SMALLINT NOT NULL	The parent query block number of the current query block; Do-at-open parent if the query block is done-at-open, this may be different from the PARENT_QBLOCKNO in PLAN_TABLE.

Table 313. DSN_STRUCT_TABLE description (continued)

Column name	Data type	Description
QBLOCK_TYPE	CHAR(6) NOT NULL WITH DEFAULT	This column indicates the type of the current query block. The possible values are • 'SELECT' • 'INSERT' • 'UPDATE' • 'DELETE' • 'SELUPD' • 'DELCUR' • ''UPDCUR' • 'CORSUB' • 'NCOSUB' • 'TABLEX' • 'TRIGGR' • 'UNION' • 'UNIONA' • 'CTE' It is equivalent to QBLOCK_TYPE column in PLAN_TABLE, except for CTE.
EXPLAIN_TIME	TIMESTAMP NOT NULL	The EXPLAIN timestamp.
QUERY_STAGE	CHAR(8) NOT NULL	IBM internal use only.
GROUP_MEMBER	CHAR(8) NOT NULL	The member name of the DB2 subsystem that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.

DSN_PGROUP_TABLE

The parallel group table, PGROUP_TABLE, contains information about the parallel groups in a query.

Recommendation: Do not manually insert data into or delete data from EXPLAIN tables. The data is intended to be manipulated only by the DB2 EXPLAIN function and various optimization tools.

CREATE TABLE statement

The following figure shows the format of DSN_PGROUP_TABLE

```
CREATE TABLE userid.DSN_PGROUP_TABLE
  ( QUERYNO            INTEGER      NOT NULL,
    QBLOCKNO           SMALLINT     NOT NULL,
    PLANNAME           VARCHAR(24)      NOT NULL,
    COLLID             VARCHAR(128) NOT NULL,
    PROGNAME           VARCHAR(128) NOT NULL,
    EXPLAIN_TIME       TIMESTAMP    NOT NULL,
    VERSION            VARCHAR(122)  NOT NULL,
    GROUPID            SMALLINT     NOT NULL,
    FIRSTPLAN          SMALLINT     NOT NULL,
    LASTPLAN           SMALLINT     NOT NULL,
    CPUCOST            REAL         NOT NULL,
    IOCOST             REAL         NOT NULL,
```

```
    BESTTIME          REAL         NOT NULL,
    DEGREE            SMALLINT     NOT NULL,
    MODE              CHAR(1)      NOT NULL,
    REASON            SMALLINT     NOT NULL,
    LOCALCPU          SMALLINT     NOT NULL,
    TOTALCPU          SMALLINT     NOT NULL,
    FIRSTBASE         SMALLINT,
    LARGETS           CHAR(1),
    PARTKIND          CHAR(1),
    GROUPTYPE         CHAR(3),
    ORDER             CHAR(1),
    STYLE             CHAR(4),
    RANGEKIND         CHAR(1),
    NKEYCOLS          SMALLINT,
    LOWBOUND          VARCHAR(40),
    HIGHBOUND         VARCHAR(40),
    LOWKEY            VARCHAR(40),
    HIGHKEY           VARCHAR(40),
    FIRSTPAGE         CHAR(4),
    LASTPAGE          CHAR(4),
    GROUP_MEMBER      VARCHAR(24)      NOT NULL,
    HOST_REASON       SMALLINT,
    PARA_TYPE         CHAR(4),
    PART_INNER        CHAR(1),
    GRNU_KEYRNG       CHAR(1),
    OPEN_KEYRNG       CHAR(1)
  ) IN database-name.table-space-name

CCSID UNICODE;
```

Figure 168. The CREATE TABLE statement for userid.DSN_PGROUP_TABLE

Column descriptions

The following table describes the columns of DSN_PGROUP_TABLE

Table 314. DSN_PGROUP_TABLE description

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL	The query number, a number used to help identify the query being explained. It is not a unique identifier. Using negative number will cause problems. The possible sources are: **1** The statement line number in the program **2** QUERYNO clause **3** The EXPLAIN statement **4** EDM unique token in the statement cache
QBLOCKNO	SMALLINT NOT NULL	The query block number, a number used to identify each query block within a query.
PLANNAME	VARCHAR(24) NOT NULL	The application plan name.
COLLID	VARCHAR(128) NOT NULL	The collection ID for the package.
PROGNAME	VARCHAR(128) NOT NULL	The program name (binding an application) or the package name (binding a package).
EXPLAIN_TIME	TIMESTAMP NOT NULL	The explain timestamp.
VERSION	VARCHAR(122) NOT NULL	The version identifier for the package.

Table 314. DSN_PGROUP_TABLE description (continued)

Column name	Data type	Description
GROUPID	SMALLINT NOT NULL	The parallel group identifier within the current query block.
FIRSTPLAN	SMALLINT NOT NULL	The plan number of the first contributing mini-plan associated within this parallel group.
LASTPLAN	SMALLINT NOT NULL	The plan number of the last mini-plan associated within this parallel group.
CPUCOST	REAL NOT NULL	The estimated total CPU cost of this parallel group in milliseconds.
IOCOST	REAL NOT NULL	The estimated total I/O cost of this parallel group in milliseconds.
BESTTIME	REAL NOT NULL	The estimated elapsed time for each parallel task for this parallel group.
DEGREE	SMALLINT NOT NULL	The degree of parallelism for this parallel group determined at bind time. Max parallelism degree if the Table space is large is 255, otherwise 64.
MODE	CHAR(1) NOT NULL	The parallel mode: **'I'** IO parallelism **'C'** CPU parallelism **'X'** multiple CPU SyspleX parallelism (highest level) **'N'** No parallelism
REASON	SMALLINT NOT NULL	The reason code for downgrading parallelism mode.
LOCALCPU	SMALLINT NOT NULL	The number of CPUs currently online when preparing the query.
TOTALCPU	SMALLINT NOT NULL	The total number of CPUs in Sysplex. LOCALCPU and TOTALCPU are different only for the DB2 coordinator in a Sysplex.
FIRSTBASE	SMALLINT	The table number of the table that partitioning is performed on.
LARGETS	CHAR(1)	'Y' if the TableSpace is large in this group.
PARTKIND	CHAR(1)	The partitioning type: **'L'** Logical partitioning **'P'** Physical partitioning
GROUPTYPE	CHAR(3)	Determines what operations this parallel group contains: table Access, Join, or Sort 'A' 'AJ' 'AJS'
ORDER	CHAR(1)	The ordering requirement of this parallel group : **'N'** No order. Results need no ordering. **'T'** Natural Order. Ordering is required but results already ordered if accessed via index. **'K'** Key Order. Ordering achieved by sort. Results ordered by sort key. This value applies only to parallel sort.

Table 314. DSN_PGROUP_TABLE description (continued)

Column name	Data type	Description
STYLE	CHAR(4)	The Input/Output format style of this parallel group. Blank for IO Parallelism. For other modes: **'RIRO'** Records IN, Records OUT **'WIRO'** Work file IN, Records OUT **'WIWO'** Work file IN, Work file OUT
RANGEKIND	CHAR(1)	The range type: **'K'** Key range **'P'** Page range
NKEYCOLS	SMALLINT	The number of interesting key columns, that is, the number of columns that will participate in the key operation for this parallel group.
LOWBOUND	VARCHAR(40)	The low bound of parallel group.
HIGHBOUND	VARCHAR(40)	The high bound of parallel group.
LOWKEY	VARCHAR(40)	The low key of range if partitioned by key range.
HIGHKEY	VARCHAR(40)	The high key of range if partitioned by key range.
FIRSTPAGE	CHAR(4)	The first page in range if partitioned by page range.
LASTPAGE	CHAR(4)	The last page in range if partitioned by page range.
GROUP_MEMBER	CHAR(8) NOT NULL	IBM internal use only.
HOST_REASON	SMALLINT	IBM internal use only.
PARA_TYPE	CHAR(4)	IBM internal use only.
PART_INNER	CHAR(1)	IBM internal use only.
GRNU_KEYRNG	CHAR(1)	IBM internal use only.
OPEN_KEYRNG	CHAR(1)	IBM internal use only.

DSN_PTASK_TABLE

The parallel tasks table, DSN_PTASK_TABLE, contains information about all of the parallel tasks in a query.

Recommendation: Do not manually insert data into or delete data from EXPLAIN tables. The data is intended to be manipulated only by the DB2 EXPLAIN function and various optimization tools.

CREATE TABLE statement

The following figure shows the format of DSN_PTASK_TABLE.

```
CREATE TABLE userid.DSN_PTASK_TABLE
      ( QUERYNO            INTEGER      NOT NULL,
        QBLOCKNO           SMALLINT     NOT NULL,
        PGDNO              SMALLINT     NOT NULL,
        APPLNAME           VARCHAR(24)  NOT NULL,
        PROGNAME           VARCHAR(128) NOT NULL,
        LPTNO              SMALLINT     NOT NULL,
        KEYCOLID           SMALLINT     ,
```

```
        DPSI              CHAR(1)      NOT NULL,
        LPTLOKEY          VARCHAR(40)  ,
        LPTHIKEY          VARCHAR(40)  ,
        LPTLOPAG          CHAR(4)      ,
        LPTHIPAG          CHAR(4)      ,
        LPTLOPG#          CHAR(4)      ,
        LPTHIPG#          CHAR(4)      ,
        LPTLOPT#          SMALLINT     ,
        LPTHIPT#          SMALLINT     ,
        KEYCOLDT          SMALLINT     ,
        KEYCOLPREC        SMALLINT     ,
        KEYCOLSCAL        SMALLINT     ,
        EXPLAIN_TIME      TIMESTAMP    NOT NULL,
        GROUP_MEMBER      VARCHAR(24)  NOT NULL
   ) IN database-name.table-space-name
CCSID UNICODE;
```

Figure 169. The CREATE TABLE statement for userid.DSN_PTASK_TABLE

Column descriptions

The following table describes the columns of DSN_PTASK_TABLE.

Table 315. DSN_PTASK_TABLE description

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL	The query number, a number used to help identify the query being explained. It is not a unique identifier. Using negative number will cause problems. The possible sources are: **1** The statement line number in the program **2** QUERYNO clause **3** The EXPLAIN statement **4** EDM unique token in the statement cache
QBLOCKNO	SMALLINT NOT NULL	The query block number, a number used to identify each query block within a query.
APPLNAME	VARCHAR(24) NOT NULL	The application plan name.
PROGNAME	VARCHAR(128) NOT NULL	The program name (binding an application) or the package name (binding a package).
LPTNO	SMALLINT NOT NULL	The parallel task number.
KEYCOLID	SMALLINT	The key column ID (KEY range only).
DPSI	CHAR(1) NOT NULL	Indicates if a data partition secondary index (DPSI) is used.
LPTLOKEY	VARCHAR(40)	The low key value for this key column for this parallel task (KEY range only).
LPTHIKEY	VARCHAR(40)	The high key value for this key column for this parallel task (KEY range only).
LPTLOPAG	CHAR(4)	The low page information if partitioned by page range.
LPTLHIPAG	CHAR(4)	The high page information if partitioned by page range.
LPTLOPG#	CHAR(4)	The lower bound page number for this parallel task (Page range or DPSI enabled only).

Table 315. DSN_PTASK_TABLE description (continued)

Column name	Data type	Description
LPTHIPG#	CHAR(4)	The upper bound page number for this parallel task (Page range or DPSI enabled only).
LPTLOPT#	SMALLINT	The lower bound partition number for this parallel task (Page range or DPSI enabled only).
KEYCOLDT	SMALLINT	The data type for this key column (KEY range only).
KEYCOLPREC	SMALLINT	The precision/length for this key column (KEY range only).
KEYCOLSCAL	SMALLINT	The scale for this key column (KEY range with Decimal datatype only).
EXPLAIN_TIME	TIMESTAMP NOT NULL	The EXPLAIN timestamp.
GROUP_MEMBER	VARCHAR(24) NOT NULL	The member name of the DB2 that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.

DSN_FILTER_TABLE

The filter table, DSN_FILTER_TABLE, contains information about how predicates are used during query processing.

Recommendation: Do not manually insert data into or delete data from EXPLAIN tables. The data is intended to be manipulated only by the DB2 EXPLAIN function and various optimization tools.

CREATE TABLE statement

The following figure shows the format of DSN_FILTER_TABLE.

```
CREATE TABLE userid.DSN_FILTER_TABLE
   (
    QUERYNO                INTEGER      NOT NULL,
    QBLOCKNO               SMALLINT     NOT NULL,
    PLANNO                 SMALLINT     NOT NULL,
    APPLNAME               VARCHAR(24)  NOT NULL,
    PROGNAME               VARCHAR(128) NOT NULL,
    COLLID                 VARCHAR(128) NOT NULL WITH DEFAULT,
    ORDERNO                INTEGER      NOT NULL,
    PREDNO                 INTEGER      NOT NULL,
    STAGE                  CHAR(9)      NOT NULL,
    ORDERCLASS             INTEGER      NOT NULL,
    EXPLAIN_TIME        TIMESTAMP       NOT NULL,
    MIXOPSEQNO             SMALLINT     NOT NULL,
    REEVAL                 CHAR(1)      NOT NULL,
    GROUP_MEMBER       VARCHAR(24)      NOT NULL
   )
    IN database-name.table-space-name

CCSID UNICODE;
```

Figure 170. The CREATE TABLE statement for userid.DSN_FILTER_TABLE

Column descriptions

The following table describes the columns of DSN_FILTER_TABLE.

Table 316. DSN_FILTER_TABLE description

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL	The query number, a number used to help identify the query being explained. It is not a unique identifier. Using negative number will cause problems. The possible sources are: **1** The statement line number in the program **2** QUERYNO clause **3** The EXPLAIN statement **4** EDM unique token in the statement cache.
QBLOCKNO	SMALLINT NOT NULL	The query block number, a number used to identify each query block within a query.
PLANNO	SMALLINT	The plan number, a number used to identify each miniplan with a query block.
APPLNAME	VARCHAR(24) NOT NULL	The application plan name.
PROGNAME	VARCHAR(128) NOT NULL WITH DEFAULT	The program name (binding an application) or the package name (binding a package).
COLLID	INTEGER NOT NULL	The collection ID for the package.
ORDERNO	INTEGER NOT NULL	The sequence number of evaluation. Indicates the order in which the predicate is applied within each stage
PREDNO	INTEGER NOT NULL	The predicate number, a number used to identify a predicate within a query.
STAGE	CHAR(9) NOT NULL	Indicates at which stage the Predicate is evaluated. The possible values are: • 'Matching', • 'Screening', • 'Stage 1' • 'Stage 2'
ORDERCLASS	INTEGER NOT NULL	IBM internal use only.
EXPLAIN_TIME	TIMESTAMP NOT NULL	The explain timestamp.
MIXOPSEQ	SMALLINT NOT NULL	IBM internal use only.
REEVAL	CHAR(1) NOT NULL	IBM internal use only.
GROUP_MEMBER	VARCHAR(24) NOT NULL	The member name of the DB2 subsystem that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.

DSN_DETCOST_TABLE

The detailed cost table, DSN_DETCOST_TABLE, contains information about detailed cost estimation of the mini-plans in a query.

Recommendation: Do not manually insert data into or delete data from EXPLAIN tables. The data is intended to be manipulated only by the DB2 EXPLAIN function and various optimization tools.

CREATE TABLE statement

The following figure shows the format of DSN_DETCOST_TABLE.

```
CREATE TABLE  userid.DSN_DETCOST_TABLE
  (
     QUERYNO             INTEGER      NOT NULL,
     QBLOCKNO            SMALLINT     NOT NULL,
     APPLNAME            VARCHAR(24)  NOT NULL,
     PROGNAME            VARCHAR(128) NOT NULL,
     PLANNO              SMALLINT     NOT NULL,
     OPENIO              FLOAT(4)     NOT NULL,
     OPENCPU             FLOAT(4)     NOT NULL,
     OPENCOST            FLOAT(4)     NOT NULL,
     DMIO                FLOAT(4)     NOT NULL,
     DMCPU               FLOAT(4)     NOT NULL,
     DMTOT               FLOAT(4)     NOT NULL,
     SUBQIO              FLOAT(4)     NOT NULL,
     SUBQCPU             FLOAT(4)     NOT NULL,
     SUBQCOST            FLOAT(4)     NOT NULL,
     BASEIO              FLOAT(4)     NOT NULL,
     BASECPU             FLOAT(4)     NOT NULL,
     BASETOT             FLOAT(4)     NOT NULL,
     ONECOMPROWS         FLOAT(4)     NOT NULL,
     IMLEAF              FLOAT(4)     NOT NULL,
     IMIO                FLOAT(4)     NOT NULL,
     IMPREFH             CHAR(2)      NOT NULL,
     IMMPRED             INTEGER      NOT NULL,
     IMFF                FLOAT(4)     NOT NULL,
     IMSRPRED            INTEGER      NOT NULL,
     IMFFADJ             FLOAT(4)     NOT NULL,
     IMSCANCST           FLOAT(4)     NOT NULL,
     IMROWCST            FLOAT(4)     NOT NULL,
     IMPAGECST           FLOAT(4)     NOT NULL,
     IMRIDSORT           FLOAT(4)     NOT NULL,
     IMMERGCST           FLOAT(4)     NOT NULL,
     IMCPU               FLOAT(4)     NOT NULL,
     IMTOT               FLOAT(4)     NOT NULL,
     IMSEQNO             SMALLINT     NOT NULL,
     DMPREFH             CHAR(2)      NOT NULL,
     DMCLUDIO            FLOAT(4)     NOT NULL,
     DMNCLUDIO           FLOAT(4)     NOT NULL,
     DMPREDS             INTEGER      NOT NULL,
     DMSROWS             FLOAT(4)     NOT NULL,
     DMSCANCST           FLOAT(4)     NOT NULL,
```

Figure 171. The CREATE TABLE statement for userid.DSN_DSN_DETCOST_TABLE_TABLE (part 1 of 3)

```
     DMCOLS              SMALLINT     NOT NULL,
     DMROWS              FLOAT(4)     NOT NULL,
     RDSROWCST           FLOAT(4)     NOT NULL,
     DMPAGECST           FLOAT(4)     NOT NULL,
     DMDATAIO            FLOAT(4)     NOT NULL,
     DMDATACPU           FLOAT(4)     NOT NULL,
     DMDATATOT           FLOAT(4)     NOT NULL,
     RDSROW              FLOAT(4)     NOT NULL,
     SNCOLS              SMALLINT     NOT NULL,
     SNROWS              FLOAT(4)     NOT NULL,
     SNRECSZ             INTEGER      NOT NULL,
     SNPAGES             FLOAT(4)     NOT NULL,
```

```
SNRUNS              FLOAT(4)     NOT NULL,
SNMERGES            FLOAT(4)     NOT NULL,
SNIOCOST            FLOAT(4)     NOT NULL,
SNCPUCOST           FLOAT(4)     NOT NULL,
SNCOST              FLOAT(4)     NOT NULL,
SNSCANIO            FLOAT(4)     NOT NULL,
SNSCANCPU           FLOAT(4)     NOT NULL,
SNSCANCOST          FLOAT(4)     NOT NULL,
SCCOLS              SMALLINT     NOT NULL,
SCROWS              FLOAT(4)     NOT NULL,
SCRECSZ             INTEGER      NOT NULL,
SCPAGES             FLOAT(4)     NOT NULL,
SCRUNS              FLOAT(4)     NOT NULL,
SCMERGES            FLOAT(4)     NOT NULL,
SCIOCOST            FLOAT(4)     NOT NULL,
SCCPUCOST           FLOAT(4)     NOT NULL,
SCCOST              FLOAT(4)     NOT NULL,
SCSCANIO            FLOAT(4)     NOT NULL,
SCSCANCPU           FLOAT(4)     NOT NULL,
SCSCANCOST          FLOAT(4)     NOT NULL,
COMPCARD            FLOAT(4)     NOT NULL,
COMPIOCOST          FLOAT(4)     NOT NULL,
COMPCPUCOST         FLOAT(4)     NOT NULL,
COMPCOST            FLOAT(4)     NOT NULL,
JOINCOLS            SMALLINT     NOT NULL,
EXPLAIN_TIME        TIMESTAMP    NOT NULL,
COSTBLK             INTEGER      NOT NULL,
COSTSTOR            INTEGER      NOT NULL,
```

Figure 172. The CREATE TABLE statement for userid.DSN_DSN_DETCOST_TABLE_TABLE (part 2 of 3)

```
MPBLK               INTEGER      NOT NULL,
MPSTOR              INTEGER      NOT NULL,
COMPOSITES          INTEGER      NOT NULL,
CLIPPED             INTEGER      NOT NULL,
PARTITION           INTEGER      NOT NULL,
TABREF              VARCHAR(64)  NOT NULL,
MAX_COMPOSITES      INTEGER      NOT NULL,
MAX_STOR            INTEGER      NOT NULL,
MAX_CPU             INTEGER      NOT NULL,
MAX_ELAP            INTEGER      NOT NULL,
TBL_JOINED_THRESH   INTEGER      NOT NULL,
STOR_USED           INTEGER      NOT NULL,
CPU_USED            INTEGER      NOT NULL,
ELAPSED             INTEGER      NOT NULL,
MIN_CARD_KEEP       FLOAT(4)     NOT NULL,
MAX_CARD_KEEP       FLOAT(4)     NOT NULL,
MIN_COST_KEEP       FLOAT(4)     NOT NULL,
MAX_COST_KEEP       FLOAT(4)     NOT NULL,
MIN_VALUE_KEEP      FLOAT(4)     NOT NULL,
MIN_VALUE_CARD_KEEP FLOAT(4)     NOT NULL,
MIN_VALUE_COST_KEEP FLOAT(4)     NOT NULL,
MAX_VALUE_KEEP      FLOAT(4)     NOT NULL,
MAX_VALUE_CARD_KEEP FLOAT(4)     NOT NULL,
MAX_VALUE_COST_KEEP FLOAT(4)     NOT NULL,
MIN_CARD_CLIP       FLOAT(4)     NOT NULL,
MAX_CARD_CLIP       FLOAT(4)     NOT NULL,
MIN_COST_CLIP       FLOAT(4)     NOT NULL,
MAX_COST_CLIP       FLOAT(4)     NOT NULL,
MIN_VALUE_CLIP      FLOAT(4)     NOT NULL,
MIN_VALUE_CARD_CLIP FLOAT(4)     NOT NULL,
MIN_VALUE_COST_CLIP FLOAT(4)     NOT NULL,
MAX_VALUE_CLIP      FLOAT(4)     NOT NULL,
MAX_VALUE_CARD_CLIP FLOAT(4)     NOT NULL,
MAX_VALUE_COST_CLIP FLOAT(4)     NOT NULL,
```

```
    GROUP_MEMBER        VARCHAR(24)  NOT NULL,
    PSEQIOCOST          FLOAT(4)     NOT NULL,
    PSEQCPUCOST         FLOAT(4)     NOT NULL,
    PSEQCOST            FLOAT(4)     NOT NULL,
    PADJIOCOST          FLOAT(4)     NOT NULL,
    PADJCPUCOST         FLOAT(4)     NOT NULL,
    PADJCOST            FLOAT(4)     NOT NULL)
   IN database-name.table-space-name
CCSID UNICODE;
```

Figure 173. The CREATE TABLE statement for userid.DSN_DSN_DETCOST_TABLE_TABLE (part 3 of 3)

Column descriptions

The following table describes the columns of DSN_DETCOST_TABLE.

Table 317. DSN_DETCOST_TABLE description

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL	The query number, a number used to help identify the query being explained. It is not a unique identifier. Using negative number will cause problems. The possible sources are: **1** The statement line number in the program **2** QUERYNO clause **3** The EXPLAIN statement **4** EDM unique token in the statement cache.
QBLOCKNO	SMALLINT NOT NULL	The query block number, a number used to identify each query block within a query.
APPLNAME	VARCHAR(24) NOT NULL	The application plan name.
PROGNAME	VARCHAR(128) NOT NULL	The program name (binding an application) or the package name (binding a package).
PLANNO	SMALLINT NOT NULL	The plan number, a number used to identify each mini-plan with a query block.
OPENIO	FLOAT(4) NOT NULL	The Do-at-open IO cost for non-correlated subquery.
OPENCPU	FLOAT(4) NOT NULL	The Do-at-open CPU cost for non-correlated subquery.
OPENCOST	FLOAT(4) NOT NULL	The Do-at-open total cost for non-correlated subquery.
DMIO	FLOAT(4) NOT NULL	IBM internal use only.
DMCPU	FLOAT(4) NOT NULL	IBM internal use only.
DMTOT	FLOAT(4) NOT NULL	IBM internal use only.
SUBQIO	FLOAT(4) NOT NULL	IBM internal use only.
SUBQCOST	FLOAT(4) NOT NULL	IBM internal use only.
BASEIO	FLOAT(4) NOT NULL	IBM internal use only.

Table 317. DSN_DETCOST_TABLE description (continued)

Column name	Data type	Description
BASECPU	FLOAT(4) NOT NULL	IBM internal use only.
BASETOT	FLOAT(4) NOT NULL	IBM internal use only.
ONECOMPROWS	FLOAT(4) NOT NULL	The number of rows qualified after applying local predicates.
IMLEAF	FLOAT(4) NOT NULL	The number of index leaf pages scanned by Data Manager.
IMIO	FLOAT(4) NOT NULL	IBM internal use only.
IMPREFH	CHAR(2) NOT NULL	IBM internal use only.
IMMPRED	INTEGER NOT NULL	IBM internal use only.
IMFF	FLOAT(4) NOT NULL	The filter factor of matching predicates only.
IMSRPRED	INTEGER NOT NULL	IBM internal use only.
IMFFADJ	FLOAT(4) NOT NULL	The filter factor of matching and screening predicates.
IMSCANCST	FLOAT(4) NOT NULL	IBM internal use only.
IMROWCST	FLOAT(4) NOT NULL	IBM internal use only.
IMPAGECST	FLOAT(4) NOT NULL	IBM internal use only.
IMRIDSORT	FLOAT(4) NOT NULL	IBM internal use only.
IMMERGCST	FLOAT(4) NOT NULL	IBM internal use only.
IMCPU	FLOAT(4) NOT NULL	IBM internal use only.
IMTOT	FLOAT(4) NOT NULL	IBM internal use only.
IMSEQNO	SMALLINT NOT NULL	IBM internal use only.
DMPEREFH	FLOAT(4) NOT NULL	IBM internal use only.
DMCLUDIO	FLOAT(4) NOT NULL	IBM internal use only.
DMPREDS	INTEGER NOT NULL	IBM internal use only.
DMSROWS	FLOAT(4) NOT NULL	IBM internal use only.
DMSCANCST	FLOAT(4) NOT NULL	IBM internal use only.
DMCOLS	FLOAT(4) NOT NULL	The number of data manager columns.
DMROWS	FLOAT(4) NOT NULL	The number of data manager rows returned (after all stage 1 predicates are applied).
RDSROWCST	FLOAT(4) NOT NULL	IBM internal use only.
DMPAGECST	FLOAT(4) NOT NULL	IBM internal use only.
DMDATAIO	FLOAT(4) NOT NULL	IBM internal use only.
DMDATAIO	FLOAT(4) NOT NULL	IBM internal use only.
DMDATACPU	FLOAT(4) NOT NULL	IBM internal use only.
DMDATACPU	FLOAT(4) NOT NULL	IBM internal use only.
RDSROW	FLOAT(4) NOT NULL	The number of RDS rows returned (after all stage 1 and stage 2 predicates are applied).
SNCOLS	SMALLINT NOT NULL	The number of columns as sort input for new table.

Table 317. DSN_DETCOST_TABLE description (continued)

Column name	Data type	Description
SNROWS	FLOAT(4) NOT NULL	The number of rows as sort input for new table.
SNRECSZ	INTEGER NOT NULL	The record size for new table.
SNPAGES	FLOAT(4) NOT NULL	The page size for new table.
SNRUNS	FLOAT(4) NOT NULL	The number of runs generated for sort of new table.
SNMERGES	FLOAT(4) NOT NULL	The number of merges needed during sort.
SNIOCOST	FLOAT(4) NOT NULL	IBM internal use only.
SNCPUCOST	FLOAT(4) NOT NULL	IBM internal use only.
SNCOST	FLOAT(4) NOT NULL	IBM internal use only.
SNCSCANIO	FLOAT(4) NOT NULL	IBM internal use only.
SNSCANCPU	FLOAT(4) NOT NULL	IBM internal use only.
SNCCOLS	FLOAT(4) NOT NULL	The number of columns as sort input for Composite table.
SCROWS	FLOAT(4) NOT NULL	The number of rows as sort input for Composite Table.
SCRECSZ	FLOAT(4) NOT NULL	The record size for Composite table.
SCPAGES	FLOAT(4) NOT NULL	The page size for Composite table.
SCRUNS	FLOAT(4) NOT NULL	The number of runs generated during sort of composite.
SCMERGES	FLOAT(4) NOT NULL	The number of merges needed during sort of composite.
SCIOCOST	FLOAT(4) NOT NULL	IBM internal use only.
SCCPUCOST	FLOAT(4) NOT NULL	IBM internal use only.
SCCOST	FLOAT(4) NOT NULL	IBM internal use only.
SCSCANIO	FLOAT(4) NOT NULL	IBM internal use only.
SCSCANCPU	FLOAT(4) NOT NULL	IBM internal use only.
SCSCANCOST	FLOAT(4) NOT NULL	IBM internal use only.
COMPCARD	FLOAT(4) NOT NULL	The total composite cardinality.
COMPIOCOST	FLOAT(4) NOT NULL	IBM internal use only.
COMPCPUCOST	FLOAT(4) NOT NULL	IBM internal use only.
COMPCOST	FLOAT(4) NOT NULL	The total cost.
JOINCOLS	SMALLINT NOT NULL	IBM internal use only.
EXPLAIN_TIME	TIMESTAMP NOT NULL	The EXPLAIN time stamp.
COSTBLK	INTEGER NOT NULL	IBM internal use only.
COSTSTOR	INTEGER NOT NULL	IBM internal use only.
MPBLK	INTEGER NOT NULL	IBM internal use only.
MPSTOR	INTEGER NOT NULL	IBM internal use only.
COMPOSITES	INTEGER NOT NULL	IBM internal use only.
CLIPPED	INTEGER NOT NULL	IBM internal use only.
TABREF	VARCHAR(64) NOT NULL	IBM internal use only.

Table 317. DSN_DETCOST_TABLE description (continued)

Column name	Data type	Description
MAX_COMPOSITES	INTEGER NOT NULL	IBM internal use only.
MAX_STOR	INTEGER NOT NULL	IBM internal use only.
MAX_CPU	INTEGER NOT NULL	IBM internal use only.
MAX_ELAP	INTEGER NOT NULL	IBM internal use only.
TBL_JOINED_THRESH	INTEGER NOT NULL	IBM internal use only.
STOR_USED	INTEGER NOT NULL	IBM internal use only.
CPU_USED	INTEGER NOT NULL	IBM internal use only.
ELAPSED	INTEGER NOT NULL	IBM internal use only.
MIN_CARD_KEEP	FLOAT(4) NOT NULL	IBM internal use only.
MAX_CARD_KEEP	FLOAT(4) NOT NULL	IBM internal use only.
MIN_COST_KEEP	FLOAT(4) NOT NULL	IBM internal use only.
MAX_COST_KEEP	FLOAT(4) NOT NULL	IBM internal use only.
MIN_VALUE_KEEP	FLOAT(4) NOT NULL	IBM internal use only.
MIN_VALUE_CARD_KEEP	FLOAT(4) NOT NULL	IBM internal use only.
MIN_VALUE_COST_KEEP	FLOAT(4) NOT NULL	IBM internal use only.
MIN_CARD_CLIP	FLOAT(4) NOT NULL	IBM internal use only.
MAX_CARD_CLIP	FLOAT(4) NOT NULL	IBM internal use only.
MIN_COST_CLIP	FLOAT(4) NOT NULL	IBM internal use only.
MAX_COST_CLIP	FLOAT(4) NOT NULL	IBM internal use only.
MIN_VALUE_CLIP	FLOAT(4) NOT NULL	IBM internal use only.
MIN_VALUE_CARD_CLIP	FLOAT(4) NOT NULL	IBM internal use only.
MIN_VALUE_COST_CLIP	FLOAT(4) NOT NULL	IBM internal use only.
MAX_VALUE_CLIP	FLOAT(4) NOT NULL	IBM internal use only.
MAX_VALUE_CARD_CLIP	FLOAT(4) NOT NULL	IBM internal use only.
MAX_VALUE_COST_CLIP	FLOAT(4) NOT NULL	IBM internal use only.
GROUP_MEMBER	VARCHAR(240	The member name of the DB2 subsystem that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.
PSEQIOCOST	FLOAT(4) NOT NULL	IBM internal use only.
PSEQIOCOST	FLOAT(4) NOT NULL	IBM internal use only.
PSEQCPUCOST	FLOAT(4) NOT NULL	IBM internal use only.
PSEQCOST	FLOAT(4) NOT NULL	IBM internal use only.
PADJIOCOST	FLOAT(4) NOT NULL	IBM internal use only.
PADJCPUCOST	FLOAT(4) NOT NULL	IBM internal use only.
PADJCOST	FLOAT(4) NOT NULL	IBM internal use only.

DSN_SORT_TABLE

The sort table, DSN_SORT_TABLE, contains information about the sort operations required by a query.

Recommendation: Do not manually insert data into or delete data from EXPLAIN tables. The data is intended to be manipulated only by the DB2 EXPLAIN function and various optimization tools.

CREATE TABLE statement

The following figure shows the format of DSN_SORT_TABLE.

Figure 174. The CREATE TABLE statement for userid.DSN_SORT_TABLE

```
CREATE TABLE userid.DSN_SORT_TABLE
   (
    QUERYNO             INTEGER      NOT NULL,
    QBLOCKNO            SMALLINT     NOT NULL,
    PLANNO              SMALLINT     NOT NULL,
    APPLNAME            VARCHAR(24)  NOT NULL,
    PROGNAME            VARCHAR(128) NOT NULL,
    COLLID              VARCHAR(128) NOT NULL WITH DEFAULT,
    SORTC               CHAR(5)      NOT NULL WITH DEFAULT,
    SORTN               CHAR(5)      NOT NULL WITH DEFAULT,
    SORTNO              SMALLINT     NOT NULL,
    KEYSIZE             SMALLINT     NOT NULL,
    ORDERCLASS          INTEGER      NOT NULL,
    EXPLAIN_TIME        TIMESTAMP    NOT NULL,
    GROUP_MEMBER        VARCHAR(24)  NOT NULL
   )
    IN database-name.table-space-name
-name
CCSID UNICODE;
```

Figure 175. The CREATE TABLE statement for userid.DSN_SORT_TABLE

Column descriptions

The following table describes the columns of DSN_SORT_TABLE.

Table 318. DSN_SORT_TABLE description

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL	The query number, a number used to help identify the query being explained. It is not a unique identifier. Using negative number will cause problems. The possible sources are: **1** The statement line number in the program **2** QUERYNO clause **3** The EXPLAIN statement **4** EDM unique token in the statement cache.
QBLOCKNO	SMALLINT NOT NULL	The query block number, a number used to identify each query block within a query.
PLANNO	SMALLINT NOT NULL	The plan number, a number used to identify each miniplan with a query block.

Table 318. DSN_SORT_TABLE description (continued)

Column name	Data type	Description
APPLNAME	VARCHAR(24) NOT NULL	The application name.
PROGNAME	VARCHAR(128) NOT NULL	The program name (binding an application) or the package name (binding a package).
COLLID	VARCHAR(128) NOT NULL WITH DEFAULT	The collection ID for the package.
SORTC	CHAR(5) NOT NULL WITH DEFAULT	Indicates the reasons for sort of the Composite table. Using a bitmap of 'G','J'.'O','U'. **'G'** Group By **'O'** Order By **'J'** Join **'U'** Uniqueness
SORTN	CHAR(5) NOT NULL WITH DEFAULT	Indicates the reasons for sort of the Composite table. Using a bitmap of 'G','J'.'O','U'. **'G'** Group By **'O'** Order By **'J'** Join **'U'** Uniqueness
SORTNO	SMALLINT NOT NULL	The sequence number of the sort.
KEYSIZE	SMALLINT NOT NULL	The sum of the lengths of the sort keys.
ORDERCLASS	INTEGER NOT NULL	IBM internal use only.
EXPLAIN_TIME	TIMESTAMP NOT NULL	The EXPLAIN timestamp.
GROUP_MEMBER	VARCHAR(24) NOT NULL	The member name of the DB2 subsystem that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.

DSN_SORTKEY_TABLE

The sort key table, DSN_SORTKEY_TABLE, contains information about sort keys for all of the sorts required by a query.

Recommendation: Do not manually insert data into or delete data from EXPLAIN tables. The data is intended to be manipulated only by the DB2 EXPLAIN function and various optimization tools.

CREATE TABLE statement

The following figure shows the format of DSN_SORTKEY_TABLE.

```
CREATE TABLE  userid.DSN_SORTKEY_TABLE
   (
    QUERYNO             INTEGER      NOT NULL,
    QBLOCKNO            SMALLINT     NOT NULL,
```

```
    PLANNO              SMALLINT     NOT NULL,
    APPLNAME            VARCHAR(24)  NOT NULL,
    PROGNAME            VARCHAR(128) NOT NULL,
    COLLID              VARCHAR(128) NOT NULL WITH DEFAULT,
    SORTNO              SMALLINT     NOT NULL,
    ORDERNO             SMALLINT     NOT NULL,
    EXPTYPE             CHAR(3)      NOT NULL,
    TEXT                VARCHAR(128) NOT NULL,
    TABNO               SMALLINT     NOT NULL,
    COLNO               SMALLINT     NOT NULL,
    DATATYPE            CHAR(18)     NOT NULL,
    LENGTH              INTEGER      NOT NULL,
    CCSID               INTEGER      NOT NULL,
    ORDERCLASS          INTEGER      NOT NULL,
    EXPLAIN_TIME        TIMESTAMP    NOT NULL,
    GROUP_MEMBER       VARCHAR(24)   NOT NULL
   )
    IN database-name.table-space-name

CCSID UNICODE;
```

Figure 176. The CREATE TABLE statement for userid.DSN_SORTKEY_TABLE

Column descriptions

The following table describes the columns of DSN_SORTKEY_TABLE.

Table 319. DSN_SORTKEY_TABLE description

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL	The query number, a number used to help identify the query being explained. It is not a unique identifier. Using negative number will cause problems. The possible sources are: **1** The statement line number in the program **2** QUERYNO clause **3** The EXPLAIN statement **4** EDM unique token in the statement cache.
QBLOCKNO	SMALLINT NOT NULL	The query block number, a number used to identify each query block within a query.
PLANNO	SMALLINT NOT NULL	The plan number, a number used to identify each miniplan with a query block.
APPLNAME	VARCHAR(24) NOT NULL	The application name.
PROGNAME	VARCHAR(128) NOT NULL	The program name (binding an application) or the package name (binding a package).
COLLID	VARCHAR(128) NOT NULL WITH DEFAULT	The collection ID for the package.
SORTNO	SMALLINT NOT NULL	The sequence number of the sort
ORDERNO	SMALLINT NOT NULL	The sequence number of the sort key

Table 319. DSN_SORTKEY_TABLE description (continued)

Column name	Data type	Description
EXPTYPE	CHAR(3) NOT NULL	The type of the sort key. The possible values are: • 'COL' • 'EXP' • 'QRY'
TEXT	VARCHAR(128) NOT NULL	The sort key text, can be a column name, an expression, or a scalar subquery, or 'Record ID'.
TABNO	SMALLINT NOT NULL	The table number, a number which uniquely identifies the corresponding table reference within a query.
COLNO	SMALLINT NOT NULL	The column number, a number which uniquely identifies the corresponding column within a query. Only applicable when the sort key is a column.
DATATYPE	CHAR(18)	The data type of sort key. The possible values are • 'HEXADECIMAL' • 'CHARACTER' • 'PACKED FIELD ' • 'FIXED(31)' • 'FIXED(15)' • 'DATE' • 'TIME' • 'VARCHAR' • 'PACKED FLD' • 'FLOAT' • 'TIMESTAMP' • 'UNKNOWN DATA TYPE'
LENGTH	INTEGER NOT NULL	The length of sort key.
CCSID	INTEGER NOT NULL	IBM internal use only.
ORDERCLASS	INTEGER NOT NULL	IBM internal use only.
EXPLAIN_TIME	TIMESTAMP NOT NULL	The EXPLAIN timestamp.
GROUP_MEMBER	VARCHAR(24) NOT NULL	The member name of the DB2 subsystem that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.

DSN_PGRANGE_TABLE

The page range table, DSN_PGRANGE_TABLE, contains information about qualified partitions for all page range scans in a query.

Recommendation: Do not manually insert data into or delete data from EXPLAIN tables. The data is intended to be manipulated only by the DB2 EXPLAIN function and various optimization tools.

CREATE TABLE statement

The following figure shows the format of DSN_PGRANGE_TABLE.

```
CREATE TABLE userid.DSN_PGRANGE_TABLE
        ( QUERYNO            INTEGER      NOT NULL,
          QBLOCKNO           SMALLINT     NOT NULL,
          TABNO              SMALLINT     NOT NULL,
          RANGE              SMALLINT     NOT NULL,
          FIRSTPART          SMALLINT     NOT NULL,
          LASTPART           SMALLINT     NOT NULL,
          NUMPARTS           SMALLINT     NOT NULL,
          EXPLAIN_TIME       TIMESTAMP    NOT NULL,
          GROUP_MEMBER       VARCHAR(24)  NOT NULL
   )
     IN database-name.table-space-name

CCSID UNICODE;
```

Figure 177. The CREATE TABLE statement for userid.DSN_PGRANGE_TABLE

Column descriptions

The following table describes the columns of DSN_PGRANGE_TABLE.

Table 320. DSN_PGRANGE_TABLE description

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL	The query number, a number used to help identify the query being explained. It is not a unique identifier. Using negative number will cause problems. The possible sources are: **1** The statement line number in the program **2** QUERYNO clause **3** The EXPLAIN statement **4** EDM unique token in the statement cache.
QBLOCKNO	SMALLINT NOT NULL	The query block number, a number used to identify each query block within a query.
TABNO	SMALLINT NOT NULL	The table number, a number which uniquely identifies the corresponding table reference within a query.
RANGE	SMALLINT NOT NULL	The sequence number of the current page range.
FIRSTPART	SMALLINT NOT NULL	The starting partition in the current page range.
LASTPART	SMALLINT NOT NULL	The ending partition in the current page range.
NUMPARTS	SMALLINT NOT NULL	The number of partitions in the current page range.
EXPLAIN_TIME	SMALLINT NOT NULL	The EXPLAIN timestamp.
GROUP_MEMBER	VARCHAR(24) NOT NULL	The member name of the DB2 subsystem that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.

DSN_VIEWREF_TABLE

The view reference table, DSN_VIEWREF_TABLE, contains information about all of the views and materialized query tables that are used to process a query.

Recommendation: Do not manually insert data into or delete data from EXPLAIN tables. The data is intended to be manipulated only by the DB2 EXPLAIN function and various optimization tools.

CREATE TABLE statement

The following figure shows the format of DSN_VIEWREF_TABLE.

```
CREATE TABLE userid.DSN_VIEWREF_TABLE    (
  QUERYNO        INTEGER      NOT NULL WITH DEFAULT,
  APPLNAME       VARCHAR(24)  NOT NULL WITH DEFAULT,
  PROGNAME       VARCHAR(128) NOT NULL WITH DEFAULT,
  VERSION        VARCHAR(122) NOT NULL WITH DEFAULT,
  COLLID         VARCHAR(128) NOT NULL WITH DEFAULT,
  CREATOR        VARCHAR(128) NOT NULL WITH DEFAULT,
  NAME           VARCHAR(128) NOT NULL WITH DEFAULT,
  TYPE           CHAR(1)      NOT NULL WITH DEFAULT,
  MQTUSE         SMALLINT     NOT NULL WITH DEFAULT,
  EXPLAIN_TIME   TIMESTAMP    NOT NULL WITH DEFAULT,
  GROUP_MEMBER   VARCHAR(24)  NOT NULL
)
    IN database-name.table-space-name

CCSID UNICODE;
```

Figure 178. The CREATE TABLE statement for userid.DSN_VIEWREF_TABLE

Column descriptions

The following table describes the columns of DSN_VIEWREF_TABLE.

Table 321. DSN_VIEWREF_TABLE description

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL	The query number, a number used to help identify the query being explained. It is not a unique identifier. Using negative number will cause problems. The possible sources are: **1** The statement line number in the program **2** QUERYNO clause **3** The EXPLAIN statement **4** EDM unique token in the statement cache.
APPLNAME	VARCHAR(24) NOT NULL	The application plan name.
PROGNAME	VARCHAR(128) NOT NULL	The program name (binding an application) or the package name (binding a package).
VERSION	VARCHAR(128)	The version identifier for the package. Applies only to an embedded EXPLAIN statement executed from a package or to a statement that is explained when binding a package. Blank if not applicable.
COLLID	VARCHAR(128)	The collection ID for the package. Applies only to an embedded EXPLAIN statement that is executed from a package or to a statement that is explained when binding a package. Blank if not applicable. The value DSNDYNAMICSQLCACHE indicates that the row is for a cached statement.
CREATOR	VARCHAR(128)	Authorization ID of the owner of the object.
NAME	VARCHAR(128)	Name of the object.

Table 321. DSN_VIEWREF_TABLE description (continued)

Column name	Data type	Description
TYPE	CHAR(1) NOT NULL WITH DEFAULT	The type of the object: **'V'** View **'R'** MQT that has been used to replace the base table for rewrite **'M'** MQT
MQTUSE	SMALLINT	IBM internal use only.
EXPLAIN TIME	TIMESTAMP	The EXPLAIN timestamp.
GROUP_MEMBER	VARCHAR(24) NOT NULL	The member name of the DB2 subsystem that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.

DSN_QUERY_TABLE

The query table, DSN_QUERY_TABLE, contains information about a SQL statement, and displays the statement before and after query transformation in XML.

Recommendation: Do not manually insert data into or delete data from EXPLAIN tables. The data is intended to be manipulated only by the DB2 EXPLAIN function and various optimization tools.

CREATE TABLE statement

The following figure shows the format of DSN_QUERY_TABLE.

```
CREATE TABLE userid.DSN_QUERY_TABLE(
   QUERYNO        INTEGER      NOT NULL,
   TYPE           CHAR(8)      NOT NULL,
   QUERY_STAGE    CHAR(8)      NOT NULL,
   SEQNO          INTEGER      NOT NULL,
   NODE_DATA      CLOB(2M)     NOT NULL,
   EXPLAIN_TIME   TIMESTAMP    NOT NULL,
   QUERY_ROWID    ROWID        NOT NULL GENERATED ALWAYS,
   GROUP_MEMBER   VARCHAR(24)  NOT NULL,
   HASHKEY        INTEGER      NOT NULL,
   HAS_PRED       CHAR(1)      NOT NULL
  ) IN database-name.table-space-name

CCSID UNICODE;
```

Figure 179. The CREATE TABLE statement for userid.DSN_QUERY_TABLE

Column descriptions

The following table describes the columns of DSN_QUERY_TABLE.

Table 322. DSN_QUERY_TABLE description

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL	The query number, a number used to help identify the query being explained. It is not a unique identifier. Using negative number will cause problems. The possible sources are: **1** The statement line number in the program **2** QUERYNO clause **3** The EXPLAIN statement **4** EDM unique token in the statement cache.
TYPE	CHAR(8) NOT NULL	The type of the data in the NODE_DATA column.
QUERY STAGE	CHAR(8) NOT NULL WITH DEFAULT	The stage during query transformation when this row is populated.
SEQNO	NOT NULL	The sequence number for this row if NODE_DATA exceeds the size of its column.
NODE_DATA	CLOB(2M)	The XML data containing the SQL statement and its query block, table, and column information.
EXPLAIN_TIME	TIMESTAMP	The EXPLAIN timestamp.
QUERY_ROWID	ROWID NOT NULL GENERATED ALWAYS	The ROWID of the statement.
GROUP MEMBER	VARCHAR(24) NOT NULL	The member name of the DB2 subsystem that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.
HASHKEY	INTEGER NOT NULL	The hash value of the contents in NODE_DATA
HAS_PRED	CHAR(1) NOT NULL	When NODE_DATA contains an SQL statement, this column indicates if the statement contains a parameter marker literal, non-parameter marker literal, or no predicates.

Appendix H. Profile tables that are used by optimization tools

Optimization tools, such as IBM Optimization Service Center for DB2 for z/OS and IBM DB2 Optimization Expert for z/OS can create and use profile tables to enable monitoring of query performance.

.

SYSIBM.DSN_VIRTUAL_INDEXES

The virtual indexes table, DSN_VIRTUAL_INDEXES, enables optimization tools to test the effect of creating and dropping indexes on the performance of particular queries.

PSPI The following table describes the columns in DSN_VIRTUAL_INDEXES.

Recommendation: Do not manually insert data into or delete data from this table, it is intended to be used only by optimization tools.

Table 323. SYSIBM.DSN_VIRTUAL_INDEXES description

Column name	Data type	Description
TBCREATOR	VARCHAR(128)	The schema or authorization ID of the owner of the table on which the index is being created or dropped.
TBNAME	VARCHAR(128)	The name of the table on which the index is being created or dropped.
IXCREATOR	VARCHAR(128)	The schema or authorization ID of the owner of the index.
IXNAME	VARCHAR(128)	The index name.
ENABLE	CHAR(1)	Indicates whether this index should be considered in the scenario that is being tested. This column can have one of the following values: **Y** Use this index. **N** Do not use this index. If this column contains 'Y', but the index definition is not valid, the index is ignored.
MODE	CHAR(1)	Indicates whether the index is being created or dropped. This column can have one of the following values: **C** This index is to be created. **D** This index is to be dropped.
UNIQUERULE	CHAR(1)	Indicates whether the index is unique. This column can have one of the following values: **D** The index is not unique. (Duplicates are allowed.) **U** This index is unique.
COLCOUNT	SMALLINT	The number of columns in the key.

© Copyright IBM Corp. 1982, 2008

Table 323. SYSIBM.DSN_VIRTUAL_INDEXES description (continued)

Column name	Data type	Description
CLUSTERING	CHAR(1)	Indicates whether the index is clustered. This column can have one of the following values: **Y** The index is clustered. **N** The index is not clustered.
NLEAF	INTEGER	The number of active leaf pages in the index. If unknown, the value is -1.
NLEVELS	SMALLINT	The number of levels in the index tree. If unknown, the value is -1.
INDEXTYPE	CHAR(1)	The index type. This column can have one of the following values: **2** The index is a nonpartitioned secondary index. **D** The index is a data-partitioned secondary index.
PGSIZE	SMALLINT	The size, in kilobytes, of the leaf pages in the index. This column can have one of the following values: 4, 8, 16, or 32.
FIRSTKEYCARDF	FLOAT	The number of distinct values of the first key column. If unknown, the value is -1.
FULLKEYCARDF	FLOAT	The number of distinct values of the key. If unknown, the value is -1.
CLUSTERRATIOF	FLOAT	The percentage of rows that are in clustering order. Multiply this column value by 100 to get the percent value. For example, a value of .9125 in this column indicates that 91.25%. of the rows are in clustering order. If unknown, the value is -1.
PADDED	CHAR(1)	Indicates whether keys within the index are padded for varying-length column data. This column can have one of the following values: **Y** The keys are padded. **N** The keys are not padded.
COLNO1	SMALLINT	The column number of the first column in the index key.
ORDERING1	CHAR(1)	Indicates the order of the first column in the index key. This column can have one of the following values: **A** Ascending **D** Descending
COLNO*n*	SMALLINT	The column number of the *n*th column in the index key, where *n* is a number between 2 and 64, including 2 and 64. If the number of index keys is less than *n*, this column is null.
ORDERING*n*	CHAR(1)	Indicates the order of the *n*th column in the index key, where *n* is a number between 2 and 64, including 2 and 64. This column can have one of the following values: **A** Ascending **D** Descending If the number of index keys is less than *n*, this column is null.

The SQL statement to create DSN_VIRTUAL_INDEXES is in member DSNTESC of the SDSNSAMP library. PSPI

Appendix I. Real-time statistics tables

The information under this heading is Product-sensitive Programming Interface and Associated Guidance Information, as defined in "Notices" on page 1301.

DB2 collects statistics that you can use to determine when you need to perform certain maintenance functions on your table spaces and index spaces.

DB2 collects the statistics in real time. You create tables into which DB2 periodically writes the statistics. You can then write applications that query the statistics and help you decide when to run REORG, RUNSTATS, or COPY, or to enlarge your data sets. Figure 180 shows an overview of the process of collecting and using real-time statistics.

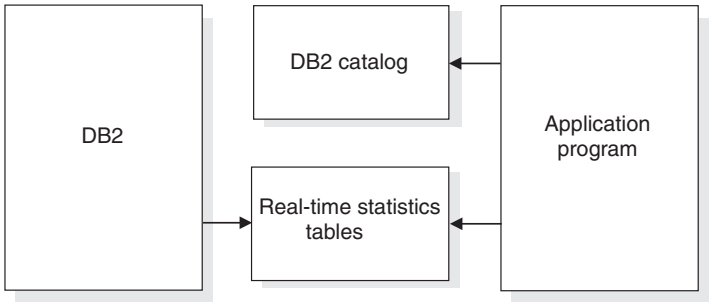


Figure 180. Real-time statistics overview

The following sections provide detailed information about the real-time statistics tables:
- "Setting up your system for real-time statistics"
- "Contents of the real-time statistics tables" on page 1231
- "Operating with real-time statistics" on page 1243

For information about a DB2-supplied stored procedure that queries the real-time statistics tables, see "The DB2 real-time statistics stored procedure" on page 1257.

Setting up your system for real-time statistics

DB2 always generates in-memory statistics for each table space, index space, including catalog objects but not the directory, in your system. For partitioned spaces, DB2 generates information for each partition. However, you need to perform the following steps before DB2 externalizes the statistics to DB2 tables:

1. Create the real-time statistics objects. See "Creating and altering the real-time statistics objects" on page 1230.
2. Set the interval for writing statistics. See "Setting the interval for writing real-time statistics" on page 1231.
3. Start the real-time statistics database. See "Starting the real-time statistics database" on page 1231.
4. Establish base values for the statistics. See "Establishing base values for real-time statistics" on page 1231.

© Copyright IBM Corp. 1982, 2008

Creating and altering the real-time statistics objects

You need to create a database, table space, tables, and indexes for the real-time statistics. Those objects are listed in Table 324. Use the SQL statements in member DSNTESS of data set DSN810.SDSNSAMP as a model for creating the real-time statistics objects. You can create these objects in user-managed or DB2-managed data sets.

Restrictions on changing the provided definitions for the real-time statistics objects: You can change most of the attributes in the provided definitions of the real-time statistics objects. However, you cannot change the following items:

- Object names. You must use the names that are specified in DSNTESS for the database, table space, tables, indexes, and table columns.
- The CCSID parameter on the CREATE DATABASE, CREATE TABLESPACE, and CREATE TABLE statements. The CCSID must be EBCDIC.
- Number of columns or column definitions. You cannot add table columns or modify column definitions.
- The LOCKSIZE parameter on the CREATE TABLESPACE statement. The LOCKSIZE must be ROW.

Before you can alter an object in the real-time statistics database, you must stop the database. Otherwise, you receive an SQL error. Table 324 shows the DB2 objects for storing real-time statistics.

Table 324. DB2 objects for storing real-time statistics

Object name	Description
DSNRTSDB	Database for real-time statistics objects
DSNRTSTS	Table space for real-time statistics objects
SYSIBM.TABLESPACESTATS	Table for statistics on table spaces and table space partitions
SYSIBM.INDEXSPACESTATS	Table for statistics on index spaces and index space partitions
SYSIBM.TABLESPACESTATS_IX	Unique index on SYSIBM.TABLESPACESTATS (columns DBID, PSID, and PARTITION)
SYSIBM.INDEXSPACESTATS_IX	Unique index on SYSIBM.INDEXSPACESTATS (columns DBID, ISOBID, and PARTITION)

To create the real-time statistics objects, you need the authority to create tables and indexes on behalf of the SYSIBM authorization ID.

DB2 inserts one row in the table for each partition or non-partitioned table space or index space. You therefore need to calculate the amount of disk space that you need for the real-time statistics tables based on the current number of table spaces and indexes in your subsystem.

To determine the amount of storage that you need for the real-time statistics when they are in memory, use the following formula:

```
Max_concurrent_objects_updated * 152 bytes = Storage_in_bytes
```

Where Max_concurrent_objects_updated is the peak number of objects that might be updated concurrently and 152 bytes is the amount of in-memory space that DB2 uses for each object.

Recommendation: Place the statistics indexes and tables in their own buffer pool. When the statistics pages are in memory, the speed at which in-memory statistics are written to the tables improves.

Setting the interval for writing real-time statistics

You can set the interval for writing real-time statistics when you install DB2, and you can subsequently update that interval online. The installation field is REAL TIME STATS on panel DSNTIPO. The default interval is 30 minutes. To update the interval, modify system parameter STATSINT.

In a data sharing environment, each member has its own interval for writing real-time statistics.

Starting the real-time statistics database

After you create the real-time statistics database, DB2 puts it into a stopped state. After you create all the objects in the database, you need to issue START DATABASE(DSNRTSDB) to explicitly start the database.

You must start the database in read-write modeso that DB2 can externalize real-time statistics. See "When DB2 externalizes real-time statistics" on page 1243 for information about the conditions for which DB2 externalizes the statistics.

Establishing base values for real-time statistics

Many columns in the real-time statistics tables show the number of times an operation was performed between the last time a particular utility was run and when the real-time statistics are written. For example, STATSINSERT in TABLESPACESTATS indicates the number of records or LOBs that have been inserted after the last RUNSTATS utility was run on the table space or partition. Therefore, for each object for which you want real-time statistics, run the appropriate utility (REORG, RUNSTATS, LOAD REPLACE, REBUILD INDEX, or COPY) to establish a base value from which the delta value can be calculated.

Contents of the real-time statistics tables

The SYSIBM.TABLESPACESTATS table contains statistics information about table spaces and table space partitions. The SYSIBM.INDEXSPACESTATS table contains statistics information about index spaces and index space partitions.

Table 325 describes the columns of the TABLESPACESTATS table and explains how you can use them in deciding when to run REORG, RUNSTATS, or COPY.

Table 325. Descriptions of columns in the TABLESPACESTATS table

Column name	Data type	Description
DBNAME	CHAR(8) NOT NULL	The name of the database. This column is used to map a database to its statistics.
NAME	CHAR(8) NOT NULL	The name of the table space. This column is used to map a table space to its statistics.
PARTITION	SMALLINT NOT NULL	The data set number within the table space. This column is used to map a data set number in a table space to its statistics. For partitioned table spaces, this value corresponds to the partition number for a single partition. For nonpartitioned table spaces, this value is 0.

Table 325. Descriptions of columns in the TABLESPACESTATS table (continued)

Column name	Data type	Description
DBID	SMALLINT NOT NULL	The internal identifier of the database. This column is used to map a DBID to its statistics.
PSID	SMALLINT NOT NULL	The internal identifier of the table space page set descriptor. This column is used to map a PSID to its statistics.
UPDATESTATSTIME	TIMESTAMP NOT NULL WITH DEFAULT	The timestamp when the row was inserted or last updated. This column is updated with the current timestamp when a row in the TABLESPACESTATS table is inserted or updated. You can use this column in several ways: • To determine the actions that caused the latest change to the table. Do this by selecting any of the timestamp columns and comparing them to the UPDATESTATSTIME column. • To determine whether an analysis of data is needed. This determination might be based on a given time interval, or on a combination of the time interval and the amount of activity. For example, suppose that you want to analyze statistics for the last seven days. To determine whether there has been any activity in the past seven days, check whether the difference between the current date and the UPDATESTATSTIME value is less than or equal to seven: `(JULIAN_DAY(CURRENT DATE)-JULIAN_DAY(UPDATESTATSTIME))<= 7`
TOTALROWS	FLOAT	The number of rows or LOBs in the table space or partition. If the table space contains more than one table, this value is the sum of all rows in all tables. A null value means that the number of rows is unknown or that REORG or LOAD has never been run. Use the TOTALROWS value with the value of any column that contains some affected rows to determine the percentage of rows that are affected by a particular action.
NACTIVE	INTEGER	The number of active pages in the table space or partition. A null value means that the number of active pages is unknown. This value is equivalent to the number of preformatted pages. For multi-piece table spaces, this value is the total number of preformatted pages in all data sets. Use the NACTIVE value with the value of any column that contains some affected pages to determine the percentage of pages that are affected by a particular action. For example, suppose that your site's maintenance policies require that COPY is to be run after 20% of the pages in a table space have changed. To determine if a COPY might be required, calculate the ratio of updated pages since the last COPY to the total number of active pages. If the percentage is greater than 20, you need to run COPY: `((COPYUPDATEDPAGES*100)/NACTIVE)>20`

Table 325. Descriptions of columns in the TABLESPACESTATS table (continued)

Column name	Data type	Description
SPACE	INTEGER	The amount of space, in KB, that is allocated to the table space or partition. For multi-piece linear page sets, this value is the amount of space in all data sets. A null value means the amount of space is unknown. Use this value to monitor growth and validate design assumptions.
EXTENTS	SMALLINT	The number of extents in the table space or partition. For multi-piece table spaces, this value is the number of extents for the last data set. For a data set that is striped across multiple volumes, the value is the number of logical extents. A null value means that the number of extents is unknown. Use this value to determine: • When the primary or secondary allocation value for a table space or partition needs to be altered. • When you are approaching the maximum number of extents and risking extend failures.
LOADRLASTTIME	TIMESTAMP	The timestamp of the last LOAD REPLACE on the table space or partition. A null value means that LOAD REPLACE has never been run on the table space or partition or that the timestamp of the last LOAD REPLACE is unknown. You can compare this timestamp to the timestamp of the last COPY on the same object to determine when a COPY is needed. If the date of the last LOAD REPLACE is more recent than the last COPY, you might need to run COPY: `(JULIAN_DAY(LOADRLASTTIME)>JULIAN_DAY(COPYLASTTIME))`
REORGLASTTIME	TIMESTAMP	The timestamp of the last REORG on the table space or partition. A null value means REORG has never been run on the table space or partition or that the timestamp of the last REORG is unknown. You can compare this timestamp to the timestamp of the last COPY on the same object to determine when a COPY is needed. If the date of the last REORG is more recent than the last COPY, you might need to run COPY: `(JULIAN_DAY(REORGLASTTIME)>JULIAN_DAY(COPYLASTTIME))`
REORGINSERTS	INTEGER	The number of records or LOBs that have been inserted since the last REORG or LOAD REPLACE on the table space or partition. A null value means that the number of inserted records or LOBs is unknown.
REORGDELETES	INTEGER	The number of records or LOBs that have been deleted since the last REORG or LOAD REPLACE on the table space or partition. A null value means that the number of deleted records or LOBs is unknown.

Table 325. Descriptions of columns in the TABLESPACESTATS table (continued)

Column name	Data type	Description
REORGUPDATES	INTEGER	The number of rows that have been updated since the last REORG or LOAD REPLACE on the table space or partition. This value does not include LOB updates because LOB updates are really deletions followed by insertions. A null value means that the number of updated rows is unknown. This value can be used with REORGDELETES and REORGINSERTS to determine if a REORG is necessary. For example, suppose that your site's maintenance policies require that REORG is run after 20 per cent of the rows in a table space have changed. To determine if a REORG is required, calculate the sum of updated, inserted, and deleted rows since the last REORG. Then calculate the ratio of that sum to the total number of rows. If the percentage is greater than 20, you might need to run REORG: `(((REORGINSERTS+REORGDELETES+REORGUPDATES)*100)/TOTALROWS)>20`
REORGDISORGLOB	INTEGER	The number of LOBs that were inserted since the last REORG or LOAD REPLACE that are not perfectly chunked. A LOB is perfectly chunked if the allocated pages are in the minimum number of chunks. A null value means that the number of imperfectly chunked LOBs is unknown. Use this value to determine whether you need to run REORG. For example, you might want to run REORG if the ratio of REORGDISORGLOB to the total number of LOBs is greater than 10%: `((REORGDISORGLOB*100)/TOTALROWS)>10`
REORGUNCLUSTINS	INTEGER	The number of records that were inserted since the last REORG or LOAD REPLACE that are not well-clustered with respect to the clustering index. A record is well-clustered if the record is inserted into a page that is within 16 pages of the ideal candidate page. The clustering index determines the ideal candidate page. A null value means that the number of badly-clustered pages is unknown. You can use this value to determine whether you need to run REORG. For example, you might want to run REORG if the following comparison is true: `((REORGUNCLUSTINS*100)/TOTALROWS)>10`
REORGMASSDELETE	INTEGER	The number of mass deletes from a segmented or LOB table space, or the number of dropped tables from a segmented table space, since the last REORG or LOAD REPLACE. A null value means that the number of mass deletes is unknown. If this value is non-zero, a REORG might be necessary.
REORGNEARINDREF	INTEGER	The number of overflow records that were created since the last REORG or LOAD REPLACE and were relocated near the pointer record. For nonsegmented table spaces, a page is near the present page if the two page numbers differ by 16 or less. For segmented table spaces, a page is near the present page if the two page numbers differ by SEGSIZE*2 or less. A null value means that the number of overflow records near the pointer record is unknown.

Table 325. Descriptions of columns in the TABLESPACESTATS table (continued)

Column name	Data type	Description

REORGFARINDEF	INTEGER	The number of overflow records that were created since the last REORG or LOAD REPLACE and were relocated far from the pointer record. For nonsegmented table spaces, a page is far from the present page if the two page numbers differ by more than 16. For segmented table spaces, a page is far from the present page if the two page numbers differ by at least (SEGSIZE*2)+1.
		
		A null value means that the number of overflow records far from the pointer record is unknown.
		
		For example, in a non-data sharing environment, you might run REORG if the following comparison is true:
		`(((REORGNEARINDREF+REORGFARINDREF)*100)/TOTALROWS)>10`
		
		In a data sharing environment, you might run REORG if the following comparison is true:
		`(((REORGNEARINDREF+REORGFARINDREF)*100)/TOTALROWS)>5`

STATSLASTTIME	TIMESTAMP	The timestamp of the last RUNSTATS on the table space or partition. A null value means that RUNSTATS has never been run on the table space or partition, or that the timestamp of the last RUNSTATS is unknown. You can compare this timestamp to the timestamp of the last REORG on the same object to determine when RUNSTATS is needed. If the date of the last REORG is more recent than the last RUNSTATS, you might need to run RUNSTATS: `(JULIAN_DAY(REORGLASTTIME)>JULIAN_DAY(STATSLASTTIME))`
STATSINSERTS	INTEGER	The number of records or LOBs that have been inserted since the last RUNSTATS on the table space or partition. A null value means that the number of inserted records or LOBs is unknown.
STATSDELETES	INTEGER	The number of records or LOBs that have been deleted since the last RUNSTATS on the table space or partition. A null value means that the number of deleted records or LOBs is unknown.
STATSUPDATES	INTEGER	The number of rows that have been updated since the last RUNSTATS on the table space or partition. This value does not include LOB updates because LOB updates are really deletions followed by insertions. A null value means that the number of updated rows is unknown. This value can be used with STATSDELETES and STATSINSERTS to determine if RUNSTATS is necessary. For example, suppose that your site's maintenance policies require that RUNSTATS is to be run after 20% of the rows in a table space have changed. To determine if RUNSTATS is required, calculate the sum of updated, inserted, and deleted rows since the last RUNSTATS. Then calculate the ratio of that sum to the total number of rows. If the percentage is greater than 20, you need to run RUNSTATS: `(((STATSINSERTS+STATSDELETES+STATSUPDATES)*100)/TOTALROWS)>20`

Table 325. Descriptions of columns in the TABLESPACESTATS table (continued)

Column name	Data type	Description
STATSMASSDELETE	INTEGER	The number of mass deletes from a segmented or LOB table space, or the number of dropped tables from a segmented table space, since the last RUNSTATS. A null value means that the number of mass deletes is unknown. If this value is non-zero, RUNSTATS might be necessary.
COPYLASTTIME	TIMESTAMP	The timestamp of the last full or incremental image copy on the table space or partition. A null value means that COPY has never been run on the table space or partition, or that the timestamp of the last full image copy is unknown. You can compare this timestamp to the timestamp of the last REORG on the same object to determine when a COPY is needed. If the date of the last REORG is more recent than the last COPY, you might need to run COPY: `(JULIAN_DAY(REORGRLASTTIME)>JULIAN_DAY(COPYLASTTIME))`

COPYUPDATEDPAGES	INTEGER	The number of distinct pages that have been updated since the last COPY.
		
		A null value means that the number of updated pages is unknown.
		
		You can compare this value to the total number of pages to determine when a COPY is needed.
		
		For example, you might want to take an incremental image copy when 1% of the pages have changed:
		`((COPYUPDATEDPAGES*100)/NACTIVE)>1`
		
		You might want to take a full image copy when 20% of the pages have changed:
		`((COPYUPDATEDPAGES*100)/NACTIVE)>20`

COPYCHANGES	INTEGER	The number of insert, delete, and update operations since the last COPY.
		
		A null value means that the number of insert, delete, or update operations is unknown.
		
		This number indicates the approximate number of log records that DB2 needs to process to recover to the current state.
		
		For example, you might want to take an incremental image copy when DB2 processes more than 1% of the rows from the logs:
		`((COPYCHANGES*100)/TOTALROWS)>1`
		
		You might want to take a full image copy when DB2 processes more than 10% of the rows from the logs:
		`((COPYCHANGES*100)/TOTALROWS)>10`

Table 325. Descriptions of columns in the TABLESPACESTATS table (continued)

Column name	Data type	Description
COPYUPDATELRSN	CHAR(6) FOR BIT DATA	The LRSN or RBA of the first update after the last COPY. A null value means that the LRSN or RBA is unknown. Consider running COPY if this value is not in the active logs. To determine the oldest LRSN or RBA in the active logs, use the print log map utility (DSNJU004).
COPYUPDATETIME	TIMESTAMP	The timestamp of the first update after the last COPY. A null value means that the timestamp is unknown.

Table 326 describes the columns of the INDEXSPACESTATS table and explains how you can use them in deciding when to run REORG, RUNSTATS, or COPY.

Table 326. Descriptions of columns in the INDEXSPACESTATS table

Column name	Data type	Description
DBNAME	CHAR(8) NOT NULL	The name of the database. This column is used to map a database to its statistics.
NAME	CHAR(8) NOT NULL	The name of the index space. This column is used to map an index space to its statistics.
PARTITION	SMALLINT NOT NULL	This column is used to map a data set number in an index space to its statistics. The data set number within the index space. For partitioned index spaces, this value corresponds to the partition number for a single partition. For nonpartitioned index spaces, this value is 0.
DBID	SMALLINT NOT NULL	The internal identifier of the database. This column is used to map a DBID to its statistics.
ISOBID	SMALLINT NOT NULL	The internal identifier of the index space page set descriptor. This column is used to map an ISOBID to its statistics.
PSID	SMALLINT NOT NULL	The internal identifier of the table space page set descriptor for the table space associated with the index that is represented by this row. This column is used to map a PSID to the statistics for the associated index.

Table 326. Descriptions of columns in the INDEXSPACESTATS table (continued)

Column name	Data type	Description
UPDATESTATSTIME	TIMESTAMP NOT NULL WITH DEFAULT	The timestamp when the row was inserted or last updated. This column is updated with the current timestamp when a row in the INDEXSPACESTATS table is inserted or updated. You can use this column in several ways: • To determine the actions that caused the latest change to the INDEXSPACESTATS table. Do this by selecting any of the timestamp columns and comparing them to the UPDATESTATSTIME column. • To determine whether an analysis of data is needed. This determination might be based on a given time interval, or on a combination of the time interval and the amount of activity. For example, suppose that you want to analyze statistics for the last seven days. To determine whether any activity has occurred in the past seven days, check whether the difference between the current date and the UPDATESTATSTIME value is less than or equal to seven: `(JULIAN_DAY(CURRENT DATE)-JULIAN_DAY(UPDATESTATSTIME))<= 7`
TOTALENTRIES	FLOAT	The number of entries, including duplicate entries, in the index space or partition. A null value means that the number of entries is unknown, or that REORG, LOAD, or REBUILD has never been run. Use this value with the value of any column that contains a number of affected index entries to determine the percentage of index entries that are affected by a particular action.
NLEVELS	SMALLINT	The number of levels in the index tree. A null value means that the number of levels is unknown.
NACTIVE	INTEGER	The number of active pages in the index space or partition. This value is equivalent to the number of preformatted pages. A null value means that the number of active pages is unknown. Use this value with the value of any column that contains a number of affected pages to determine the percentage of pages that are affected by a particular action. For example, suppose that your site's maintenance policies require that COPY is to be run after 20% of the pages in an index space have changed. To determine if a COPY is required, calculate the ratio of updated pages since the last COPY to the total number of active pages. If the percentage is greater than 20, you need to run COPY: `((COPYUPDATEDPAGES*100)/NACTIVE)>20`
SPACE	INTEGER	The amount of space, in KB, that is allocated to the index space or partition. For multi-piece linear page sets, this value is the amount of space in all data sets. A null value means that the amount of space is unknown. Use this value to monitor growth and validate design assumptions.

Table 326. Descriptions of columns in the INDEXSPACESTATS table (continued)

Column name	Data type	Description
EXTENTS	SMALLINT	The number of extents in the index space or partition. For multi-piece index spaces, this value is the number of extents for the last data set. For a data set that is striped across multiple volumes, the value is the number of logical extents. A null value means that the number of extents is unknown. Use this value to determine: • When the primary allocation value for an index space or partition needs to be altered. • When you are approaching the maximum number of extents and risking extend failures.
LOADRLASTTIME	TIMESTAMP	The timestamp of the last LOAD REPLACE on the index space or partition. A null value means that the timestamp of the last LOAD REPLACE is unknown. If COPY YES was specified when the index was created (the value of COPY is Y in SYSIBM.SYSINDEXES), you can compare this timestamp to the timestamp of the last COPY on the same object to determine when a COPY is needed. If the date of the last LOAD REPLACE is more recent than the last COPY, you might need to run COPY: `(JULIAN_DAY(LOADRLASTTIME)>JULIAN_DAY(COPYLASTTIME))`
REBUILDLASTTIME	TIMESTAMP	The timestamp of the last REBUILD INDEX on the index space or partition. A null value means that the timestamp of the last REBUILD INDEX is unknown. If COPY YES was specified when the index was created (the value of COPY is Y in SYSIBM.SYSINDEXES), you can compare this timestamp to the timestamp of the last COPY on the same object to determine when a COPY is needed. If the date of the last REBUILD INDEX is more recent than the last COPY, you might need to run COPY: `(JULIAN_DAY(REBUILDLASTTIME)>JULIAN_DAY(COPYLASTTIME))`
REORGLASTTIME	TIMESTAMP	The timestamp of the last REORG INDEX on the index space or partition. A null value means that the timestamp of the last REORG INDEX is unknown. If COPY YES was specified when the index was created (the value of COPY is Y in SYSIBM.SYSINDEXES), you can compare this timestamp to the timestamp of the last COPY on the same object to determine when a COPY is needed. If the date of the last REORG INDEX is more recent than the last COPY, you might need to run COPY: `(JULIAN_DAY(REORGLASTTIME)>JULIAN_DAY(COPYLASTTIME))`

Table 326. Descriptions of columns in the INDEXSPACESTATS table (continued)

Column name	Data type	Description
REORGINSERTS	INTEGER	The number of index entries that have been inserted since the last REORG, REBUILD INDEX, or LOAD REPLACE on the index space or partition. A null value means that the number of inserted index entries is unknown.
REORGDELETES	INTEGER	The number of index entries that have been deleted since the last REORG, REBUILD INDEX, or LOAD REPLACE on the index space or partition. A null value means that the number of deleted index entries is unknown. This value can be used with REORGINSERTS to determine if a REORG is necessary. For example, suppose that your site's maintenance policies require that REORG is to be run after 20% of the index entries have changed. To determine if a REORG is required, calculate the sum of inserted and deleted rows since the last REORG. Then calculate the ratio of that sum to the total number of index entries. If the percentage is greater than 20, you need to run REORG: `(((REORGINSERTS+REORGDELETES)*100)/TOTALENTRIES)>20`
REORGAPPENDINSERT	INTEGER	The number of index entries that have been inserted since the last REORG, REBUILD INDEX, or LOAD REPLACE on the index space or partition that have a key value that is greater than the maximum key value in the index or partition. A null value means that the number of inserted index entries is unknown. This value can be used with REORGINSERTS to decide when to adjust the PCTFREE specification for the index. For example, if the ratio of REORGAPPENDINSERT to REORGINSERTS is greater than 10%, you might need to run ALTER INDEX to adjust PCTFREE or to run REORG more frequently: `((REORGAPPENDINSERT*100)/REORGINSERTS)>10`
REORGPSEUDODELETES	INTEGER	The number of index entries that have been pseudo-deleted since the last REORG, REBUILD INDEX, or LOAD REPLACE on the index space or partition. A pseudo-delete is a RID entry that has been marked as deleted. A null value means that the number of pseudo-deleted index entries is unknown. This value can be used to determine if a REORG is necessary. For example, if the ratio of pseudo-deletes to total index entries is greater than 10%, you might need to run REORG: `((REORGPSEUDODELETES*100)/TOTALENTRIES)>10`
REORGMASSDELETE	INTEGER	The number of times that an index or index space partition was mass deleted since the last REORG, REBUILD INDEX, or LOAD REPLACE. A null value means that the number of mass deletes is unknown. If this value is non-zero, a REORG might be necessary.

Table 326. Descriptions of columns in the INDEXSPACESTATS table (continued)

Column name	Data type	Description
REORGLEAFNEAR	INTEGER	The number of index page splits that occurred since the last REORG, REBUILD INDEX, or LOAD REPLACE in which the higher part of the split page was near the location of the original page. The higher part of a split page is near the original page if the two page numbers differ by 16 or less. A null value means that the number of split pages near their original pages is unknown.
REORGLEAFFAR	INTEGER	The number of index page splits that occurred since the last REORG, REBUILD INDEX, or LOAD REPLACE in which the higher part of the split page was far from the location of the original page. The higher part of a split page is far from the original page if the two page numbers differ by more than 16. A null value means that the number of split pages that are far from their original pages is unknown. This value can be used to decide when to run REORG. For example, calculate the ratio of index page splits in which the higher part of the split page was far from the location of the original page to the number of active pages. If this value is greater than 10% you might need to run REORG: `((REORGLEAFFAR*100)/NACTIVE)>10`
REORGNUMLEVELS	INTEGER	The number of levels in the index tree that were added or removed since the last REORG, REBUILD INDEX, or LOAD REPLACE. A null value means that the number of added or deleted levels is unknown. If this value has increased since the last REORG, REBUILD INDEX, or LOAD REPLACE, you need to check other values such as REORGPSEUDODELETES to determine whether to run REORG. If this value is less than zero, the index space contains empty pages. Running REORG can save disk space and decrease index sequential scan I/O time by eliminating those empty pages.
STATSLASTTIME	TIMESTAMP	The timestamp of the last RUNSTATS on the index space or partition. A null value means that RUNSTATS has never been run on the index space or partition, or that the timestamp of the last RUNSTATS is unknown. You can compare this timestamp to the timestamp of the last REORG on the same object to determine when RUNSTATS is needed. If the date of the last REORG is more recent than the last RUNSTATS, you might need to run RUNSTATS: `(JULIAN_DAY(REORGLASTTIME)>JULIAN_DAY(STATSLASTTIME))`
STATSINSERTS	INTEGER	The number of index entries that have been inserted since the last RUNSTATS on the index space or partition. A null value means that the number of inserted index entries unknown.

Table 326. Descriptions of columns in the INDEXSPACESTATS table (continued)

Column name	Data type	Description
STATSDELETES	INTEGER	The number of index entries that have been deleted since the last RUNSTATS on the index space or partition. A null value means that the number of deleted index entries is unknown. This value can be used with STATSINSERTS to determine if RUNSTATS is necessary. For example, suppose that your site's maintenance policies require that RUNSTATS is run after 20% of the rows in an index space have changed. To determine if RUNSTATS is required, calculate the sum of inserted and deleted index entries since the last RUNSTATS. Then calculate the ratio of that sum to the total number of index entries. If the percentage is greater than 20, you need to run RUNSTATS: `(((STATSINSERTS+STATSDELETES)*100)/TOTALENTRIES)>20`
STATSMASSDELETE	INTEGER	The number of times that the index or index space partition was mass deleted since the last RUNSTATS. A null value means that the number of mass deletes is unknown. If this value is non-zero, RUNSTATS might be necessary.
COPYLASTTIME	TIMESTAMP	The timestamp of the last full image copy on the index space or partition. A null value means that COPY has never been run on the index space or partition, or that the timestamp of the last full image copy is unknown. You can compare this timestamp to the timestamp of the last REORG on the same object to determine when a COPY is needed. If the date of the last REORG is more recent than the last COPY, you might need to run COPY: `(JULIAN_DAY(REORGRLASTTIME)>JULIAN_DAY(COPYLASTTIME))`
COPYUPDATEDPAGES	INTEGER	The number of distinct pages that have been updated since the last COPY. A null value means that the number of updated pages is unknown, or that the index was created with COPY NO. You can compare this value to the total number of pages to determine when a COPY is needed. For example, you might want to take a full image copy when 20% of the pages have changed: `((COPYUPDATEDPAGES*100)/NACTIVE)>20`
COPYCHANGES	INTEGER	The number of insert or delete operations since the last COPY. A null value means that the number of insert or update operations is unknown, or that the index was created with COPY NO. This number indicates the approximate number of log records that DB2 needs to process to recover to the current state. For example, you might want to take a full image copy when DB2 processes more than 10% of the index entries from the logs: `((COPYCHANGES*100)/TOTALENTRIES)>10`

Table 326. Descriptions of columns in the INDEXSPACESTATS table (continued)

Column name	Data type	Description
COPYUPDATELRSN	CHAR(6) FOR BIT DATA	The LRSN or RBA of the first update after the last COPY. A null value means that the LRSN or RBA is unknown, or that the index was created with COPY NO. Consider running COPY if this value is not in the active logs. To determine the oldest LRSN or RBA in the active logs, use the print log map utility (DSNJU004).
COPYUPDATETIME	TIMESTAMP	The timestamp of the first update after the last COPY. A null value means that the timestamp is unknown, or tthat he index was created with COPY NO.

Operating with real-time statistics

To use the real-time statistics effectively, you need to understand when DB2 collects and externalizes them, and what factors in your system can affect the statistics. This section contains the following topics:

- "When DB2 externalizes real-time statistics"
- "How DB2 utilities affect the real-time statistics" on page 1244
- "How non-DB2 utilities affect real-time statistics" on page 1251
- "Real-time statistics on objects in work file databases and the TEMP database" on page 1251
- "Real-time statistics on read-only or nonmodified objects" on page 1251
- "How dropping objects affects real-time statistics" on page 1251
- "How SQL operations affect real-time statistics counters" on page 1252
- "Real-time statistics in data sharing" on page 1253
- "Improving concurrency with real-time statistics" on page 1253
- "Recovering the real-time statistics tables" on page 1253
- "Statistics accuracy" on page 1254

When DB2 externalizes real-time statistics

DB2 externalizes real-time statistics in the following circumstances:

- When you issue STOP DATABASE(DSNRTSDB)

  This command stops the in-memory statistics database and externalizes statistics for all objects in the subsystem.
- When you issue STOP DATABASE(*database-name*) SPACENAM(*space-name*)

  This command externalizes statistics only for *database-name* and *space-name*.
- At the end of the time interval that you specify during installation

  See "Setting the interval for writing real-time statistics" on page 1231 for information about how to set this time interval.
- When you issue STOP DB2 MODE(QUIESCE)

  DB2 writes any statistics that are in memory when you issue this command to the statistics tables. However, if you issue STOP DB2 MODE(FORCE), DB2 does not write the statistics, and you lose them.
- During utility operations

  "How DB2 utilities affect the real-time statistics" on page 1244 gives details on how the utilities modify the statistics tables.

DB2 does not maintain real-time statistics for any objects in the real-time statistics database. Therefore, if you run a utility with a utility list, and the list contains any real-time statistics objects, DB2 does not externalize real-time statistics during the execution of that utility for any of the objects in the utility list. DB2 does not maintain interval counter real-time statistics for SYSLGRNG and its indexes during utility operation. DB2 maintains statistics for these objects only during non-utility operation.

Recommendation: Do not include real-time statistics objects in utility lists.

DB2 does not externalize real-time statistics at a tracker site.

How DB2 utilities affect the real-time statistics

In general, SQL INSERT, UPDATE, and DELETE statements cause DB2 to modify the real-time statistics. However, certain DB2 utilities also affect the statistics. The following sections discuss the effect of each of those utilities on the statistics:

- "How LOAD affects real-time statistics"
- "How REORG affects real-time statistics" on page 1246
- "How REBUILD INDEX affects real-time statistics" on page 1248
- "How RUNSTATS affects real-time statistics" on page 1249
- "How COPY affects real-time statistics" on page 1250
- "How RECOVER affects real-time statistics" on page 1251

How LOAD affects real-time statistics

Table 327 shows how running LOAD REPLACE on a table space or table space partition affects the TABLESPACESTATS statistics.

Table 327. Changed TABLESPACESTATS values during LOAD

Column name	Settings for LOAD REPLACE after RELOAD phase
TOTALROWS	Number of loaded rows or LOBs[1]
NACTIVE	Actual value
SPACE	Actual value
EXTENTS	Actual value
LOADRLASTTIME	Current timestamp
REORGINSERTS	0
REORGDELETES	0
REORGUPDATES	0
REORGDISORGLOB	0
REORGUNCLUSTINS	0
REORGMASSDELETE	0
REORGNEARINDREF	0
REORGFARINDEF	0
STATSLASTTIME	Current timestamp[2]
STATSINSERTS	0[2]
STATSDELETES	0[2]
STATSUPDATES	0[2]
STATSMASSDELETE	0[2]

Table 327. Changed TABLESPACESTATS values during LOAD (continued)

Column name	Settings for LOAD REPLACE after RELOAD phase
COPYLASTTIME	Current timestamp[3]
COPYUPDATEDPAGES	0[3]
COPYCHANGES	0[3]
COPYUPDATELRSN	Null[3]
COPYUPDATETIME	Null[3]

Notes:

1. Under certain conditions, such as a utility restart, the LOAD utility might not have an accurate count of loaded records. In those cases, DB2 sets this value to null. Some rows that are loaded into a table space and are included in this value might later be removed during the index validation phase or the referential integrity check. DB2 includes counts of those removed records in the statistics that record deleted records.
2. DB2 sets this value only if the LOAD invocation includes the STATISTICS option.
3. DB2 sets this value only if the LOAD invocation includes the COPYDDN option.

Table 328 shows how running LOAD REPLACE affects the INDEXSPACESTATS statistics for an index space or physical index partition.

Table 328. Changed INDEXSPACESTATS values during LOAD REPLACE

Column name	Settings for LOAD REPLACE after BUILD phase
TOTALENTRIES	Number of index entries added[1]
NLEVELS	Actual value
NACTIVE	Actual value
SPACE	Actual value
EXTENTS	Actual value
LOADRLASTTIME	Current timestamp
REORGINSERTS	0
REORGDELETES	0
REORGAPPENDINSERT	0
REORGPSEUDODELETES	0
REORGMASSDELETE	0
REORGLEAFNEAR	0
REORGLEAFFAR	0
REORGNUMLEVELS	0
STATSLASTTIME	Current timestamp[2]
STATSINSERTS	0[2]
STATSDELETES	0[2]
STATSMASSDELETE	0[2]
COPYLASTTIME	Current timestamp[3]
COPYUPDATEDPAGES	0[3]
COPYCHANGES	0[3]
COPYUPDATELRSN	Null[3]

Table 328. Changed INDEXSPACESTATS values during LOAD REPLACE (continued)

Column name	Settings for LOAD REPLACE after BUILD phase
COPYUPDATETIME	Null[3]

Notes:

1. Under certain conditions, such as a utility restart, the LOAD utility might not have an accurate count of loaded records. In those cases, DB2 sets this value to null.
2. DB2 sets this value only if the LOAD invocation includes the STATISTICS option.
3. DB2 sets this value only if the LOAD invocation includes the COPYDDN option.

For a logical index partition:

- A LOAD operation without the REPLACE option behaves similar to a SQL INSERT operation in that the number of records loaded are counted in the incremental counters such as REORGINSERTS, REORGAPPENDINSERT, STATSINSERTS, and COPYCHANGES. A LOAD operation without the REPLACE option affects the organization of the data and can be a trigger to run REORG, RUNSTATS or COPY.
- DB2 does not reset the nonpartitioned index when it does a LOAD REPLACE on a partition. Therefore, DB2 does not reset the statistics for the index. The REORG counters from the last REORG are still correct. DB2 updates LOADRLASTTIME when the entire nonpartitioned index is replaced.
- When DB2 does a LOAD RESUME YES on a partition, after the BUILD phase, DB2 increments TOTALENTRIES by the number of index entries that were inserted during the BUILD phase.

How REORG affects real-time statistics

Table 329 shows how running REORG on a table space or table space partition affects the TABLESPACESTATS statistics.

Table 329. Changed TABLESPACESTATS values during REORG

Column name	Settings for REORG SHRLEVEL NONE after RELOAD phase	Settings for REORG SHRLEVEL REFERENCE or CHANGE after SWITCH phase
TOTALROWS	Number rows or LOBs loaded[1]	For SHRLEVEL REFERENCE: Number of loaded rows or LOBs during RELOAD phase For SHRLEVEL CHANGE: Number of loaded rows or LOBs during RELOAD phase plus number of rows inserted during LOG APPLY phase minus number of rows deleted during LOG phase
NACTIVE	Actual value	Actual value
SPACE	Actual value	Actual value
EXTENTS	Actual value	Actual value
REORGLASTTIME	Current timestamp	Current timestamp
REORGINSERTS	0	Actual value[2]
REORGDELETES	0	Actual value[2]
REORGUPDATES	0	Actual value[2]
REORGDISORGLOB	0	Actual value[2]

Table 329. Changed TABLESPACESTATS values during REORG (continued)

Column name	Settings for REORG SHRLEVEL NONE after RELOAD phase	Settings for REORG SHRLEVEL REFERENCE or CHANGE after SWITCH phase
REORGUNCLUSTINS	0	Actual value[2]
REORGMASSDELETE	0	Actual value[2]
REORGNEARINDREF	0	Actual value[2]
REORGFARINDEF	0	Actual value[2]
STATSLASTTIME	Current timestamp[3]	Current timestamp[3]
STATSINSERTS	0[3]	Actual value[2]
STATSDELETES	0[3]	Actual value[2]
STATSUPDATES	0[3]	Actual value[2]
STATSMASSDELETE	0[3]	Actual value[2]
COPYLASTTIME	Current timestamp[4]	Current timestamp
COPYUPDATEDPAGES	0[4]	Actual value[2]
COPYCHANGES	0[4]	Actual value[2]
COPYUPDATELRSN	Null[4]	Actual value[5]
COPYUPDATETIME	Null[4]	Actual value[5]

Notes:

1. Under certain conditions, such as a utility restart, the REORG utility might not have an accurate count of loaded records. In those cases, DB2 sets this value to null. Some rows that are loaded into a table space and are included in this value might later be removed during the index validation phase or the referential integrity check. DB2 includes counts of those removed records in the statistics that record deleted records.
2. This is the actual number of inserts, updates, or deletes that are due to applying the log to the shadow copy.
3. DB2 sets this value only if the REORG invocation includes the STATISTICS option.
4. DB2 sets this value only if the REORG invocation includes the COPYDDN option.
5. This is the LRSN or timestamp for the first update that is due to applying the log to the shadow copy.

Table 330 shows how running REORG affects the INDEXSPACESTATS statistics for an index space or physical index partition.

Table 330. Changed INDEXSPACESTATS values during REORG

Column name	Settings for REORG SHRLEVEL NONE after RELOAD phase	Settings for REORG SHRLEVEL REFERENCE or CHANGE after SWITCH phase
TOTALENTRIES	Number of index entries added[1]	For SHRLEVEL REFERENCE: Number of added index entries during BUILD phase For SHRLEVEL CHANGE: Number of added index entries during BUILD phase plus number of added index entries during LOG phase minus number of deleted index entries during LOG phase
NLEVELS	Actual value	Actual value
NACTIVE	Actual value	Actual value
SPACE	Actual value	Actual value
EXTENTS	Actual value	Actual value
REORGLASTTIME	Current timestamp	Current timestamp

Table 330. Changed INDEXSPACESTATS values during REORG (continued)

Column name	Settings for REORG SHRLEVEL NONE after RELOAD phase	Settings for REORG SHRLEVEL REFERENCE or CHANGE after SWITCH phase
REORGINSERTS	0	Actual value[2]
REORGDELETES	0	Actual value[2]
REORGAPPENDINSERT	0	Actual value[2]
REORGPSEUDODELETES	0	Actual value[2]
REORGMASSDELETE	0	Actual value[2]
REORGLEAFNEAR	0	Actual value[2]
REORGLEAFFAR	0	Actual value[2]
REORGNUMLEVELS	0	Actual value[2]
STATSLASTTIME	Current timestamp[3]	Current timestamp[3]
STATSINSERTS	0[3]	Actual value[2]
STATSDELETES	0[3]	Actual value[2]
STATSMASSDELETE	0[3]	Actual value[2]
COPYLASTTIME	Current timestamp[4]	Unchanged[5]
COPYUPDATEDPAGES	0[4]	Unchanged[5]
COPYCHANGES	0[4]	Unchanged[5]
COPYUPDATELRSN	Null[4]	Unchanged[5]
COPYUPDATETIME	Null[4]	Unchanged[5]

Notes:

1. Under certain conditions, such as a utility restart, the REORG utility might not have an accurate count of loaded records. In those cases, DB2 sets this value to null.
2. This is the actual number of inserts, updates, or deletes that are due to applying the log to the shadow copy.
3. DB2 sets this value only if the REORG invocation includes the STATISTICS option.
4. DB2 sets this value only if the REORG invocation includes the COPYDDN option.
5. Inline COPY is not allowed for SHRLEVEL CHANGE or SHRLEVEL REFERENCE.

For a logical index partition, DB2 does not reset the nonpartitioned index when it does a REORG on a partition. Therefore, DB2 does not reset the statistics for the index. The REORG counters and REORGLASTTIME are relative to the last time the entire nonpartitioned index is reorganized. In addition, the REORG counters might be low because, due to the methodology, some index entries are changed during REORG of a partition.

How REBUILD INDEX affects real-time statistics

Table 331 shows how running REBUILD INDEX affects the INDEXSPACESTATS statistics for an index space or physical index partition.

Table 331. Changed INDEXSPACESTATS values during REBUILD INDEX

Column name	Settings after BUILD phase
TOTALENTRIES	Number of index entries added[1]
NLEVELS	Actual value
NACTIVE	Actual value
SPACE	Actual value

Table 331. Changed INDEXSPACESTATS values during REBUILD INDEX (continued)

Column name	Settings after BUILD phase
EXTENTS	Actual value
REBUILDLASTTIME	Current timestamp
REORGINSERTS	0
REORGDELETES	0
REORGAPPENDINSERT	0
REORGPSEUDODELETES	0
REORGMASSDELETE	0
REORGLEAFNEAR	0
REORGLEAFFAR	0
REORGNUMLEVELS	0

Note:

1. Under certain conditions, such as a utility restart, the REBUILD utility might not have an accurate count of loaded records. In those cases, DB2 sets this value to null.

For a logical index partition, DB2 does not collect TOTALENTRIES statistics for the entire nonpartitioned index when it runs REBUILD INDEX. Therefore, DB2 does not reset the statistics for the index. The REORG counters from the last REORG are still correct. DB2 updates REBUILDLASTTIME when the entire nonpartitioned index is rebuilt.

How RUNSTATS affects real-time statistics

Only RUNSTATS UPDATE ALL affects the real-time statistics. When the RUNSTATS job starts, DB2 externalizes all in-memory statistics to the real-time statistics tables.

Table 332 shows how running RUNSTATS UPDATE ALL on a table space or table space partition affects the TABLESPACESTATS statistics.

Table 332. Changed TABLESPACESTATS values during RUNSTATS UPDATE ALL

Column name	During UTILINIT phase	After RUNSTATS phase
STATSLASTTIME	Current timestamp[1]	Timestamp of the start of RUNSTATS phase
STATSINSERTS	Actual value[1]	Actual value[2]
STATSDELETES	Actual value[1]	Actual value[2]
STATSUPDATES	Actual value[1]	Actual value[2]
STATSMASSDELETE	Actual value[1]	Actual value[2]

Notes:

1. DB2 externalizes the current in-memory values.
2. This value is 0 for SHRLEVEL REFERENCE, or the actual value for SHRLEVEL CHANGE.

Table 333 on page 1250 shows how running RUNSTATS UPDATE ALL on an index affects the INDEXSPACESTATS statistics.

Table 333. Changed INDEXSPACESTATS values during RUNSTATS UPDATE ALL

Column name	During UTILINIT phase	After RUNSTATS phase
STATSLASTTIME	Current timestamp[1]	Timestamp of the start of RUNSTATS phase
STATSINSERTS	Actual value[1]	Actual value[2]
STATSDELETES	Actual value[1]	Actual value[2]
STATSMASSDELETE	Actual value[1]	Actual value[2]

Notes:

1. DB2 externalizes the current in-memory values.
2. This value is 0 for SHRLEVEL REFERENCE, or the actual value for SHRLEVEL CHANGE.

How COPY affects real-time statistics

When a COPY job starts, DB2 externalizes all in-memory statistics to the real-time statistics tables. Statistics are gathered for a full image copy or an incremental copy, but not for a data set copy.

Table 334 shows how running COPY on a table space or table space partition affects the TABLESPACESTATS statistics.

Table 334. Changed TABLESPACESTATS values during COPY

Column name	During UTILINIT phase	After COPY phase
COPYLASTTIME	Current timestamp[1]	Timestamp of the start of COPY phase
COPYUPDATEDPAGES	Actual value[1]	Actual value[2]
COPYCHANGES	Actual value[1]	Actual value[2]
COPYUPDATELRSN	Actual value[1]	Actual value[3]
COPYUPDATETIME	Actual value[1]	Actual value[3]

Notes:

1. DB2 externalizes the current in-memory values.
2. This value is 0 for SHRLEVEL REFERENCE, or the actual value for SHRLEVEL CHANGE.
3. This value is null for SHRLEVEL REFERENCE, or the actual value for SHRLEVEL CHANGE.

Table 335 shows how running COPY on an index affects the INDEXSPACESTATS statistics.

Table 335. Changed INDEXSPACESTATS values during COPY

Column name	During UTILINIT phase	After COPY phase
COPYLASTTIME	Current timestamp[1]	Timestamp of the start of COPY phase
COPYUPDATEDPAGES	Actual value[1]	Actual value[2]
COPYCHANGES	Actual value[1]	Actual value[2]
COPYUPDATELRSN	Actual value[1]	Actual value[3]
COPYUPDATETIME	Actual value[1]	Actual value[3]

Table 335. Changed INDEXSPACESTATS values during COPY (continued)

Column name	During UTILINIT phase	After COPY phase

Note:

1. DB2 externalizes the current in-memory values.
2. This value is 0 for SHRLEVEL REFERENCE, or the actual value for SHRLEVEL CHANGE.
3. This value is null for SHRLEVEL REFERENCE, or the actual value for SHRLEVEL CHANGE.

How RECOVER affects real-time statistics

After recovery to the current state, the in-memory counter fields are still valid, so DB2 does not modify them. However, after a point-in-time recovery, the statistics might not be valid. DB2 therefore sets all the REORG, STATS, and COPY counter statistics to null after a point-in-time recovery. After recovery to the current state, DB2 sets NACTIVE, SPACE, and EXTENTS to their new values. After a point-in-time recovery, DB2 sets NLEVELS, NACTIVE, SPACE, and EXTENTS to their new values.

How non-DB2 utilities affect real-time statistics

Non-DB2 utilities do not affect real-time statistics. Therefore, an object that is the target of a non-DB2 COPY, LOAD, REBUILD, REORG, or RUNSTATS job can cause incorrect statistics to be inserted in the real-time statistics tables. Follow this process to ensure correct statistics when you run non-DB2 utilities:

1. Stop the table space or index on which you plan to run the utility. This action causes DB2 to write the in-memory statistics to the real-time statistics tables and initialize the in-memory counters. If DB2 cannot externalize the statistics, the STOP command does not fail.
2. Run the utility.
3. When the utility completes, update the statistics tables with new totals and timestamps, and put zero values in the incremental counter.

Real-time statistics on objects in work file databases and the TEMP database

Although you cannot run utilities on objects in the work files databases and TEMP database, DB2 records the NACTIVE, SPACE, and EXTENTS statistics on table spaces in those databases.

Real-time statistics for DEFINE NO objects

For objects that are created with DEFINE NO, no row is inserted into the real-time statistics table until the object is physically defined.

Real-time statistics on read-only or nonmodified objects

DB2 does not externalize the NACTIVE, SPACE, or EXTENTS statistics for read-only objects or objects that are not modified.

How dropping objects affects real-time statistics

If you drop a table space or index, DB2 deletes its statistics from the real-time statistics tables. However, if the real-time statistics database is not available when you drop a table space or index, the statistics remain in the real-time statistics

tables, even though the corresponding object no longer exists. You need to use SQL DELETE statements to manually remove those rows from the real-time statistics tables.

If a row still exists in the real-time statistics tables for a dropped table space or index, and if you create a new object with the same DBID and PSID as the dropped object, DB2 reinitializes the row before it updates any values in that row.

How SQL operations affect real-time statistics counters

SQL operations affect the counter columns in the real-time statistics tables. These are the columns that record the number of insert, delete, or update operations, as well as the total counters, TOTALROWS, and TOTALENTRIES.

UPDATE: When you perform an UPDATE, DB2 increments the update counters.

INSERT: When you perform an INSERT, DB2 increments the insert counters. DB2 keeps separate counters for clustered and unclustered INSERTs.

DELETE: When you perform a DELETE, DB2 increments the delete counters.

ROLLBACK: When you perform a ROLLBACK, DB2 increments the counters, depending on the type of SQL operation that is rolled back:

Rolled-back SQL statement	Incremented counters
UPDATE	Update counters
INSERT	Delete counters
DELETE	Insert counters

Notice that for INSERT and DELETE, the counter for the inverse operation is incremented. For example, if two INSERT statements are rolled back, the delete counter is incremented by 2.

UPDATE of partitioning keys: If an update to a partitioning key causes rows to move to a new partition, the following real-time statistics are impacted:

Action	Incremented counters
When UPDATE is executed	Update count of old partition = +1 Insert count of new partition = +1
When UPDATE is committed	Delete count of old partition = +1
When UPDATE is rolled back	Update count of old partition = +1 (compensation log record) Delete count of new partition = +1 (remove inserted record)

If an update to a partitioning key does not cause rows to move to a new partition, the counts are accumulated as expected:

Action	Incremented counters
When UPDATE is executed	Update count of current partition = +1 NEAR/FAR indirect reference count = +1 (if overflow occured)

Action	Incremented counters
When UPDATE is rolled back	Update count of current partition = +1 (compensation log record)

Mass DELETE:Performing a mass delete operation on a table space does not cause DB2 to reset the counter columns in the real-time statistics tables. After a mass delete operation, the value in a counter column includes the count from a time prior to the mass delete operation, as well as the count after the mass delete operation.

Real-time statistics in data sharing

In a data sharing environment, DB2 members update their statistics serially. Each member reads the target row from the statistics table, obtains a lock, aggregates its in-memory statistics, and updates the statistics table with the new totals. Each member sets its own interval for writing real-time statistics.

DB2 does locking based on the lock size of the DSNRTSDB.DSNRTSTS table space. DB2 uses cursor stability isolation and CURRENTDATA(YES) when it reads the statistics tables.

At the beginning of a RUNSTATS job, all data sharing members externalize their statistics to the real-time statistics tables and reset their in-memory statistics. If all members cannot externalize their statistics, DB2 sets STATSLASTTIME to null. An error in gathering and externalizing statistics does not prevent RUNSTATS from running.

At the beginning of a COPY job, all data sharing members externalize their statistics to the real-time statistics tables and reset their in-memory statistics. If all members cannot externalize their statistics, DB2 sets COPYLASTTIME to null. An error in gathering and externalizing statistics does not prevent COPY from running.

Utilities that reset page sets to empty can invalidate the in-memory statistics of other DB2 members. The member that resets a page set notifies the other DB2 members that a page set has been reset to empty, and the in-memory statistics are invalidated. If the notify process fails, the utility that resets the page set does not fail. DB2 sets the appropriate timestamp (REORGLASTTIME, STATSLASTTIME, or COPYLASTTIME) to null in the row for the empty page set to indicate that the statistics for that page set are unknown.

Improving concurrency with real-time statistics

Follow these recommendations to reduce the risk of timeouts and deadlocks when you work with the real-time statistics tables:

- When you run COPY, RUNSTATS, or REORG on the real-time statistics objects, use SHRLEVEL CHANGE.
- When you execute SQL statements to query the real-time statistics tables, use uncommitted read isolation.

Recovering the real-time statistics tables

When you recover a DB2 subsystem after a disaster, DB2 starts with the ACESS(MAINT) option. No statistics are externalized in this state. Therefore, you need to perform the following actions on the real-time statistics database:

- Recover the real-time statistics objects after you recover the DB2 catalog and directory.
- Start the real-time statistics database explicitly, after DB2 restart.

Statistics accuracy

In general, the real-time statistics are accurate values. However, several factors can affect the accuracy of the statistics:

- Certain utility restart scenarios
- Certain utility operations that leave indexes in a database restrictive state, such as RECOVER-pending (RECP)

  Always consider the database restrictive state of objects before accepting a utility recommendation that is based on real-time statistics.
- A DB2 subsystem failure
- A notify failure in a data sharing environment

If you think that some statistics values might be inaccurate, you can correct the statistics by running REORG, RUNSTATS, or COPY on the objects for which DB2 generated the statistics.

Appendix J. DB2-supplied stored procedures

DB2 provides several stored procedures that you can call in your application programs to perform a number of utility and application programming functions. Those stored procedures are:

- The utilities stored procedure for EBCDIC input (DSNUTILS)

  This stored procedure lets you invoke utilities from a local or remote client program. See Appendix B of *DB2 Utility Guide and Reference* for information.
- The utilities stored procedure for Unicode input (DSNUTILU)

  This stored procedure lets you invoke utilities from a local or remote client program that generates Unicode utility control statements. See Appendix B of *DB2 Utility Guide and Reference* for information.
- The DB2 Universal Database Control Center (Control Center) table space and index information stored procedure (DSNACCQC)

  This stored procedure helps you determine when utilities should be run on your databases. This stored procedure is designed primarily for use by the Control Center but can be invoked from any client program. See Appendix B ofAppendix B of *DB2 Utility Guide and Reference* for information.
- The Control Center partition information stored procedure (DSNACCAV)

  This stored procedure helps you determine when utilities should be run on your partitioned table spaces. This stored procedure is designed primarily for use by the Control Center but can be invoked from any client program. See Appendix B of *DB2 Utility Guide and Reference* for information.
- The real-time statistics stored procedure (DSNACCOR)

  This stored procedure queries the DB2 real-time statistics tables to help you determine when you should run COPY, REORG, or RUNSTATS, or enlarge your DB2 data sets. See "The DB2 real-time statistics stored procedure" on page 1257 for more information.
- The WLM environment refresh stored procedure (WLM_REFRESH)

  This stored procedure lets you refresh a WLM environment from a remote workstation. See "WLM environment refresh stored procedure (WLM_REFRESH)" on page 1278 for information.
- The CICS transaction invocation stored procedure (DSNACICS)

  This stored procedure lets you invoke CICS transactions from a remote workstation. See "The CICS transaction invocation stored procedure (DSNACICS)" on page 1280 for more information.
- The SYSIBM.USERNAMES encryption stored procedure (DSNLEUSR)

  This stored procedure lets you store encrypted values in the NEWAUTHID and PASSWORD fields of the SYSIBM.USERNAMES catalog table. See "The SYSIBM.USERNAMES encryption stored procedure" on page 1287 for more information.
- The IMS transactions stored procedure (DSNAIMS)

  This stored procedure allows DB2 to invoke IMS transactions and commands easily, without maintaining their own connections to IMS. See "IMS transactions stored procedure (DSNAIMS)" on page 1291 for more information.
- The EXPLAIN stored procedure (DSN8EXP)

© Copyright IBM Corp. 1982, 2008

This stored procedure allows a user to perform an EXPLAIN on an SQL statement without having the authorization to execute that SQL statement.See "The DB2 EXPLAIN stored procedure" on page 1295for more information.

- The MQ XML stored procedures

  All of the MQ XML stored procedures have been deprecated.

  These stored procedures perform the following functions:

Table 336. MQ XML stored procedures

Stored procedure name	Function	For information, see:
DXXMQINSERT	Returns a message that contains an XML document from an MQ message queue, decomposes the document, and stores the data in DB2 tables that are specified by an enabled XML collection.	*DB2 Application Programming and SQL Guide*
DXXMQSHRED	Returns a message that contains an XML document from an MQ message queue, decomposes the document, and stores the data in DB2 tables that are specified in a document access definition (DAD) file. DXXMQSHRED does not require an enabled XML collection.	*DB2 Application Programming and SQL Guide*
DXXMQINSERTCLOB	Returns a message that contains an XML document from an MQ message queue, decomposes the document, and stores the data in DB2 tables that are specified by an enabled XML collection. DXXMQINSERTCLOB is intended for an XML document with a length of up to 1MB.	*DB2 Application Programming and SQL Guide*
DXXMQSHREDCLOB	Returns a message that contains an XML document from an MQ message queue, decomposes the document, and stores the data in DB2 tables that are specified in a document access definition (DAD) file. DXXMQSHREDCLOB does not require an enabled XML collection. DXXMQSHREDCLOB is intended for an XML document with a length of up to 1MB.	*DB2 Application Programming and SQL Guide*
DXXMQINSERTALL	Returns messages that contains XML documents from an MQ message queue, decomposes the documents, and stores the data in DB2 tables that are specified by an enabled XML collection. DXXMQINSERTALL is intended for XML documents with a length of up to 3KB.	*DB2 Application Programming and SQL Guide*
DXXMQSHREDALL	Returns messages that contain XML documents from an MQ message queue, decomposes the documents, and stores the data in DB2 tables that are specified in a document access definition (DAD) file. DXXMQSHREDALL does not require an enabled XML collection. DXXMQSHREDALL is intended for XML documents with a length of up to 3KB.	*DB2 Application Programming and SQL Guide*

Table 336. MQ XML stored procedures (continued)

Stored procedure name	Function	For information, see:
DXXMQSHREDALLCLOB	Returns messages that contain XML documents from an MQ message queue, decomposes the documents, and stores the data in DB2 tables that are specified in a document access definition (DAD) file. DXXMQSHREDALLCLOB does not require an enabled XML collection. DXXMQSHREDALLCLOB is intended for XML documents with a length of up to 1MB.	*DB2 Application Programming and SQL Guide*
DXXMQINSERTALLCLOB	Returns messages that contains XML documents from an MQ message queue, decomposes the documents, and stores the data in DB2 tables that are specified by an enabled XML collection. DXXMQINSERTALLCLOB is intended for XML documents with a length of up to 1MB.	*DB2 Application Programming and SQL Guide*
DXXMQGEN	Constructs XML documents from data that is stored in DB2 tables that are specified in a document access definition (DAD) file, and sends the XML documents to an MQ message queue. DXXMQGEN is intended for XML documents with a length of up to 3KB.	*DB2 Application Programming and SQL Guide*
DXXMQRETRIEVE	Constructs XML documents from data that is stored in DB2 tables that are specified in an enabled XML collection, and sends the XML documents to an MQ message queue. DXXMQRETRIEVE is intended for XML documents with a length of up to 3KB.	*DB2 Application Programming and SQL Guide*
DXXMQGENCLOB	Constructs XML documents from data that is stored in DB2 tables that are specified in a document access definition (DAD) file, and sends the XML documents to an MQ message queue. DXXMQGENCLOB is intended for XML documents with a length of up to 32KB.	*DB2 Application Programming and SQL Guide*
DXXMQRETRIEVECLOB	Constructs XML documents from data that is stored in DB2 tables that are specified in an enabled XML collection, and sends the XML documents to an MQ message queue. DXXMQRETRIEVECLOB is intended for XML documents with a length of up to 32KB.	*DB2 Application Programming and SQL Guide*

The DB2 real-time statistics stored procedure

The information under this heading is Product-sensitive Programming Interface and Associated Guidance Information.

The DSNACCOR stored procedure is a sample stored procedure that makes recommendations to help you maintain your DB2 databases. In particular, DSNACCOR performs these actions:

- Recommends when you should reorganize, image copy, or update statistics for table spaces or index spaces
- Indicates table spaces or index spaces that have exceeded their data set
- Indicates whether objects are in a restricted state

DSNACCOR uses data from the SYSIBM.TABLESPACESTATS and SYSIBM.INDEXSPACESTATS real-time statistics tables to make its recommendations. DSNACCOR provides its recommendations in a result set.

DSNACCOR uses the set of criteria that are shown in "DSNACCOR formulas for recommending actions" on page 1267 to evaluate table spaces and index spaces. By default, DSNACCOR evaluates **all** table spaces and index spaces in the subsystem that have entries in the real-time statistics tables. However, you can override this default through input parameters.

Important information about DSNACCOR recommendations:

- DSNACCOR makes recommendations based on general formulas that require input from the user about the maintenance policies for a subsystem. These recommendations might not be accurate for every installation.
- If the real-time statistics tables contain information for only a small percentage of your DB2 subsystem, the recommendations that DSNACCOR makes might not be accurate for the entire subsystem.
- Before you perform any action that DSNACCOR recommends, ensure that the object for which DSNACCOR makes the recommendation is available, and that the recommended action can be performed on that object. For example, before you can perform an image copy on an index, the index must have the COPY YES attribute.

Environment for DSNACCOR

DSNACCOR must run in a WLM-established stored procedure address space.

DSNACCOR creates and uses declared temporary tables. Therefore, before you can invoke DSNACCOR, you need to create a TEMP database and segmented table spaces in the TEMP database. For information about creating TEMP databases and table spaces, see CREATE DATABASE and CREATE TABLESPACE Chapter 5 of *DB2 SQL Reference*.

Before you can invoke DSNACCOR, the real-time statistics tables, SYSIBM.TABLESPACESTATS and SYSIBM.INDEXSPACESTATS, must exist, and the real-time statistics database must be started. See Appendix I, "Real-time statistics tables," on page 1229 for information about the real-time statistics tables.

You should bind the package for DSNACCOR with isolation UR to avoid lock contention. You can find the installation steps for DSNACCOR in job DSNTIJSG.

Authorization required for DSNACCOR

To execute the CALL DSNACCOR statement, the owner of the package or plan that contains the CALL statement must have one or more of the following privileges on each package that the stored procedure uses:

- The EXECUTE privilege on the package for DSNACCOR
- Ownership of the package
- PACKADM authority for the package collection
- SYSADM authority

The owner of the package or plan that contains the CALL statement must also have:

- SELECT authority on the real-time statistics tables
- The DISPLAY system privilege

DSNACCOR syntax diagram

The following syntax diagram shows the CALL statement for invoking DSNACCOR. Because the linkage convention for DSNACCOR is GENERAL WITH NULLS, if you pass parameters in host variables, you need to include a null indicator with every host variable. Null indicators for input host variables must be initialized before you execute the CALL statement.

```
>>─CALL─DSNACCOR─(─┬─QueryType──┬─,─┬─ObjectType─┬─,─┬─ICType─┬─,─┬─StatsSchema─┬─,──────────>
                   └─NULL───────┘   └─NULL───────┘   └─NULL───┘   └─NULL───────┘

>─┬─CatlgSchema─┬─,─┬─LocalSchema─┬─,─┬─ChkLvl─┬─,─┬─Criteria─┬─,─┬─Restricted─┬─,──────────>
  └─NULL───────┘   └─NULL───────┘   └─NULL───┘   └─NULL───┘   └─NULL───────┘

>─┬─CRUpdatedPagesPct─┬─,─┬─CRChangesPct─┬─,─┬─CRDaySncLastCopy─┬─,─┬─ICRUpdatedPagesPct─┬─,──>
  └─NULL─────────────┘   └─NULL─────────┘   └─NULL─────────────┘   └─NULL─────────────┘

>─┬─ICRChangesPct─┬─,─┬─CRIndexSize─┬─,─┬─RRTInsDelUpdPct─┬─,─┬─RRTUnclustInsPct─┬─,──────>
  └─NULL─────────┘   └─NULL───────┘   └─NULL─────────────┘   └─NULL─────────────┘

>─┬─RRTDisorgLOBPct─┬─,─┬─RRTMassDelLimit─┬─,─┬─RRTIndRefLimit─┬─,─┬─RRIInsertDeletePct─┬─,──>
  └─NULL───────────┘   └─NULL─────────────┘   └─NULL─────────────┘   └─NULL───────────────┘

>─┬─RRIAppendInsertPct─┬─,─┬─RRIPseudoDeletePct─┬─,─┬─RRIMassDelLimit─┬─,─┬─RRILeafLimit─┬─,──>
  └─NULL─────────────┘   └─NULL─────────────┘   └─NULL─────────────┘   └─NULL─────────┘

>─┬─RRINumLevelsLimit─┬─,─┬─SRTInsDelUpdPct─┬─,─┬─SRTInsDelUpdAbs─┬─,─┬─SRTMassDelLimit─┬─,──>
  └─NULL─────────────┘   └─NULL─────────────┘   └─NULL─────────────┘   └─NULL─────────────┘

>─┬─SRIInsDelUpdPct─┬─,─┬─SRIInsDelUpdAbs─┬─,─┬─SRIMassDelLimit─┬─,─┬─ExtentLimit─┬─,──────>
  └─NULL───────────┘   └─NULL─────────────┘   └─NULL─────────────┘   └─NULL───────┘

>─LastStatement,─ReturnCode,─ErrorMsg,─IFCARetCode,─IFCAResCode,─ExcessBytes─)──────────><
```

DSNACCOR option descriptions

In the following option descriptions, the default value for an input parameter is the value that DSNACCOR uses if you specify a null value.

QueryType

Specifies the types of actions that DSNACCOR recommends. This field contains one or more of the following values. Each value is enclosed in single quotation marks and separated from other values by a space.

ALL Makes recommendations for all of the following actions.

COPY Makes a recommendation on whether to perform an image copy.

RUNSTATS Makes a recommendation on whether to perform RUNSTATS.

REORG Makes a recommendation on whether to perform REORG. Choosing this value causes DSNACCOR to process the EXTENTS value also.

EXTENTS Indicates when data sets have exceeded a user-specified extents limit.

RESTRICT Indicates which objects are in a restricted state.

QueryType is an input parameter of type VARCHAR(40). The default is **ALL**.

ObjectType
Specifies the types of objects for which DSNACCOR recommends actions:

ALL Table spaces and index spaces.

TS Table spaces only.

IX Index spaces only.

ObjectType is an input parameter of type VARCHAR(3). The default is **ALL**.

ICType
Specifies the types of image copies for which DSNACCOR is to make recommendations:

F Full image copy.

I Incremental image copy. This value is valid for table spaces only.

B Full image copy or incremental image copy.

ICType is an input parameter of type VARCHAR(1). The default is **B**.

StatsSchema
Specifies the qualifier for the real-time statistics table names. *StatsSchema* is an input parameter of type VARCHAR(128). The default is **SYSIBM**.

CatlgSchema
Specifies the qualifier for DB2 catalog table names. *CatlgSchema* is an input parameter of type VARCHAR(128). The default is **SYSIBM**.

LocalSchema
Specifies the qualifier for the names of tables that DSNACCOR creates. *LocalSchema* is an input parameter of type VARCHAR(128). The default is **DSNACC**.

ChkLvl
Specifies the types of checking that DSNACCOR performs, and indicates whether to include objects that fail those checks in the DSNACCOR recommendations result set. This value is the sum of any combination of the following values:

0 DSNACCOR performs none of the following actions.

1 For objects that are listed in the recommendations result set, check the SYSTABLESPACE or SYSINDEXES catalog tables to ensure that those objects have not been deleted. If value 16 is **not** also chosen, exclude rows for the deleted objects from the recommendations result set.

DSNACCOR excludes objects from the recommendations result set if those objects are not in the SYSTABLESPACE or SYSINDEXES catalog tables.

When this setting is specified, DSNACCOR does not use EXTENTS>*ExtentLimit* to determine whether a LOB table space should be reorganized.

2 For index spaces that are listed in the recommendations result set, check the SYSTABLES, SYSTABLESPACE, and SYSINDEXES catalog tables to determine the name of the table space that is associated with each index space.

Choosing this value causes DSNACCOR to also check for rows in the recommendations result set for objects that have been deleted but have entries in the real-time statistics tables (value 1). This means that if value 16 is **not** also chosen, rows for deleted objects are excluded from the recommendations result set.

4 Check whether rows that are in the DSNACCOR recommendations result set refer to objects that are in the exception table. For recommendations result set rows that have corresponding exception table rows, copy the contents of the QUERYTYPE column of the exception table to the INEXCEPTTABLE column of the recommendations result set.

8 Check whether objects that have rows in the recommendations result set are restricted. Indicate the restricted status in the OBJECTSTATUS column of the result set.

16 For objects that are listed in the recommendations result set, check the SYSTABLESPACE or SYSINDEXES catalog tables to ensure that those objects have not been deleted (value 1). In result set rows for deleted objects, specify the word ORPHANED in the OBJECTSTATUS column.

32 Exclude rows from the DSNACCOR recommendations result set for index spaces for which the related table spaces have been recommended for REORG. Choosing this value causes DSNACCOR to perform the actions for values 1 and 2.

64 For index spaces that are listed in the DSNACCOR recommendations result set, check whether the related table spaces are listed in the exception table. For recommendations result set rows that have corresponding exception table rows, copy the contents of the QUERYTYPE column of the exception table to the INEXCEPTTABLE column of the recommendations result set.

ChkLvl is an input parameter of type INTEGER. The default is 7 (values 1+2+4).

Criteria
Narrows the set of objects for which DSNACCOR makes recommendations. This value is the search condition of an SQL WHERE clause. *Criteria* is an input parameter of type VARCHAR(4096). The default is that DSNACCOR makes recommendations for all table spaces and index spaces in the subsystem. The search condition can use any column in the result set and wildcards are allowed.

Restricted
A parameter that is reserved for future use. Specify the null value for this parameter. *Restricted* is an input parameter of type VARCHAR(80).

CRUpdatedPagesPct
Specifies a criterion for recommending a full image copy on a table space or index space. If the following condition is true for a table space, DSNACCOR recommends an image copy:

The total number of distinct updated pages, divided by the total number of preformatted pages (expressed as a percentage) is greater than *CRUpdatedPagesPct.*

See item 2 in Figure 181 on page 1267. If both of the following conditions are true for an index space, DSNACCOR recommends an image copy:

- The total number of distinct updated pages, divided by the total number of preformatted pages (expressed as a percentage) is greater than *CRUpdatedPagesPct.*
- The number of active pages in the index space or partition is greater than *CRIndexSize*. See items 2 and 3 in Figure 182 on page 1268.

CRUpdatedPagesPct is an input parameter of type INTEGER. The default is **20**.

CRChangesPct
Specifies a criterion for recommending a full image copy on a table space or index space. If the following condition is true for a table space, DSNACCOR recommends an image copy:

The total number of insert, update, and delete operations since the last image copy, divided by the total number of rows or LOBs in a table space or partition (expressed as a percentage) is greater than *CRChangesPct.*

See item 3 in Figure 181 on page 1267. If both of the following conditions are true for an index table space, DSNACCOR recommends an image copy:

- The total number of insert and delete operations since the last image copy, divided by the total number of entries in the index space or partition (expressed as a percentage) is greater than *CRChangesPct*.
- The number of active pages in the index space or partition is greater than *CRIndexSize.*

See items 2 and 4 in Figure 182 on page 1268. *CRChangesPct* is an input parameter of type INTEGER. The default is **10**.

CRDaySncLastCopy
Specifies a criterion for recommending a full image copy on a table space or index space. If the number of days since the last image copy is greater than this value, DSNACCOR recommends an image copy. (See item 1 in Figure 181 on page 1267 and item 1 in Figure 182 on page 1268.) *CRDaySncLastCopy* is an input parameter of type INTEGER. The default is **7**.

ICRUpdatedPagesPct
Specifies a criterion for recommending an incremental image copy on a table space. If the following condition is true, DSNACCOR recommends an incremental image copy:

The number of distinct pages that were updated since the last image copy, divided by the total number of active pages in the table space or partition (expressed as a percentage) is greater than *CRUpdatedPagesPct.*

(See item 1 in Figure 183 on page 1268.) *ICRUpdatedPagesPct* is an input parameter of type INTEGER. The default is **1**.

ICRChangesPct
Specifies a criterion for recommending an incremental image copy on a table space. If the following condition is true, DSNACCOR recommends an incremental image copy:

The ratio of the number of insert, update, or delete operations since the last image copy, to the total number of rows or LOBs in a table space or partition (expressed as a percentage) is greater than *ICRChangesPct.*

(See item 2 in Figure 183 on page 1268.) *ICRChangesPct* is an input parameter of type INTEGER. The default is **1**.

CRIndexSize
Specifies, when combined with *CRUpdatedPagesPct* or *CRChangesPct*, a criterion

for recommending a full image copy on an index space. (See items 2, 3, and 4 in Figure 182 on page 1268.) *CRIndexSize* is an input parameter of type INTEGER. The default is **50**.

RRTInsDelUpdPct
Specifies a criterion for recommending that the REORG utility is to be run on a table space. If the following condition is true, DSNACCOR recommends running REORG:

The sum of insert, update, and delete operations since the last REORG, divided by the total number of rows or LOBs in the table space or partition (expressed as a percentage) is greater than *RRTInsDelUpdPct*

(See item 1 in Figure 184 on page 1268.) *RRTInsDelUpdPct* is an input parameter of type INTEGER. The default is **20**.

RRTUnclustInsPct
Specifies a criterion for recommending that the REORG utility is to be run on a table space. If the following condition is true, DSNACCOR recommends running REORG:

The number of unclustered insert operations, divided by the total number of rows or LOBs in the table space or partition (expressed as a percentage) is greater than *RRTUnclustInsPct.*

(See item 2 in Figure 184 on page 1268.) *RRTUnclustInsPct* is an input parameter of type INTEGER. The default is **10**.

RRTDisorgLOBPct
Specifies a criterion for recommending that the REORG utility is to be run on a table space. If the following condition is true, DSNACCOR recommends running REORG:

The number of imperfectly chunked LOBs, divided by the total number of rows or LOBs in the table space or partition (expressed as a percentage) is greater than *RRTDisorgLOBPct.*

(See item 3 in Figure 184 on page 1268.) *RRTDisorgLOBPct* is an input parameter of type INTEGER. The default is **10**.

RRTMassDelLimit
Specifies a criterion for recommending that the REORG utility is to be run on a table space. If one of the following values is greater than *RRTMassDelLimit*, DSNACCOR recommends running REORG:

- The number of mass deletes from a segmented or LOB table space since the last REORG or LOAD REPLACE
- The number of dropped tables from a nonsegmented table space since the last REORG or LOAD REPLACE

(See item 5 in Figure 184 on page 1268.) *RRTMassDelLimit* is an input parameter of type INTEGER. The default is **0**.

RRTIndRefLimit
Specifies a criterion for recommending that the REORG utility is to be run on a table space. If the following value is greater than *RRTIndRefLimit*, DSNACCOR recommends running REORG:

The total number of overflow records that were created since the last REORG or LOAD REPLACE, divided by the total number of rows or LOBs in the table space or partition (expressed as a percentage)

(See item 4 in Figure 184 on page 1268.) *RRTIndRefLimit* is an input parameter of type INTEGER. The default is **10**.

RRIInsertDeletePct
Specifies a criterion for recommending that the REORG utility is to be run on an index space. If the following value is greater than *RRIInsertDeletePct*, DSNACCOR recommends running REORG:

The sum of the number of index entries that were inserted and deleted since the last REORG, divided by the total number of index entries in the index space or partition (expressed as a percentage)

(See item 1 in Figure 185 on page 1269.) This is an input parameter of type INTEGER. The default is **20**.

RRIAppendInsertPct
Specifies a criterion for recommending that the REORG utility is to be run on an index space. If the following value is greater than *RRIAppendInsertPct*, DSNACCOR recommends running REORG:

The number of index entries that were inserted since the last REORG, REBUILD INDEX, or LOAD REPLACE with a key value greater than the maximum key value in the index space or partition, divided by the number of index entries in the index space or partition (expressed as a percentage)

(See item 2 in Figure 185 on page 1269.) *RRIInsertDeletePct* is an input parameter of type INTEGER. The default is **10**.

RRIPseudoDeletePct
Specifies a criterion for recommending that the REORG utility is to be run on an index space. If the following value is greater than *RRIPseudoDeletePct*, DSNACCOR recommends running REORG:

The number of index entries that were pseudo-deleted since the last REORG, REBUILD INDEX, or LOAD REPLACE, divided by the number of index entries in the index space or partition (expressed as a percentage)

(See item 3 in Figure 185 on page 1269.) *RRIPseudoDeletePct* is an input parameter of type INTEGER. The default is **10**.

RRIMassDelLimit
Specifies a criterion for recommending that the REORG utility is to be run on an index space. If the number of mass deletes from an index space or partition since the last REORG, REBUILD, or LOAD REPLACE is greater than this value, DSNACCOR recommends running REORG.

(See item 4 in Figure 185 on page 1269.) *RRIMassDelLimit* is an input parameter of type INTEGER. The default is **0**.

RRILeafLimit
Specifies a criterion for recommending that the REORG utility is to be run on an index space. If the following value is greater than *RRILeafLimit*, DSNACCOR recommends running REORG:

The number of index page splits that occurred since the last REORG, REBUILD INDEX, or LOAD REPLACE in which the higher part of the split page was far from the location of the original page, divided by the total number of active pages in the index space or partition (expressed as a percentage)

(See item 5 in Figure 185 on page 1269.) *RRILeafLimit* is an input parameter of type INTEGER. The default is **10**.

RRINumLevelsLimit
Specifies a criterion for recommending that the REORG utility is to be run on an index space. If the following value is greater than *RRINumLevelsLimit*, DSNACCOR recommends running REORG:

The number of levels in the index tree that were added or removed since the last REORG, REBUILD INDEX, or LOAD REPLACE

(See item 6 in Figure 185 on page 1269.) *RRINumLevelsLimit* is an input parameter of type INTEGER. The default is **0**.

SRTInsDelUpdPct
Specifies, when combined with *SRTInsDelUpdAbs*, a criterion for recommending that the RUNSTATS utility is to be run on a table space. If both of the following conditions are true, DSNACCOR recommends running RUNSTATS:

- The number of insert, update, or delete operations since the last RUNSTATS on a table space or partition, divided by the total number of rows or LOBs in table space or partition (expressed as a percentage) is greater than *SRTInsDelUpdPct.*
- The sum of the number of inserted and deleted index entries since the last RUNSTATS on an index space or partition is greater than *SRTInsDelUpdAbs.*

(See items 1 and 2 in Figure 186 on page 1269.) *SRTInsDelUpdPct* is an input parameter of type INTEGER. The default is **20**.

SRTInsDelUpdAbs
Specifies, when combined with *SRTInsDelUpdPct*, a criterion for recommending that the RUNSTATS utility is to be run on a table space. If both of the following conditions are true, DSNACCOR recommends running RUNSTATS:

- The number of insert, update, and delete operations since the last RUNSTATS on a table space or partition, divided by the total number of rows or LOBs in table space or partition (expressed as a percentage) is greater than *SRTInsDelUpdPct.*
- The sum of the number of inserted and deleted index entries since the last RUNSTATS on an index space or partition is greater than *SRTInsDelUpdAbs.*

(See items 1 and 2 in Figure 186 on page 1269.) *SRTInsDelUpdAbs* is an input parameter of type INTEGER. The default is **0**.

SRTMassDelLimit
Specifies a criterion for recommending that the RUNSTATS utility is to be run on a table space. If the following condition is true, DSNACCOR recommends running RUNSTATS:

- The number of mass deletes from a table space or partition since the last REORG or LOAD REPLACE is greater than *SRTMassDelLimit.*

(See item 3 in Figure 186 on page 1269.) *SRTMassDelLimit* is an input parameter of type INTEGER. The default is **0**.

SRIInsDelPct
Specifies, when combined with *SRIInsDelAbs*, a criterion for recommending that the RUNSTATS utility is to be run on an index space. If both of the following conditions are true, DSNACCOR recommends running RUNSTATS:

- The number of inserted and deleted index entries since the last RUNSTATS on an index space or partition, divided by the total number of index entries in the index space or partition (expressed as a percentage) is greater than *SRIInsDelUpdPct.*
- The sum of the number of inserted and deleted index entries since the last RUNSTATS on an index space or partition is greater than *SRIInsDelUpdAbs.*

(See items 1 and 2 in Figure 187 on page 1269.) *SRIInsDelPct* is an input parameter of type INTEGER. The default is **20**.

SRIInsDelAbs
Specifies, when combined with *SRIInsDelPct*, specifies a criterion for recommending that the RUNSTATS utility is to be run on an index space. If the following condition is true, DSNACCOR recommends running RUNSTATS:

- The number of inserted and deleted index entries since the last RUNSTATS on an index space or partition, divided by the total number of index entries in the index space or partition (expressed as a percentage) is greater than *SRIInsDelUpdPct.*
- The sum of the number of inserted and deleted index entries since the last RUNSTATS on an index space or partition is greater than *SRIInsDelUpdAbs*,

(See items 1 and 2 in Figure 187 on page 1269.) *SRIInsDelAbs* is an input parameter of type INTEGER. The default is **0**.

SRIMassDelLimit
Specifies a criterion for recommending that the RUNSTATS utility is to be run on an index space. If the number of mass deletes from an index space or partition since the last REORG, REBUILD INDEX, or LOAD REPLACE is greater than this value, DSNACCOR recommends running RUNSTATS.

(See item 3 in Figure 187 on page 1269.) *SRIMassDelLimit* is an input parameter of type INTEGER. The **default** is **0**.

ExtentLimit
Specifies a criterion for recommending that the RUNSTATS or REORG utility is to be run on a table space or index space. Also specifies that DSNACCOR is to warn the user that the table space or index space has used too many extents. DSNACCOR recommends running RUNSTATS or REORG, and altering data set allocations if the following condition is true:

- The number of physical extents in the index space, table space, or partition is greater than *ExtentLimit.*

(See Figure 188 on page 1269.) *ExtentLimit* is an input parameter of type INTEGER. The default is **50**.

LastStatement
When DSNACCOR returns a severe error (return code 12), this field contains the SQL statement that was executing when the error occurred. *LastStatement* is an output parameter of type VARCHAR(8012).

ReturnCode
The return code from DSNACCOR execution. Possible values are:

0 DSNACCOR executed successfully. The *ErrorMsg* parameter contains the approximate percentage of the total number of objects in the subsystem that have information in the real-time statistics tables.

4 DSNACCOR completed, but one or more input parameters might be incompatible. The *ErrorMsg* parameter contains the input parameters that might be incompatible.

8 DSNACCOR terminated with errors. The *ErrorMsg* parameter contains a message that describes the error.

12 DSNACCOR terminated with severe errors. The *ErrorMsg* parameter contains a message that describes the error. The *LastStatement* parameter contains the SQL statement that was executing when the error occurred.

14 DSNACCOR terminated because it could not access one or more of the

real-time statistics tables. The *ErrorMsg* parameter contains the names of the tables that DSNACCOR could not access.

15 DSNACCOR terminated because it encountered a problem with one of the declared temporary tables that it defines and uses.

16 DSNACCOR terminated because it could not define a declared temporary table. No table spaces were defined in the TEMP database.

NULL DSNACCOR terminated but could not set a return code.

ReturnCode is an output parameter of type INTEGER.

ErrorMsg
Contains information about DSNACCOR execution. If DSNACCOR runs successfully (*ReturnCode*=0), this field contains the approximate percentage of objects in the subsystem that are in the real-time statistics tables. Otherwise, this field contains error messages. *ErrorMsg* is an output parameter of type VARCHAR(1331).

IFCARetCode
Contains the return code from an IFI COMMAND call. DSNACCOR issues commands through the IFI interface to determine the status of objects. *IFCARetCode* is an output parameter of type INTEGER.

IFCAResCode
Contains the reason code from an IFI COMMAND call. *IFCAResCode* is an output parameter of type INTEGER.

ExcessBytes
Contains the number of bytes of information that did not fit in the IFI return area after an IFI COMMAND call. *ExcessBytes* is an output parameter of type INTEGER.

DSNACCOR formulas for recommending actions

The following formulas specify the criteria that DSNACCOR uses for its recommendations and warnings. The variables in italics are DSNACCOR input parameters. The capitalized variables are columns of the SYSIBM.TABLESPACESTATS or SYSIBM.INDEXSPACESTATS tables. The numbers to the right of selected items are reference numbers for the option descriptions in "DSNACCOR option descriptions" on page 1259.

Figure 181 shows the formula that DSNACCOR uses to recommend a full image copy on a table space.

```
((QueryType='COPY' OR QueryType='ALL') AND
 (ObjectType='TS' OR ObjectType='ALL') AND
 ICType='F') AND
 (COPYLASTTIME IS NULL OR
 REORGLASTTIME>COPYLASTTIME OR
 LOADRLASTTIME>COPYLASTTIME OR
 (CURRENT DATE-COPYLASTTIME)>CRDaySncLastCopy OR      1
 (COPYUPDATEDPAGES*100)/NACTIVE>CRUpdatedPagesPct OR  2
 (COPYCHANGES*100)/TOTALROWS>CRChangesPct)            3
```

Figure 181. DSNACCOR formula for recommending a full image copy on a table space

Figure 182 shows the formula that DSNACCOR uses to recommend a full image copy on an index space.

```
((QueryType='COPY' OR QueryType='ALL') AND
 (ObjectType='IX' OR ObjectType='ALL') AND
 (ICType='F'  OR ICType='B')) AND
 (COPYLASTTIME IS NULL OR
 REORGLASTTIME>COPYLASTTIME OR
 LOADRLASTTIME>COPYLASTTIME OR
 REBUILDLASTTIME>COPYLASTTIME OR
 (CURRENT DATE-COPYLASTTIME)>CRDaySncLastCopy OR        1
 (NACTIVE>CRIndexSize AND                               2
 ((COPYUPDATEDPAGES*100)/NACTIVE>CRUpdatedPagesPct OR   3
 (COPYCHANGES*100)/TOTALENTRIES>CRChangesPct)))         4
```

Figure 182. DSNACCOR formula for recommending a full image copy on an index space

Figure 183 shows the formula that DSNACCOR uses to recommend an incremental image copy on a table space.

```
((QueryType='COPY' OR QueryType='ALL') AND
 (ObjectType='TS' OR ObjectType='ALL') AND
 ICType='I' AND
 COPYLASTTIME IS NOT NULL) AND
 (LOADRLASTTIME>COPYLASTTIME OR
 REORGLASTTIME>COPYLASTTIME OR
 (COPYUPDATEDPAGES*100)/NACTIVE>ICRUpdatedPagesPct OR   1
 (COPYCHANGES*100)/TOTALROWS>ICRChangesPct))            2
```

Figure 183. DSNACCOR formula for recommending an incremental image copy on a table space

Figure 184 shows the formula that DSNACCOR uses to recommend a REORG on a table space. If the table space is a LOB table space, and CHCKLVL=1, the formula does not include EXTENTS>*ExtentLimit*.

```
((QueryType='REORG' OR QueryType='ALL') AND
 (ObjectType='TS' OR ObjectType='ALL')) AND
 (REORGLASTTIME IS NULL OR
 ((REORGINSERTS+REORGDELETES+REORGUPDATES)*100)/TOTALROWS>RRTInsDelUpdPct OR   1
 (REORGUNCLUSTINS*100)/TOTALROWS>RRTUnclustInsPct OR                           2
 (REORGDISORGLOB*100)/TOTALROWS>RRTDisorgLOBPct OR                             3
 ((REORGNEARINDREF+REORGFARINDREF)*100)/TOTALROWS>RRTIndRefLimit OR            4
 REORGMASSDELETE>RRTMassDelLimit OR                                            5
 EXTENTS>ExtentLimit)                                                          6
```

Figure 184. DSNACCOR formula for recommending a REORG on a table space

Figure 185 on page 1269 shows the formula that DSNACCOR uses to recommend a REORG on an index space.

```
((QueryType='REORG' OR QueryType='ALL') AND
 (ObjectType='IX' OR ObjectType='ALL')) AND
 (REORGLASTTIME IS NULL OR
 ((REORGINSERTS+REORGDELETES)*100)/TOTALENTRIES>RRIInsertDeletePct OR        1
 (REORGAPPENDINSERT*100)/TOTALENTRIES>RRIAppendInsertPct OR                  2
 (REORGPSEUDODELETES*100)/TOTALENTRIES>RRIPseudoDeletePct OR                 3
 REORGMASSDELETE>RRIMassDeleteLimit OR                                       4
 (REORGLEAFFAR*100)/NACTIVE>RRILeafLimit OR                                  5
 REORGNUMLEVELS>RRINumLevelsLimit OR                                         6
 EXTENTS>ExtentLimit)                                                        7
```

Figure 185. DSNACCOR formula for recommending a REORG on an index space

Figure 186 shows the formula that DSNACCOR uses to recommend RUNSTATS on a table space.

```
((QueryType='RUNSTATS' OR QueryType='ALL') AND
 (ObjectType='TS' OR ObjectType='ALL')) AND
 (STATSLASTTIME IS NULL OR
 (((STATSINSERTS+STATSDELETES+STATSUPDATES)*100)/TOTALROWS>SRTInsDelUpdPct AND  1
 (STATSINSERTS+STATSDELETES+STATSUPDATES)>SRTInsDelUpdAbs) OR                   2
 STATSMASSDELETE>SRTMassDeleteLimit)                                            3
```

Figure 186. DSNACCOR formula for recommending RUNSTATS on a table space

Figure 187 shows the formula that DSNACCOR uses to recommend RUNSTATS on an index space.

```
((QueryType='RUNSTATS' OR QueryType='ALL') AND
 (ObjectType='IX' OR ObjectType='ALL')) AND
 (STATSLASTTIME IS NULL OR
 (((STATSINSERTS+STATSDELETES)*100)/TOTALENTRIES>SRIInsDelUpdPct AND   1
 (STATSINSERTS+STATSDELETES)>SRIInsDelPct) OR                          2
 STATSMASSDELETE>SRIInsDelAbs)                                         3
```

Figure 187. DSNACCOR formula for recommending RUNSTATS on an index space

Figure 188 shows the formula that DSNACCOR uses to that too many index space or table space extents have been used.

```
EXTENTS>ExtentLimit
```

Figure 188. DSNACCOR formula for warning that too many data set extents for a table space or index space are used

Using an exception table

An exception table is an optional, user-created DB2 table that you can use to place information in the INEXCEPTTABLE column of the recommendations result set. You can put any information in the INEXCEPTTABLE column, but the most common use of this column is to filter the recommendations result set. Each row in the exception table represents an object for which you want to provide information for the recommendations result set.

To create the exception table, execute a CREATE TABLE statement similar to the following one. You can include other columns in the exception table, but you must include at least the columns that are shown.

```
CREATE TABLE DSNACC.EXCEPT_TBL
 (DBNAME CHAR(8) NOT NULL,
  NAME CHAR(8) NOT NULL,
  QUERYTYPE CHAR(40))
 CCSID EBCDIC;
```

The meanings of the columns are:

DBNAME
The database name for an object in the exception table.

NAME
The table space name or index space name for an object in the exception table.

QUERYTYPE
The information that you want to place in the INEXCEPTTABLE column of the recommendations result set.

If you put a null value in this column, DSNACCOR puts the value YES in the INEXCEPTTABLE column of the recommendations result set row for the object that matches the DBNAME and NAME values.

Recommendation: If you plan to put many rows in the exception table, create a nonunique index on DBNAME, NAME, and QUERYTYPE.

After you create the exception table, insert a row for each object for which you want to include information in the INEXCEPTTABLE column.

Example: Suppose that you want the INEXCEPTTABLE column to contain the string 'IRRELEVANT' for table space STAFF in database DSNDB04. You also want the INEXCEPTTABLE column to contain 'CURRENT' for table space DSN8S81D in database DSN8D81A. Execute these INSERT statements:

```
INSERT INTO DSNACC.EXCEPT_TBL VALUES('DSNDB04 ', 'STAFF ', 'IRRELEVANT');
INSERT INTO DSNACC.EXCEPT_TBL VALUES('DSN8D81A', 'DSN8S81D', 'CURRENT');
```

To use the contents of INEXCEPTTABLE for filtering, include a condition that involves the INEXCEPTTABLE column in the search condition that you specify in your *Criteria* input parameter.

Example: Suppose that you want to include all rows for database DSNDB04 in the recommendations result set, except for those rows that contain the string 'IRRELEVANT' in the INEXCEPTTABLE column. You might include the following search condition in your *Criteria* input parameter:

```
DBNAME='DSNDB04' AND INEXCEPTTABLE<>'IRRELEVANT'
```

Example of DSNACCOR invocation

Figure 189 on page 1271 is a COBOL example that shows variable declarations and an SQL CALL for obtaining recommendations for objects in databases DSN8D81A and DSN8D81L. This example also outlines the steps that you need to perform to retrieve the two result sets that DSNACCOR returns. These result sets are described in "DSNACCOR output" on page 1274. See *DB2 Application Programming and SQL Guide* for more information about how to retrieve result sets from a stored procedure.

```
 WORKING-STORAGE SECTION.
:
:
**********************
* DSNACCOR PARAMETERS *
**********************
 01  QUERYTYPE.
     49   QUERYTYPE-LN        PICTURE S9(4) COMP VALUE 40.
     49   QUERYTYPE-DTA       PICTURE X(40)  VALUE 'ALL'.
 01  OBJECTTYPE.
     49   OBJECTTYPE-LN       PICTURE S9(4) COMP VALUE 3.
     49   OBJECTTYPE-DTA      PICTURE X(3)  VALUE 'ALL'.
 01  ICTYPE.
     49   ICTYPE-LN           PICTURE S9(4) COMP VALUE 1.
     49   ICTYPE-DTA          PICTURE X(1)  VALUE 'B'.
 01  STATSSCHEMA.
     49   STATSSCHEMA-LN      PICTURE S9(4) COMP VALUE 128.
     49   STATSSCHEMA-DTA     PICTURE X(128)  VALUE 'SYSIBM'.
 01  CATLGSCHEMA.
 49   CATLGSCHEMA-LN      PICTURE S9(4) COMP VALUE 128.
     49   CATLGSCHEMA-DTA     PICTURE X(128)  VALUE 'SYSIBM'.
 01  LOCALSCHEMA.
     49   LOCALSCHEMA-LN      PICTURE S9(4) COMP VALUE 128.
     49   LOCALSCHEMA-DTA     PICTURE X(128)  VALUE 'DSNACC'.
 01  CHKLVL                  PICTURE S9(9) COMP VALUE +3.
 01  CRITERIA.
     49   CRITERIA-LN         PICTURE S9(4) COMP VALUE 4096.
     49   CRITERIA-DTA        PICTURE X(4096)  VALUE SPACES.
 01  RESTRICTED.
     49   RESTRICTED-LN       PICTURE S9(4) COMP VALUE 80.
     49   RESTRICTED-DTA      PICTURE X(80)  VALUE SPACES.
 01  CRUPDATEDPAGESPCT       PICTURE S9(9) COMP VALUE +0.
 01  CRCHANGESPCT            PICTURE S9(9) COMP VALUE +0.
 01  CRDAYSNCLASTCOPY        PICTURE S9(9) COMP VALUE +0.
 01  ICRUPDATEDPAGESPCT      PICTURE S9(9) COMP VALUE +0.
 01  ICRCHANGESPCT           PICTURE S9(9) COMP VALUE +0.
 01  CRINDEXSIZE             PICTURE S9(9) COMP VALUE +0.
 01  RRTINSDELUPDPCT         PICTURE S9(9) COMP VALUE +0.
 01  RRTUNCLUSTINSPCT        PICTURE S9(9) COMP VALUE +0.
 01  RRTDISORGLOBPCT         PICTURE S9(9) COMP VALUE +0.
 01  RRTMASSDELLIMIT         PICTURE S9(9) COMP VALUE +0.
 01  RRTINDREFLIMIT          PICTURE S9(9) COMP VALUE +0.
 01  RRIINSERTDELETEPCT      PICTURE S9(9) COMP VALUE +0.
 01  RRIAPPENDINSERTPCT      PICTURE S9(9) COMP VALUE +0.
 01  RRIPSEUDODELETEPCT      PICTURE S9(9) COMP VALUE +0.
 01  RRIMASSDELLIMIT         PICTURE S9(9) COMP VALUE +0.
 01  RRILEAFLIMIT            PICTURE S9(9) COMP VALUE +0.
 01  RRINUMLEVELSLIMIT       PICTURE S9(9) COMP VALUE +0.
 01  SRTINSDELUPDPCT         PICTURE S9(9) COMP VALUE +0.
 01  SRTINSDELUPDABS         PICTURE S9(9) COMP VALUE +0.
 01  SRTMASSDELLIMIT         PICTURE S9(9) COMP VALUE +0.
 01  SRIINSDELUPDPCT         PICTURE S9(9) COMP VALUE +0.
 01  SRIINSDELUPDABS         PICTURE S9(9) COMP VALUE +0.
 01  SRIMASSDELLIMIT         PICTURE S9(9) COMP VALUE +0.
 01  EXTENTLIMIT             PICTURE S9(9) COMP VALUE +0.
 01  LASTSTATEMENT.
     49   LASTSTATEMENT-LN    PICTURE S9(4) COMP VALUE 8012.
     49   LASTSTATEMENT-DTA   PICTURE X(8012)  VALUE SPACES.
 01  RETURNCODE              PICTURE S9(9) COMP VALUE +0.
 01  ERRORMSG.
     49   ERRORMSG-LN         PICTURE S9(4) COMP VALUE 1331.
     49   ERRORMSG-DTA        PICTURE X(1331)  VALUE SPACES.
 01  IFCARETCODE             PICTURE S9(9) COMP VALUE +0.
 01  IFCARESCODE             PICTURE S9(9) COMP VALUE +0.
 01  EXCESSBYTES             PICTURE S9(9) COMP VALUE +0.
```

Figure 189. Example of DSNACCOR invocation (Part 1 of 4)

```
****************************************
* INDICATOR VARIABLES.                 *
* INITIALIZE ALL NON-ESSENTIAL INPUT   *
* VARIABLES TO -1, TO INDICATE THAT THE *
* INPUT VALUE IS NULL.                 *
****************************************
 01  QUERYTYPE-IND            PICTURE S9(4) COMP-4 VALUE +0.
 01  OBJECTTYPE-IND           PICTURE S9(4) COMP-4 VALUE +0.
 01  ICTYPE-IND               PICTURE S9(4) COMP-4 VALUE +0.
 01  STATSSCHEMA-IND          PICTURE S9(4) COMP-4 VALUE -1.
 01  CATLGSCHEMA-IND          PICTURE S9(4) COMP-4 VALUE -1.
 01  LOCALSCHEMA-IND          PICTURE S9(4) COMP-4 VALUE -1.
 01  CHKLVL-IND               PICTURE S9(4) COMP-4 VALUE -1.
 01  CRITERIA-IND             PICTURE S9(4) COMP-4 VALUE -1.
 01  RESTRICTED-IND           PICTURE S9(4) COMP-4 VALUE -1.
 01  CRUPDATEDPAGESPCT-IND    PICTURE S9(4) COMP-4 VALUE -1.
 01  CRCHANGESPCT-IND         PICTURE S9(4) COMP-4 VALUE -1.
 01  CRDAYSNCLASTCOPY-IND     PICTURE S9(4) COMP-4 VALUE -1.
 01  ICRUPDATEDPAGESPCT-IND   PICTURE S9(4) COMP-4 VALUE -1.
 01  ICRCHANGESPCT-IND        PICTURE S9(4) COMP-4 VALUE -1.
 01  CRINDEXSIZE-IND          PICTURE S9(4) COMP-4 VALUE -1.
 01  RRTINSDELUPDPCT-IND      PICTURE S9(4) COMP-4 VALUE -1.
 01  RRTUNCLUSTINSPCT-IND     PICTURE S9(4) COMP-4 VALUE -1.
 01  RRTDISORGLOBPCT-IND      PICTURE S9(4) COMP-4 VALUE -1.
 01  RRTMASSDELLIMIT-IND      PICTURE S9(4) COMP-4 VALUE -1.
 01  RRTINDREFLIMIT-IND       PICTURE S9(4) COMP-4 VALUE -1.
 01  RRIINSERTDELETEPCT-IND   PICTURE S9(4) COMP-4 VALUE -1.
 01  RRIAPPENDINSERTPCT-IND   PICTURE S9(4) COMP-4 VALUE -1.
 01  RRIPSEUDODELETEPCT-IND   PICTURE S9(4) COMP-4 VALUE -1.
 01  RRIMASSDELLIMIT-IND      PICTURE S9(4) COMP-4 VALUE -1.
 01  RRILEAFLIMIT-IND         PICTURE S9(4) COMP-4 VALUE -1.
 01  RRINUMLEVELSLIMIT-IND    PICTURE S9(4) COMP-4 VALUE -1.
 01  SRTINSDELUPDPCT-IND      PICTURE S9(4) COMP-4 VALUE -1.
 01  SRTINSDELUPDABS-IND      PICTURE S9(4) COMP-4 VALUE -1.
 01  SRTMASSDELLIMIT-IND      PICTURE S9(4) COMP-4 VALUE -1.
 01  SRIINSDELUPDPCT-IND      PICTURE S9(4) COMP-4 VALUE -1.
 01  SRIINSDELUPDABS-IND      PICTURE S9(4) COMP-4 VALUE -1.
 01  SRIMASSDELLIMIT-IND      PICTURE S9(4) COMP-4 VALUE -1.
 01  EXTENTLIMIT-IND          PICTURE S9(4) COMP-4 VALUE -1.
 01  LASTSTATEMENT-IND        PICTURE S9(4) COMP-4 VALUE +0.
 01  RETURNCODE-IND           PICTURE S9(4) COMP-4 VALUE +0.
 01  ERRORMSG-IND             PICTURE S9(4) COMP-4 VALUE +0.
 01  IFCARETCODE-IND          PICTURE S9(4) COMP-4 VALUE +0.
 01  IFCARESCODE-IND          PICTURE S9(4) COMP-4 VALUE +0.
 01  EXCESSBYTES-IND          PICTURE S9(4) COMP-4 VALUE +0.

 PROCEDURE DIVISION.
:
*******************************************************
* SET VALUES FOR DSNACCOR INPUT PARAMETERS:           *
* - USE THE CHKLVL PARAMETER TO CAUSE DSNACCOR TO CHECK *
*   FOR ORPHANED OBJECTS AND INDEX SPACES WITHOUT     *
*   TABLE SPACES, BUT INCLUDE THOSE OBJECTS IN THE    *
*   RECOMMENDATIONS RESULT SET (CHKLVL=1+2+16=19)     *
* - USE THE CRITERIA PARAMETER TO CAUSE DSNACCOR TO   *
*   MAKE RECOMMENDATIONS ONLY FOR OBJECTS IN DATABASES *
*   DSN8D81A AND DSN8D81L.                            *
```

Figure 189. Example of DSNACCOR invocation (Part 2 of 4)

```
* - FOR THE FOLLOWING PARAMETERS, SET THESE VALUES,     *
*   WHICH ARE LOWER THAN THE DEFAULTS:                  *
*   CRUPDATEDPAGESPCT   4                               *
*   CRCHANGESPCT        2                               *
*   RRTINSDELUPDPCT     2                               *
*   RRTUNCLUSTINSPCT    5                               *
*   RRTDISORGLOBPCT     5                               *
*   RRIAPPENDINSERTPCT  5                               *
*   SRTINSDELUPDPCT     5                               *
*   SRIINSDELUPDPCT     5                               *
*   EXTENTLIMIT         3                               *
*******************************************************
     MOVE 19 TO CHKLVL.
     MOVE SPACES TO CRITERIA-DTA.
     MOVE 'DBNAME = ''DSN8D81A'' OR DBNAME = ''DSN8D81L'''
           TO CRITERIA-DTA.
     MOVE 46 TO CRITERIA-LN.
     MOVE 4 TO CRUPDATEDPAGESPCT.
     MOVE 2 TO CRCHANGESPCT.
     MOVE 2 TO RRTINSDELUPDPCT.
     MOVE 5 TO RRTUNCLUSTINSPCT.
     MOVE 5 TO RRTDISORGLOBPCT.
     MOVE 5 TO RRIAPPENDINSERTPCT.
     MOVE 5 TO SRTINSDELUPDPCT.
     MOVE 5 TO SRIINSDELUPDPCT.
     MOVE 3 TO EXTENTLIMIT.
********************************
* INITIALIZE OUTPUT PARAMETERS *
********************************
     MOVE SPACES TO LASTSTATEMENT-DTA.
     MOVE 1 TO LASTSTATEMENT-LN.
     MOVE 0 TO RETURNCODE-O2.
     MOVE SPACES TO ERRORMSG-DTA.
     MOVE 1 TO ERRORMSG-LN.
     MOVE 0 TO IFCARETCODE.
     MOVE 0 TO IFCARESCODE.
     MOVE 0 TO EXCESSBYTES.
*******************************************************
* SET THE INDICATOR VARIABLES TO 0 FOR NON-NULL INPUT *
* PARAMETERS (PARAMETERS FOR WHICH YOU DO NOT WANT    *
* DSNACCOR TO USE DEFAULT VALUES) AND FOR OUTPUT      *
* PARAMETERS.                                         *
*******************************************************
     MOVE 0 TO CHKLVL-IND.
     MOVE 0 TO CRITERIA-IND.
     MOVE 0 TO CRUPDATEDPAGESPCT-IND.
     MOVE 0 TO CRCHANGESPCT-IND.
     MOVE 0 TO RRTINSDELUPDPCT-IND.
     MOVE 0 TO RRTUNCLUSTINSPCT-IND.
     MOVE 0 TO RRTDISORGLOBPCT-IND.
     MOVE 0 TO RRIAPPENDINSERTPCT-IND.
     MOVE 0 TO SRTINSDELUPDPCT-IND.
     MOVE 0 TO SRIINSDELUPDPCT-IND.
     MOVE 0 TO EXTENTLIMIT-IND.
     MOVE 0 TO LASTSTATEMENT-IND.
     MOVE 0 TO RETURNCODE-IND.
     MOVE 0 TO ERRORMSG-IND.
     MOVE 0 TO IFCARETCODE-IND.
     MOVE 0 TO IFCARESCODE-IND.
     MOVE 0 TO EXCESSBYTES-IND.
⋮
```

Figure 189. Example of DSNACCOR invocation (Part 3 of 4)

```
*****************
* CALL DSNACCOR *
*****************
     EXEC SQL
      CALL SYSPROC.DSNACCOR
      (:QUERYTYPE           :QUERYTYPE-IND,
       :OBJECTTYPE          :OBJECTTYPE-IND,
       :ICTYPE              :ICTYPE-IND,
       :STATSSCHEMA         :STATSSCHEMA-IND,
       :CATLGSCHEMA         :CATLGSCHEMA-IND,
       :LOCALSCHEMA         :LOCALSCHEMA-IND,
       :CHKLVL              :CHKLVL-IND,
       :CRITERIA            :CRITERIA-IND,
       :RESTRICTED          :RESTRICTED-IND,
       :CRUPDATEDPAGESPCT   :CRUPDATEDPAGESPCT-IND,
       :CRCHANGESPCT        :CRCHANGESPCT-IND,
       :CRDAYSNCLASTCOPY    :CRDAYSNCLASTCOPY-IND,
       :ICRUPDATEDPAGESPCT  :ICRUPDATEDPAGESPCT-IND,
       :ICRCHANGESPCT       :ICRCHANGESPCT-IND,
       :CRINDEXSIZE         :CRINDEXSIZE-IND,
       :RRTINSDELUPDPCT     :RRTINSDELUPDPCT-IND,
       :RRTUNCLUSTINSPCT    :RRTUNCLUSTINSPCT-IND,
       :RRTDISORGLOBPCT     :RRTDISORGLOBPCT-IND,
       :RRTMASSDELLIMIT     :RRTMASSDELLIMIT-IND,
       :RRTINDREFLIMIT      :RRTINDREFLIMIT-IND,
       :RRIINSERTDELETEPCT  :RRIINSERTDELETEPCT-IND,
       :RRIAPPENDINSERTPCT  :RRIAPPENDINSERTPCT-IND,
       :RRIPSEUDODELETEPCT  :RRIPSEUDODELETEPCT-IND,
       :RRIMASSDELLIMIT     :RRIMASSDELLIMIT-IND,
       :RRILEAFLIMIT        :RRILEAFLIMIT-IND,
       :RRINUMLEVELSLIMIT   :RRINUMLEVELSLIMIT-IND,
       :SRTINSDELUPDPCT     :SRTINSDELUPDPCT-IND,
       :SRTINSDELUPDABS     :SRTINSDELUPDABS-IND,
       :SRTMASSDELLIMIT     :SRTMASSDELLIMIT-IND,
       :SRIINSDELUPDPCT     :SRIINSDELUPDPCT-IND,
       :SRIINSDELUPDABS     :SRIINSDELUPDABS-IND,
       :SRIMASSDELLIMIT     :SRIMASSDELLIMIT-IND,
       :EXTENTLIMIT         :EXTENTLIMIT-IND,
       :LASTSTATEMENT       :LASTSTATEMENT-IND,
       :RETURNCODE          :RETURNCODE-IND,
       :ERRORMSG            :ERRORMSG-IND,
       :IFCARETCODE         :IFCARETCODE-IND,
       :IFCARESCODE         :IFCARESCODE-IND,
       :EXCESSBYTES         :EXCESSBYTES-IND)
     END-EXEC.
*************************************************************
* ASSUME THAT THE SQL CALL RETURNED +466, WHICH MEANS THAT  *
* RESULT SETS WERE RETURNED. RETRIEVE RESULT SETS.          *
*************************************************************
* LINK EACH RESULT SET TO A LOCATOR VARIABLE
     EXEC SQL ASSOCIATE LOCATORS (:LOC1, :LOC2)
       WITH PROCEDURE SYSPROC.DSNACCOR
     END-EXEC.
* LINK A CURSOR TO EACH RESULT SET
     EXEC SQL ALLOCATE C1 CURSOR FOR RESULT SET :LOC1
     END-EXEC.
     EXEC SQL ALLOCATE C2 CURSOR FOR RESULT SET :LOC2
     END-EXEC.
* PERFORM FETCHES USING C1 TO RETRIEVE ALL ROWS FROM FIRST RESULT SET
* PERFORM FETCHES USING C2 TO RETRIEVE ALL ROWS FROM SECOND RESULT SET
```

Figure 189. Example of DSNACCOR invocation (Part 4 of 4)

DSNACCOR output

If DSNACCOR executes successfully, in addition to the output parameters described in "DSNACCOR option descriptions" on page 1259, DSNACCOR returns two result sets.

The first result set contains the results from IFI COMMAND calls that DSNACCOR makes. Table 337 on page 1275 shows the format of the first result set.

Table 337. Result set row for first DSNACCOR result set

Column name	Data type	Contents
RS_SEQUENCE	INTEGER	Sequence number of the output line
RS_DATA	CHAR(80)	A line of command output

The second result set contains DSNACCOR's recommendations. This result set contains one or more rows for a table space or index space. A nonpartitioned table space or nonpartitioning index space can have at most one row in the result set. A partitioned table space or partitioning index space can have at most one row for each partition. A table space, index space, or partition has a row in the result set if both of the following conditions are true:

- If the *Criteria* input parameter contains a search condition, the search condition is true for the table space, index space, or partition.
- DSNACCOR recommends at least one action for the table space, index space, or partition.

Table 338 shows the columns of a result set row.

Table 338. Result set row for second DSNACCOR result set

Column name	Data type	Description
DBNAME	CHAR(8)	Name of the database that contains the object.
NAME	CHAR(8)	Table space or index space name.
PARTITION	INTEGER	Data set number or partition number.
OBJECTTYPE	CHAR(2)	DB2 object type: • TS for a table space • IX for an index space
OBJECTSTATUS	CHAR(36)	Status of the object: • ORPHANED, if the object is an index space with no corresponding table space, or if the object does not exist • If the object is in a restricted state, one of the following values: – TS=*restricted-state*, if OBJECTTYPE is TS – IX=*restricted-state*, if OBJECTTYPE is IX *restricted-state* is one of the status codes that appear in DISPLAY DATABASE output. See Chapter 2 of *DB2 Command Reference* for details. • A, if the object is in an advisory state. • L, if the object is a logical partition, but not in an advisory state. • AL, if the object is a logical partition and in an advisory state.
IMAGECOPY	CHAR(3)	COPY recommendation: • If OBJECTTYPE is TS: FUL (full image copy), INC (incremental image copy), or NO • If OBJECTTYPE is IX: YES or NO
RUNSTATS	CHAR(3)	RUNSTATS recommendation: YES or NO.
EXTENTS	CHAR(3)	Indicates whether the data sets for the object have exceeded *ExtentLimit*: YES or NO.
REORG	CHAR(3)	REORG recommendation: YES or NO.

Table 338. Result set row for second DSNACCOR result set (continued)

Column name	Data type	Description
INEXCEPTTABLE	CHAR(40)	A string that contains one of the following values: • Text that you specify in the QUERYTYPE column of the exception table. • YES, if you put a row in the exception table for the object that this result set row represents, but you specify NULL in the QUERYTYPE column. • NO, if the exception table exists but does not have a row for the object that this result set row represents. • Null, if the exception table does not exist, or if the *ChkLvl* input parameter does not include the value 4.
ASSOCIATEDTS	CHAR(8)	If OBJECTTYPE is IX and the *ChkLvl* input parameter includes the value 2, this value is the name of the table space that is associated with the index space. Otherwise null.
COPYLASTTIME	TIMESTAMP	Timestamp of the last full image copy on the object. Null if COPY was never run, or if the last COPY execution was terminated.
LOADRLASTTIME	TIMESTAMP	Timestamp of the last LOAD REPLACE on the object. Null if LOAD REPLACE was never run, or if the last LOAD REPLACE execution was terminated.
REBUILDLASTTIME	TIMESTAMP	Timestamp of the last REBUILD INDEX on the object. Null if REBUILD INDEX was never run, or if the last REBUILD INDEX execution was terminated.
CRUPDPGSPCT	INTEGER	If OBJECTTYPE is TS or IX and IMAGECOPY is YES, the ratio of distinct updated pages to preformatted pages, expressed as a percentage. Otherwise null.
CRCPYCHGPCT	INTEGER	If OBJECTTYPE is TS and IMAGECOPY is YES, the ratio of the total number insert, update, and delete operations since the last image copy to the total number of rows or LOBs in the table space or partition, expressed as a percentage. If OBJECTTYPE is IX and IMAGECOPY is YES, the ratio of the total number of insert and delete operations since the last image copy to the total number of entries in the index space or partition, expressed as a percentage. Otherwise null.
CRDAYSCELSTCPY	INTEGER	If OBJECTTYPE is TS or IX and IMAGECOPY is YES, the number of days since the last image copy. Otherwise null.
CRINDEXSIZE	INTEGER	If OBJECTTYPE is IX and IMAGECOPY is YES, the number of active pages in the index space or partition. Otherwise null.
REORGLASTTIME	TIMESTAMP	Timestamp of the last REORG on the object. Null if REORG was never run, or if the last REORG execution was terminated.
RRTINSDELUPDPCT	INTEGER	If OBJECTTYPE is TS and REORG is YES, the ratio of the sum of insert, update, and delete operations since the last REORG to the total number of rows or LOBs in the table space or partition, expressed as a percentage. Otherwise null.
RRTUNCINSPCT	INTEGER	If OBJECTTYPE is TS and REORG is YES, the ratio of the number of unclustered insert operations to the total number of rows or LOBs in the table space or partition, expressed as a percentage. Otherwise null.
RRTDISORGLOBPCT	INTEGER	If OBJECTTYPE is TS and REORG is YES, the ratio of the number of imperfectly chunked LOBs to the total number of rows or LOBs in the table space or partition, expressed as a percentage. Otherwise null.

Table 338. Result set row for second DSNACCOR result set (continued)

Column name	Data type	Description
RRTMASSDELETE	INTEGER	If OBJECTTYPE is TS, REORG is YES, and the table space is a segmented table space or LOB table space, the number of mass deletes since the last REORG or LOAD REPLACE. If OBJECTTYPE is TS, REORG is YES, and the table space is nonsegmented, the number of dropped tables since the last REORG or LOAD REPLACE. Otherwise null.
RRTINDREF	INTEGER	If OBJECTTYPE is TS, REORG is YES, the ratio of the total number of overflow records that were created since the last REORG or LOAD REPLACE to the total number of rows or LOBs in the table space or partition, expressed as a percentage. Otherwise null.
RRIINSDELPCT	INTEGER	If OBJECTTYPE is IX and REORG is YES, the ratio of the total number of insert and delete operations since the last REORG to the total number of index entries in the index space or partition, expressed as a percentage. Otherwise null.
RRIAPPINSPCT	INTEGER	If OBJECTTYPE is IX and REORG is YES, the ratio of the number of index entries that were inserted since the last REORG, REBUILD INDEX, or LOAD REPLACE that had a key value greater than the maximum key value in the index space or partition, to the number of index entries in the index space or partition, expressed as a percentage. Otherwise null.
RRIPSDDELPCT	INTEGER	If OBJECTTYPE is IX and REORG is YES, the ratio of the number of index entries that were pseudo-deleted (the RID entry was marked as deleted) since the last REORG, REBUILD INDEX, or LOAD REPLACE to the number of index entries in the index space or partition, expressed as a percentage. Otherwise null.
RRIMASSDELETE	INTEGER	If OBJECTTYPE is IX and REORG is YES, the number of mass deletes from the index space or partition since the last REORG, REBUILD, or LOAD REPLACE. Otherwise null.
RRILEAF	INTEGER	If OBJECTTYPE is IX and REORG is YES, the ratio of the number of index page splits that occurred since the last REORG, REBUILD INDEX, or LOAD REPLACE in which the higher part of the split page was far from the location of the original page, to the total number of active pages in the index space or partition, expressed as a percentage. Otherwise null.
RRINUMLEVELS	INTEGER	If OBJECTTYPE is IX and REORG is YES, the number of levels in the index tree that were added or removed since the last REORG, REBUILD INDEX, or LOAD REPLACE. Otherwise null.
STATSLASTTIME	TIMESTAMP	Timestamp of the last RUNSTATS on the object. Null if RUNSTATS was never run, or if the last RUNSTATS execution was terminated.
SRTINSDELUPDPCT	INTEGER	If OBJECTTYPE is TS and RUNSTATS is YES, the ratio of the total number of insert, update, and delete operations since the last RUNSTATS on a table space or partition, to the total number of rows or LOBs in the table space or partition, expressed as a percentage. Otherwise null.
SRTINSDELUPDABS	INTEGER	If OBJECTTYPE is TS and RUNSTATS is YES, the total number of insert, update, and delete operations since the last RUNSTATS on a table space or partition. Otherwise null.

Table 338. Result set row for second DSNACCOR result set (continued)

Column name	Data type	Description
SRTMASSDELETE	INTEGER	If OBJECTTYPE is TS and RUNSTATS is YES, the number of mass deletes from the table space or partition since the last REORG or LOAD REPLACE. Otherwise null.
SRIINSDELPCT	INTEGER	If OBJECTTYPE is IX and RUNSTATS is YES, the ratio of the total number of insert and delete operations since the last RUNSTATS on the index space or partition, to the total number of index entries in the index space or partition, expressed as a percentage. Otherwise null.
SRIINSDELABS	INTEGER	If OBJECTTYPE is IX and RUNSTATS is YES, the number insert and delete operations since the last RUNSTATS on the index space or partition. Otherwise null.
SRIMASSDELETE	INTEGER	If OBJECTTYPE is IX and RUNSTATS is YES, the number of mass deletes from the index space or partition since the last REORG, REBUILD INDEX, or LOAD REPLACE. Otherwise, this value is null.
TOTALEXTENTS	SMALLINT	If EXTENTS is YES, the number of physical extents in the table space, index space, or partition. Otherwise, this value is null.

WLM environment refresh stored procedure (WLM_REFRESH)

The WLM_REFRESH stored procedure refreshes a WLM environment. WLM_REFRESH can recycle the environment in which it runs, as well as any other WLM environment.

Environment for WLM_REFRESH

WLM_REFRESH runs in a WLM-established stored procedures address space. The load module for WLM_REFRESH, DSNTWR, must reside in an APF-authorized library.

Authorization required for WLM_REFRESH

To execute the CALL statement, the SQL authorization ID of the process must have READ access or higher to the z/OS Security Server System Authorization Facility (SAF) resource profile *ssid*.WLM_REFRESH.*WLM-environment-name* in resource class DSNR. This is a different resource profile from the *ssid*.WLMENV.*WLM-environment-name* resource profile, which DB2 uses to determine whether a stored procedure or user-defined function is authorized to run in the specified WLM environment.

WLM_REFRESH uses an extended MCS console to monitor the operating system response to a WLM environment refresh request. The privilege to create an extended MCS console is controlled by the resource profile MVS.MCSOPER.* in the OPERCMDS class. If the MVS.MCSOPER.* profile exists, or if the specific profile MVS.MCSOPER.DSNTWR exists, the task ID that is associated with the WLM environment in which WLM_REFRESH runs must have READ access to it.

If the MVS.VARY.* profile exists, or if the specific profile MVS.VARY.WLM exists, the task ID that is associated with the WLM environment in which WLM_REFRESH runs must have CONTROL access to it.

See Chapter 11, “Controlling access to a DB2 subsystem,” on page 231 for information about authorizing access to SAF resource profiles. See *z/OS MVS Planning: Operations* for more information about permitting access to the extended MCS console.

WLM_REFRESH syntax diagram

The WLM_REFRESH stored procedure refreshes a WLM environment. WLM_REFRESH can recycle the environment in which it runs, as well as any other WLM environment.

The following syntax diagram shows the SQL CALL statement for invoking WLM_REFRESH. The linkage convention for WLM_REFRESH is GENERAL WITH NULLS.

```
►►──CALL──WLM_REFRESH──(──WLM-environment,──┬─ssid──┬──,──status-message,──return-code──)──────────►◄
                                            ├─NULL──┤
                                            └'    '─┘
```

WLM_REFRESH option descriptions

WLM-environment
Specifies the name of the WLM environment that you want to refresh. This is an input parameter of type VARCHAR(32).

ssid
Specifies the subsystem ID of the DB2 subsystem with which the WLM environment is associated. If this parameter is NULL or blank, DB2 uses one of the following values for this parameter:

- In a non-data sharing environment, DB2 uses the subsystem ID of the subsystem on which WLM_REFRESH runs.
- In a data sharing environment, DB2 uses the group attach name for the data sharing group in which WLM_REFRESH runs.

This is an input parameter of type VARCHAR(4).

status-message
Contains an informational message about the execution of the WLM refresh. This is an output parameter of type VARCHAR(120).

return-code
Contains the return code from the WLM_REFRESH call, which is one of the following values:

0 WLM_REFRESH executed successfully.

4 One of the following conditions exists:

- The SAF resource profile *ssid*.WLM_REFRESH.*wlm-environment* is not defined in resource class DSNR.
- The SQL authorization ID of the process (CURRENT SQLID) is not defined to SAF.
- # The wait time to obtain a response from z/OS was exceeded.

8 The SQL authorization ID of the process (CURRENT SQLID) is not authorized to refresh the WLM environment.

990 DSNTWR received an unexpected SQLCODE while determining the current SQLID.

993 One of the following conditions exists:

- The *WLM-environment* parameter value is null, blank, or contains invalid characters.
- The *ssid* value contains invalid characters.

994 The extended MCS console was not activated within the number of
seconds indicated by message DSNT5461.

995 DSNTWR is not running as an authorized program.

996 DSNTWR could not activate an extended MCS console. See message DSNT533I for more information.

997 DSNTWR made an unsuccessful request for a message from its extended MCS console. See message DSNT533I for more information.

998 The extended MCS console for DSNTWR posted an alert. See message DSNT534I for more information.

999 The operating system denied an authorized WLM_REFRESH request. See message DSNT545I for more information.

return-code is an output parameter of type INTEGER.

Example of WLM_REFRESH invocation

Suppose that you want to refresh WLM environment WLMENV1, which is associated with a DB2 subsystem with ID DSN. Assume that you already have READ access to the DSN.WLM_REFRESH.WLMENV1 SAF profile. The CALL statement for WLM_REFRESH looks like this:

```
strcpy(WLMENV,"WLMENV1");
strcpy(SSID,"DSN");
EXEC SQL CALL SYSPROC.WLM_REFRESH(:WLMENV, :SSID, :MSGTEXT, :RC);
```

For a complete example of setting up access to an SAF profile and calling WLM_REFRESH, see job DSNTEJ6W, which is in data set DSN810.SDSNSAMP.

The CICS transaction invocation stored procedure (DSNACICS)

The CICS transaction invocation stored procedure (DSNACICS) invokes CICS server programs. DSNACICS gives workstation applications a way to invoke CICS server programs while using TCP/IP as their communication protocol. The workstation applications use TCP/IP and DB2 Connect to connect to a DB2 UDB for z/OS subsystem, and then call DSNACICS to invoke the CICS server programs.

The DSNACICS input parameters require knowledge of various CICS resource definitions with which the workstation programmer might not be familiar. For this reason, DSNACICS invokes the DSNACICX user exit routine. The system programmer can write a version of DSNACICX that checks and overrides the parameters that the DSNACICS caller passes. If no user version of DSNACICX is provided, DSNACICS invokes the default version of DSNACICX, which does not modify any parameters.

Environment for DSNACICS

DSNACICS runs in a WLM-established stored procedure address space and uses the Resource Recovery Services attachment facility to connect to DB2.

If you use CICS Transaction Server for OS/390 Version 1 Release 3 or later, you can register your CICS system as a resource manager with recoverable resource management services (RRMS). When you do that, changes to DB2 databases that are made by the program that calls DSNACICS and the CICS server program that DSNACICS invokes are in the same two-phase commit scope. This means that when the calling program performs an SQL COMMIT or ROLLBACK, DB2 and RRS inform CICS about the COMMIT or ROLLBACK.

If the CICS server program that DSNACICS invokes accesses DB2 resources, the server program runs under a separate unit of work from the original unit of work that calls the stored procedure. This means that the CICS server program might deadlock with locks that the client program acquires.

Authorization required for DSNACICS

To execute the CALL statement, the owner of the package or plan that contains the CALL statement must have one or more of the following privileges:

- The EXECUTE privilege on stored procedure DSNACICS
- Ownership of the stored procedure
- SYSADM authority

The CICS server program that DSNACICS calls runs under the same user ID as DSNACICS. That user ID depends on the SECURITY parameter that you specify when you define DSNACICS. See Part 2 of *DB2 Installation Guide*.

The DSNACICS caller also needs authorization from an external security system, such as RACF, to use CICS resources. See Part 2 of *DB2 Installation Guide*.

DSNACICS syntax diagram

The following syntax diagram shows the SQL CALL statement for invoking DSNACICS. Because the linkage convention for DSNACICS is GENERAL WITH NULLS, if you pass parameters in host variables, you need to include a null indicator with every host variable. Null indicators for input host variables must be initialized before you execute the CALL statement.

```
►►─CALL─DSNACICS─(─┬─parm-level─┬─,─┬─pgm-name─┬─,─┬─CICS-applid─┬─,─┬─CICS-level─┬─,──────►
                   └─NULL───────┘   └─NULL────┘   └─NULL───────┘   └─NULL───────┘

►─┬─connect-type─┬─,─┬─netname─┬─,─┬─mirror-trans─┬─,─┬─COMMAREA─┬─,─┬─COMMAREA-total-len─┬─,──►
  └─NULL───────┘   └─NULL────┘   └─NULL───────┘   └─NULL────┘   └─NULL──────────────┘

►─┬─sync-opts─┬─,─return-code,─msg-area─)──────────────────────────────────────────────────►◄
  └─NULL──────┘
```

DSNACICS option descriptions

parm-level

Specifies the level of the parameter list that is supplied to the stored procedure. This is an input parameter of type INTEGER. The value must be 1.

pgm-name
Specifies the name of the CICS program that DSNACICS invokes. This is the name of the program that the CICS mirror transaction calls, *not* the CICS transaction name. This is an input parameter of type CHAR(8).

CICS-applid
Specifies the applid of the CICS system to which DSNACICS connects. This is an input parameter of type CHAR(8).

CICS-level
Specifies the level of the target CICS subsystem:

1 The CICS subsystem is CICS for MVS/ESA Version 4 Release 1, CICS Transaction Server for OS/390 Version 1 Release 1, or CICS Transaction Server for OS/390 Version 1 Release 2.

2 The CICS subsystem is CICS Transaction Server for OS/390 Version 1 Release 3 or later.

This is an input parameter of type INTEGER.

connect-type
Specifies whether the CICS connection is generic or specific. Possible values are GENERIC or SPECIFIC. This is an input parameter of type CHAR(8).

netname
If the value of *connection-type* is SPECIFIC, specifies the name of the specific connection that is to be used. This value is ignored if the value of *connection-type* is GENERIC. This is an input parameter of type CHAR(8).

mirror-trans
Specifies the name of the CICS mirror transaction to invoke. This mirror transaction calls the CICS server program that is specified in the *pgm-name* parameter. *mirror-trans* must be defined to the CICS server region, and the CICS resource definition for *mirror-trans* must specify DFHMIRS as the program that is associated with the transaction.

If this parameter contains blanks, DSNACICS passes a mirror transaction parameter value of null to the CICS EXCI interface. This allows an installation to override the transaction name in various CICS user-replaceable modules. If a CICS user exit routine does not specify a value for the mirror transaction name, CICS invokes CICS-supplied default mirror transaction CSMI.

This is an input parameter of type CHAR(4).

COMMAREA
Specifies the communication area (COMMAREA) that is used to pass data between the DSNACICS caller and the CICS server program that DSNACICS calls. This is an input/output parameter of type VARCHAR(32704). In the length field of this parameter, specify the number of bytes that DSNACICS sends to the CICS server program.

commarea-total-len
Specifies the total length of the COMMAREA that the server program needs. This is an input parameter of type INTEGER. This length must be greater than or equal to the value that you specify in the length field of the COMMAREA parameter and less than or equal to 32704. When the CICS server program completes, DSNACICS passes the server program's entire COMMAREA, which is *commarea-total-len* bytes in length, to the stored procedure caller.

sync-opts
Specifies whether the calling program controls resource recovery, using two-phase commit protocols that are supported by RRS. Possible values are:

1 The client program controls commit processing. The CICS server region does not perform a syncpoint when the server program returns control to CICS. Also, the server program cannot take any explicit syncpoints. Doing so causes the server program to abnormally terminate.

2 The target CICS server region takes a syncpoint on successful completion of the server program. If this value is specified, the server program can take explicit syncpoints.

When CICS has been set up to be an RRS resource manager, the client application can control commit processing using SQL COMMIT requests. DB2 UDB for z/OS ensures that CICS is notified to commit any resources that the CICS server program modifies during two-phase commit processing.

When CICS has not been set up to be an RRS resource manager, CICS forces syncpoint processing of all CICS resources at completion of the CICS server program. This commit processing is not coordinated with the commit processing of the client program.

This option is ignored when *CICS-level* is 1. This is an input parameter of type INTEGER.

return-code
Return code from the stored procedure. Possible values are:

0 The call completed successfully.

12 The request to run the CICS server program failed. The *msg-area* parameter contains messages that describe the error.

This is an output parameter of type INTEGER.

msg-area
Contains messages if an error occurs during stored procedure execution. The first messages in this area are generated by the stored procedure. Messages that are generated by CICS or the DSNACICX user exit routine might follow the first messages. The messages appear as a series of concatenated, viewable text strings. This is an output parameter of type VARCHAR(500).

DSNACICX user exit routine

DSNACICS always calls user exit routine DSNACICX. You can use DSNACICX to change the values of DSNACICS input parameters before you pass those parameters to CICS. If you do not supply your own version of DSNACICX, DSNACICS calls the default DSNACICX, which modifies no values and does an immediate return to DSNACICS. The source code for the default version of DSNACICX is in member DSNASCIX in data set *prefix*.SDSNSAMP. The source code for a sample version of DSNACICX that is written in COBOL is in member DSNASCIO in data set *prefix*.SDSNSAMP.

General considerations for DSNACICX

The DSNACICX exit routine must follow these rules:

- It can be written in assembler, COBOL, PL/I, or C.
- It must follow the Language Environment calling linkage when the caller is an assembler language program.

- The load module for DSNACICX must reside in an authorized program library that is in the STEPLIB concatenation of the stored procedure address space startup procedure.

  You can replace the default DSNACICX in the *prefix*.SDSNLOAD, library, or you can put the DSNACICX load module in a library that is ahead of *prefix*.SDSNLOAD in the STEPLIB concatenation. It is recommended that you put DSNACICX in the *prefix*.SDSNEXIT library. Sample installation job DSNTIJEX contains JCL for assembling and link-editing the sample source code for DSNACICX into *prefix*.SDSNEXIT. You need to modify the JCL for the libraries and the compiler that you are using.
- The load module must be named DSNACICX.
- The exit routine must save and restore the caller's registers. Only the contents of register 15 can be modified.
- It must be written to be reentrant and link-edited as reentrant.
- It must be written and link-edited to execute as AMODE(31),RMODE(ANY).
- DSNACICX can contain SQL statements. However, if it does, you need to change the DSNACICS procedure definition to reflect the appropriate SQL access level for the types of SQL statements that you use in the user exit routine.

Specifying the DSNACICX exit routine

DSNACICS always calls an exit routine named DSNACICX. DSNACICS calls your DSNACICX exit routine if it finds it before the default DSNACICX exit routine. Otherwise, it calls the default DSNACICX exit routine.

When the DSNACICX exit routine is taken

The DSNACICX exit routine is taken whenever DSNACICS is called. The exit routine is taken before DSNACICS invokes the CICS server program.

Loading a new version of the DSNACICX exit routine

DB2 loads DSNACICX only once, when DSNACICS is first invoked. If you change DSNACICX, you can load the new version by quiescing and then resuming the WLM application environment for the stored procedure address space in which DSNACICS runs:

```
VARY WLM,APPLENV=DSNACICS-applenv-name,QUIESCE
VARY WLM,APPLENV=DSNACICS-applenv-name,RESUME
```

Parameter list for DSNACICX

At invocation, registers are set as described in Table 339.

Table 339. Registers at invocation of DSNACICX

Register	Contains
1	Address of pointer to the exit parameter list (XPL).
13	Address of the register save area.
14	Return address.
15	Address of entry point of exit routine.

Table 340 on page 1285 shows the contents of the DSNACICX exit parameter list, XPL. Member DSNDXPL in data set *prefix*.SDSNMACS contains an assembler language mapping macro for XPL. Sample exit routine DSNASCIO in data set *prefix*.SDSNSAMP includes a COBOL mapping macro for XPL.

Table 340. Contents of the XPL exit parameter list

Name	Hex offset	Data type	Description	Corresponding DSNACICS parameter
XPL_EYEC	0	Character, 4 bytes	Eye-catcher: 'XPL '	
XPL_LEN	4	Character, 4 bytes	Length of the exit parameter list	
XPL_LEVEL	8	4-byte integer	Level of the parameter list	*parm-level*
XPL_PGMNAME	C	Character, 8 bytes	Name of the CICS server program	*pgm-name*
XPL_CICSAPPLID	14	Character, 8 bytes	CICS VTAM applid	*CICS-applid*
XPL_CICSLEVEL	1C	4-byte integer	Level of CICS code	*CICS-level*
XPL_CONNECTTYPE	20	Character, 8 bytes	Specific or generic connection to CICS	*connect-type*
XPL_NETNAME	28	Character, 8 bytes	Name of the specific connection to CICS	*netname*
XPL_MIRRORTRAN	30	Character, 8 bytes	Name of the mirror transaction that invokes the CICS server program	*mirror-trans*
XPL_COMMAREAPTR	38	Address, 4 bytes	Address of the COMMAREA	[1]
XPL_COMMINLEN	3C	4–byte integer	Length of the COMMAREA that is passed to the server program	[2]
XPL_COMMTOTLEN	40	4–byte integer	Total length of the COMMAREA that is returned to the caller	*commarea-total-len*
XPL_SYNCOPTS	44	4–byte integer	Syncpoint control option	*sync-opts*
XPL_RETCODE	48	4–byte integer	Return code from the exit routine	*return-code*
XPL_MSGLEN	4C	4–byte integer	Length of the output message area	*return-code*
XPL_MSGAREA	50	Character, 256 bytes	Output message area	*msg-area*[3]

Note:

1. The area that this field points to is specified by DSNACICS parameter *COMMAREA*. This area does not include the length bytes.
2. This is the same value that the DSNACICS caller specifies in the length bytes of the *COMMAREA* parameter.
3. Although the total length of *msg-area* is 500 bytes, DSNACICX can use only 256 bytes of that area.

Example of DSNACICS invocation

The following PL/I example shows the variable declarations and SQL CALL statement for invoking the CICS transaction that is associated with program CICSPGM1.

```
/************************/
/* DSNACICS PARAMETERS */
/************************/
DECLARE  PARM_LEVEL  BIN FIXED(31);
DECLARE  PGM_NAME    CHAR(8);
DECLARE  CICS_APPLID CHAR(8);
DECLARE  CICS_LEVEL  BIN FIXED(31);
DECLARE  CONNECT_TYPE CHAR(8);
```

```
DECLARE  NETNAME     CHAR(8);
DECLARE  MIRROR_TRANS CHAR(4);
DECLARE  COMMAREA_TOTAL_LEN BIN FIXED(31);
DECLARE  SYNC_OPTS   BIN FIXED(31);
DECLARE  RET_CODE    BIN FIXED(31);
DECLARE  MSG_AREA    CHAR(500) VARYING;

DECLARE 1 COMMAREA  BASED(P1),
          3 COMMAREA_LEN BIN FIXED(15),
          3 COMMAREA_INPUT   CHAR(30),
          3 COMMAREA_OUTPUT  CHAR(100);

/***********************************************/
/* INDICATOR VARIABLES FOR DSNACICS PARAMETERS */
/***********************************************/
DECLARE 1 IND_VARS,
          3 IND_PARM_LEVEL   BIN FIXED(15),
          3 IND_PGM_NAME     BIN FIXED(15),
          3 IND_CICS_APPLID  BIN FIXED(15),
          3 IND_CICS_LEVEL   BIN FIXED(15),
          3 IND_CONNECT_TYPE BIN FIXED(15),
          3 IND_NETNAME      BIN FIXED(15),
          3 IND_MIRROR_TRANS BIN FIXED(15),
          3 IND_COMMAREA     BIN FIXED(15),
          3 IND_COMMAREA_TOTAL_LEN  BIN FIXED(15),
          3 IND_SYNC_OPTS    BIN FIXED(15),
          3 IND_RETCODE      BIN FIXED(15),
          3 IND_MSG_AREA     BIN FIXED(15);

/**************************/
/* LOCAL COPY OF COMMAREA */
/**************************/
DECLARE P1 POINTER;
DECLARE COMMAREA_STG CHAR(130) VARYING;
/*****************************************************************/
/* ASSIGN VALUES TO INPUT PARAMETERS PARM_LEVEL, PGM_NAME,    */
/* MIRROR_TRANS, COMMAREA, COMMAREA_TOTAL_LEN, AND SYNC_OPTS. */
/* SET THE OTHER INPUT PARAMETERS TO NULL. THE DSNACICX       */
/* USER EXIT MUST ASSIGN VALUES FOR THOSE PARAMETERS.         */
/*****************************************************************/
PARM_LEVEL = 1;
IND_PARM_LEVEL = 0;

PGM_NAME = 'CICSPGM1';
IND_PGM_NAME = 0 ;

MIRROR_TRANS = 'MIRT';
IND_MIRROR_TRANS = 0;

P1 = ADDR(COMMAREA_STG);
COMMAREA_INPUT = 'THIS IS THE INPUT FOR CICSPGM1';
COMMAREA_OUTPUT = ' ';
COMMAREA_LEN = LENGTH(COMMAREA_INPUT);
IND_COMMAREA = 0;

COMMAREA_TOTAL_LEN = COMMAREA_LEN + LENGTH(COMMAREA_OUTPUT);
IND_COMMAREA_TOTAL_LEN = 0;

SYNC_OPTS = 1;
IND_SYNC_OPTS = 0;

IND_CICS_APPLID= -1;
IND_CICS_LEVEL = -1;
IND_CONNECT_TYPE = -1;
IND_NETNAME = -1;
/****************************************/
/* INITIALIZE OUTPUT PARAMETERS TO NULL. */
```

```
/****************************************/
IND_RETCODE = -1;
IND_MSG_AREA= -1;
/****************************************/
/* CALL DSNACICS TO INVOKE CICSPGM1.     */
/****************************************/
EXEC SQL
 CALL SYSPROC.DSNACICS(:PARM_LEVEL         :IND_PARM_LEVEL,
                       :PGM_NAME           :IND_PGM_NAME,
                       :CICS_APPLID        :IND_CICS_APPLID,
                       :CICS_LEVEL         :IND_CICS_LEVEL,
                       :CONNECT_TYPE       :IND_CONNECT_TYPE,
                       :NETNAME            :IND_NETNAME,
                       :MIRROR_TRANS       :IND_MIRROR_TRANS,
                       :COMMAREA_STG       :IND_COMMAREA,
                       :COMMAREA_TOTAL_LEN :IND_COMMAREA_TOTAL_LEN,
                       :SYNC_OPTS          :IND_SYNC_OPTS,
                       :RET_CODE           :IND_RETCODE,
                       :MSG_AREA           :IND_MSG_AREA);
```

DSNACICS output

DSNACICS places the return code from DSNACICS execution in the *return-code* parameter. If the value of the return code is non-zero, DSNACICS puts its own error messages and any error messages that are generated by CICS and the DSNACICX user exit routine in the *msg-area* parameter.

The *COMMAREA* parameter contains the COMMAREA for the CICS server program that DSNACICS calls. The *COMMAREA* parameter has a VARCHAR type. Therefore, if the server program puts data other than character data in the COMMAREA, that data can become corrupted by code page translation as it is passed to the caller. To avoid code page translation, you can change the COMMAREA parameter in the CREATE PROCEDURE statement for DSNACICS to VARCHAR(32704) FOR BIT DATA. However, if you do so, the client program might need to do code page translation on any character data in the COMMAREA to make it readable.

DSNACICS restrictions

Because DSNACICS uses the distributed program link (DPL) function to invoke CICS server programs, server programs that you invoke through DSNACICS can contain only the CICS API commands that the DPL function supports. The list of supported commands is documented in CICS Transaction Server for z/OS Application Programming Reference.

DSNACICS debugging

If you receive errors when you call DSNACICS, ask your system administrator to add a DSNDUMP DD statement in the startup procedure for the address space in which DSNACICS runs. The DSNDUMP DD statement causes DB2 to generate an SVC dump whenever DSNACICS issues an error message.

The SYSIBM.USERNAMES encryption stored procedure

The DSNLEUSR stored procedure is a sample stored procedure that lets you store encrypted values in the following fields of the SYSIBM.USERNAMES table.

- The translated authorization ID (NEWAUTHID)
- The password (PASSWORD)

You provide all the values for a SYSIBM.USERNAMES row as input to DSNLEUSR. DSNLEUSR encrypts the translated authorization ID and password values before it inserts the row into SYSIBM.USERNAMES.

Environment for DSNLEUSR

DSNLEUSR has the following requirements:

- The DB2 subsystem needs to be in DB2 Version 8 new-function mode.
- DSNLEUSR runs in a WLM-established stored procedure address space.
- z/OS Integrated Cryptographic Service Facility (ICSF) must be installed, configured, and active. See *Integrated Cryptographic Service Facility System Programmer's Guide* for more information.

Authorization required for DSNLEUSR

To execute the CALL DSNLEUSR statement, the owner of the package or plan that contains the CALL statement must have one or more of the following privileges:

- The EXECUTE privilege on the package for DSNLEUSR
- Ownership of the package
- PACKADM authority for the package collection
- SYSADM authority

The owner of the package or plan that contains the CALL statement must also have INSERT authority on SYSIBM.USERNAMES.

DSNLEUSR syntax diagram

The following syntax diagram shows the CALL statement for invoking DSNLEUSR. Because the linkage convention for DSNLEUSR is GENERAL WITH NULLS, if you pass parameters in host variables, you need to include a null indicator with every host variable. Null indicators for input host variables must be initialized before you execute the CALL statement.

```
►►──CALL──DSNLEUSR──(──Type,──┬─AuthID─┬─,──┬─LinkName─┬─,──┬─NewAuthID─┬─,──┬─Password─┬─,──────►
                              └─NULL───┘    └─NULL─────┘    └─NULL───────┘    └─NULL─────┘

►─ReturnCode,──MsgArea──)──────────────────────────────────────────────────────────────────►◄
```

DSNLEUSR option descriptions

Type
Specifies the value that is to be inserted into the TYPE column of SYSIBM.USERNAMES. *Type* is an input parameter of type CHAR(1).

AuthID
Specifies the value that is to be inserted into the AUTHID column of SYSIBM.USERNAMES. *AuthID* is an input parameter of type VARCHAR(128). If you specify a null value, DSNLEUSR does not insert a value for *AuthID*.

LinkName
Specifies the value that is to be inserted into the LINKNAME column of SYSIBM.USERNAMES. *LinkName* is an input parameter of type CHAR(8). If you specify a null value, DSNLEUSR does not insert a value for *LinkName*.

NewAuthID
Specifies the value that is to be inserted into the NEWAUTHID column of SYSIBM.USERNAMES. *NewAuthID* is an input parameter of type VARCHAR(119).

Although the NEWAUTHID field of SYSIBM.USERNAMES is VARCHAR(128), your input value is restricted to 119 or fewer bytes.

If you specify a null value, DSNLEUSR does not insert a value for *NewAuthID*.

Password
Specifies the value that is to be inserted into the PASSWORD column of SYSIBM.USERNAMES. *Password* is an input parameter of type CHAR(8).

Although the PASSWORD field of SYSIBM.USERNAMES is VARCHAR(24), your input value is restricted to 8 or fewer bytes.

If you specify a null value, DSNLEUSR does not insert a value for *Password*.

ReturnCode
The return code from DSNLEUSR execution. Possible values are:

0 DSNLEUSR executed successfully.

8 The request to encrypt the translated authorization ID or password failed. *MsgArea* contains the following fields:

- An unformatted SQLCA that describes the error. See Appendix D of *DB2 SQL Reference* for the description of the SQLCA.
- A string that contains a DSNL045I message with the ICSF return code, the ICSF reason code, and the ICSF function that failed. The string immediately follows the SQLCA field and does not begin with a length field.

12 The insert operation for the SYSIBM.USERNAMES row failed. *MsgArea* contains an SQLCA that describes the error.

16 DSNLEUSR terminated because the DB2 subsystem is not in DB2 Version 8 new-function mode. *MsgArea* contains an SQLCA that describes the error.

ReturnCode is an output parameter of type INTEGER.

MsgArea
Contains information about DSNLEUSR execution. The information that is returned is described in the *ReturnCode* description. *MsgArea* is an output parameter of type VARCHAR(500).

Example of DSNLEUSR invocation

Figure 190 on page 1290 is a COBOL example that shows variable declarations and an SQL CALL for inserting a row into SYSIBM.USERNAMES with an encrypted translated authorization ID and an encrypted password.

```
 WORKING-STORAGE SECTION.
:
**********************
* DSNLEUSR PARAMETERS *
**********************
 01  TYPE                    PICTURE X(1).
 01  AUTHID.
     49   AUTHID-LN          PICTURE S9(4).
     49   AUTHID-DTA         PICTURE X(128).
 01  LINKNAME                PICTURE X(8).
 01  NEWAUTHID.
     49   NEWAUTHID-LN       PICTURE S9(4).
     49   NEWAUTHID-DTA      PICTURE X(119).
 01  PASSWORD                PICTURE X(8).
 01  RETURNCODE              PICTURE S9(9) COMP VALUE +0.
 01  MSGAREA.
     49   MSGAREA-LN         PICTURE S9(4) COMP VALUE 500.
     49   MSGAREA-DTA        PICTURE X(500) VALUE SPACES.
****************************************
* INDICATOR VARIABLES.                  *
****************************************
 01  TYPE-IND                PICTURE S9(4) COMP-4.
 01  AUTHID-IND              PICTURE S9(4) COMP-4.
 01  LINKNAME-IND            PICTURE S9(4) COMP-4.
 01  NEWAUTHID-IND           PICTURE S9(4) COMP-4.
 01  PASSWORD-IND            PICTURE S9(4) COMP-4.
 01  RETURNCODE-IND          PICTURE S9(4) COMP-4.
 01  MSGAREA-IND             PICTURE S9(4) COMP-4.
 PROCEDURE DIVISION.
:
*********************************************************
* SET VALUES FOR DSNLEUSR INPUT PARAMETERS.             *
* THE SET OF INPUT VALUES REPRESENTS A ROW THAT         *
* DSNLEUSR INSERTS INTO SYSIBM.USERNAMES WITH           *
* ENCRYPTED NEWAUTHID AND PASSWORD VALUES.              *
*********************************************************
     MOVE 'O' TO TYPE.
     MOVE 0 TO AUTHID-LN.
     MOVE SPACES TO AUTHID-DTA.
     MOVE 'SYEC1B  ' TO LINKNAME.
     MOVE 4 TO NEWAUTHID-LN.
     MOVE 'MYID' TO NEWAUTHID-DTA.
     MOVE 'MYPASS' TO PASSWORD.
*****************
* CALL DSNLEUSR *
*****************
     EXEC SQL
      CALL SYSPROC.DSNLEUSR
      (:TYPE           :TYPE-IND,
       :AUTHID         :AUTHID-IND,
       :LINKNAME       :LINKNAME-IND,
       :NEWAUTHID      :NEWAUTHID-IND,
       :PASSWORD       :PASSWORD-IND,
       :RETURNCODE     :RETURNCODE-IND,
       :MSGAREA        :MSGAREA-IND)
     END-EXEC.
```

Figure 190. Example of DSNLEUSR invocation

DSNLEUSR output

If DSNLEUSR executes successfully, it inserts a row into SYSIBM.USERNAMES with encrypted values for the NEWAUTHID and PASSWORD columns and returns 0 for the *ReturnCode* parameter value. If DSNLEUSR does not execute successfully, it returns a non-zero value for the *ReturnCode* value and additional diagnostic information for the *MsgArea* parameter value. See “DSNLEUSR option descriptions” on page 1288 for a description of the *ReturnCode* and *MsgArea* contents.

IMS transactions stored procedure (DSNAIMS)

DSNAIMS is a stored procedure that allows DB2 applications to invoke IMS transactions and commands easily, without maintaining their own connections to IMS. DSNAIMS uses the IMS Open Transaction Manager Access (OTMA) API to connect to IMS and execute the transactions.

Environment for DSNAIMS

DSNAIMS runs in a WLM-established stored procedures address space. DSNAIMS requires DB2 with RRSAF enabled and IMS version 7 or later with OTMA Callable Interface enabled.

To use a two-phase commit process, you must have IMS Version 8 with UQ70789 or later.

Authorization required for DSNAIMS

To set up and run DSNAIMS, you must be authorized the perform the following steps:

1. Use the job DSNTIJIM to issue the CREATE PROCEDURE statement for DSNAIMS and to grant the execution of DSNAIMS to PUBLIC. DSNTIJIM is provided in the SDSNSAMP data set. You need to customize DSNTIJIM to fit the parameters of your system.
2. Ensure that OTMA C/I is initialized. Refer to IMS Open Transaction Manager Access Guide and Reference for an explanation of the C/I initialization.

DSNAIMS syntax diagram

The following syntax diagram shows the SQL CALL statement for invoking DSNAIMS.

```
►►─CALL─SYSPROC.DSNAIMS─(─dsnaims-function─,─┬─dsnaims-2pc─┬─,─xcf-group-name─,──────────►
                                             └─NULL────────┘

►─xcf-ims-name─,─racf-userid─,─┬─racf-groupid─┬─,─┬─ims-lterm─┬─,─┬─ims-modname─┬─,──────►
                               └─NULL─────────┘   └─NULL───────┘   └─NULL────────┘

►─┬─ims-tran-name─┬─,─┬─ims-data-in─┬─,─┬─ims-data-out─┬─,─┬─otma-tpipe-name─┬─,──────────►
  └─NULL──────────┘   └─NULL────────┘   └─NULL─────────┘   └─NULL───────────┘

►─┬─otma-dru-name─┬─,─┬─user-data-in─┬─,─user-data-out─,─status-message─,─return-code─)──►◄
  └─NULL──────────┘   └─NULL─────────┘
```

DSNAIMS option descriptions

dsnaims-function

A string that indicates whether the transaction is send-only, receive-only, or send-and-receive:

SENDRECV

Sends and receives IMS data. SENDRECV invokes an IMS transaction or command and returns the result to the caller. The transaction can be

an IMS full function or a fast path. SENDRECV does not support multiple iterations of a conversational transaction

SEND Sends IMS data. SEND invokes an IMS transaction or command, but does not receive IMS data. If result data exists, it can be retrieved with the RECEIVE function. A send-only transaction cannot be an IMS fast path transaction or a conversations transaction.

RECEIVE
Receives IMS data. The data can be the result of a transaction or command initiated by the SEND function or an unsolicited output message from an IMS application. The RECEIVE function does not initiate an IMS transaction or command.

dsnaims-2pc

Specifies whether to use a two-phase commit process to perform the transaction syncpoint service. Possible values are **Y** or **N**. For **N**, commits and rollbacks that are issued by the IMS transaction do not affect commit and rollback processing in the DB2 application that invokes DNAIMS. Furthermore, IMS resources are not affected by commits and rollbacks that are issued by the calling DB2 application. If you specify Y, you must also specify SENDRECV. To use a two-phase commit process, you must set the IMS control region parameter (RRS) to Y.

This parameter is optional. The default is N.

xcf-group-name
Specifies the XCF group name that the IMS OTMA joins. You can obtain this name by viewing the GRNAME parameter in IMS PROCLIB member DFSPBxxx or by using the IMS command /DISPLAY OTMA.

xcf-ims-name
Specifies the XCF member name that IMS uses for the XCF group. If IMS is not using the XRF or RSR feature, you can obtain the XCF member name from the OTMANM parameter in IMS PROCLIB member DFSPBxxx. If IMS is using the XRF or RSR feature, you can obtain the XCF member name from the USERVAR parameter in IMS PROCLIB member DFSPBxxx.

racf-userid
Specifies the RACF user ID that is used for IMS to perform the transaction or command authorization checking. This parameter is required if DSNAIMS is running APF-authorized. If DSNAIMS is running unauthorized, this parameter is ignored and the EXTERNAL SECURITY setting for the DSNAIMS stored procedure definition determines the user ID that is used by IMS.

racf-groupid
Specifies the RACF group ID that is used for IMS to perform the transaction or command authorization checking. *racf_groupid* is used for stored procedures that are APF-authorized. It is ignored for other stored procedures.

ims-lterm
Specifies an IMS LTERM name that is used to override the LTERM name in the I/O program communication block of the IMS application program. This field is used as an input and an output field:

- For SENDRECV, the value is sent to IMS on input and can be updated by IMS on output.
- For SEND, the parameter is IN only.
- For RECEIVE, the parameter is OUT only.

An empty or NULL value tells IMS to ignore the parameter.

ims-modname
Specifies the formatting map name that is used by the server to map output data streams, such as 3270 streams. Although this invocation does not have IMS MFS support, the input MODNAME can be used as the map name to define the output data stream. This name is an 8-byte message output descriptor name that is placed in the I/O program communication block. When the message is inserted, IMS places this name in the message prefix with the map name in the program communication block of the IMS application program.

For SENDRECV, the value is sent to IMS on input, and can be updated on output. For SEND, the parameter is IN only. For RECEIVE it is OUT only. IMS ignores the parameter when it is an empty or NULL value.

ims-tran-name
Specifies the name of an IMS transaction or command that is sent to IMS. If the IMS command is longer than eight characters, specify the first eight characters (including the "/" of the command). Specify the remaining characters of the command in the *ims-tran-name* parameter. If you use an empty or NULL value, you must specify the full transaction name or command in the *ims-data-in* parameter.

ims-data-in
Specifies the data that is sent to IMS. This parameter is required in each of the following cases:

- Input data is required for IMS
- No transaction name or command is passed in *ims-tran-name*
- The command is longer than eight characters

This parameter is ignored when for RECEIVE functions.

ims-data-out
Data returned after successful completion of the transaction. This parameter is required for SENDRECV and RECEIVE functions. The parameter is ignored for SEND functions.

otma-tpipe-name
Specifies an 8-byte user-defined communication session name that IMS uses for the input and output data for the transaction or the command in a SEND or a RECEIVE function. If the *otma_tpipe_name* parameter is used for a SEND function to generate an IMS output message, the same otma_pipe_name must be used to retrieve output data for the subsequent RECEIVE function.

otma-dru-name
Specifies the name of an IMS user-defined exit routine, OTMA destination resolution user exit routine, if it is used. This IMS exit routine can format part of the output prefix and can determine the output destination for an IMS ALT_PCB output. If an empty or null value is passed, IMS ignores this parameter.

user-data-in
This optional parameter contains any data that is to be included in the IMS message prefix, so that the data can be accessed by IMS OTMA user exit routines (DFSYIOE0 and DFSYDRU0) and can be tracked by IMS log records. IMS applications that run in dependent regions do not access this data. The specifed user data is not included in the output message prefix. You can use this parameter to store input and output correlator tokens or other information. This parameter is ignored for RECEIEVE functions.

user-data-out
On output, this field contains the *user-data-in* in the IMS output prefix. IMS user exit routines (DFSYIOE0 and DFSYDRU0) can also create *user-data-out* for SENDRECV and RECEIVE functions. The parameter is not updated for SEND functions.

status-message
Indicates any error message that is returned from the transaction or command, OTMA, RRS, or DSNAIMS.

return-code
Indicates the return code that is returned for the transaction or command, OTMA, RRS, or DSNAIMS.

Examples of DSNAIMS invocation

The following examples show how to call DSNAIMS.

Example 1: Sample parameters for executing an IMS command:

```
CALL SYSPROC.DSNAIMS("SENDRECV", "N", "IMS7GRP", "IMS7TMEM",
                     "IMSCLNM", "", "", "", "", "",
                     "/LOG Hello World.", ims_data_out, "", "", "",
                     user_out, error_message, rc)
```

Example 2: Sample parameters for executing an IMS IVTNO transaction:

```
CALL SYSPROC.DSNAIMS("SENDRECV", "N", "IMS7GRP", "IMS7TMEM",
                     "IMSCLNM", "", "", "", "", "",
                     "IVTNO     DISPLAY LAST1     "", ims_data_out
                     "", "", "", user_out, error_message, rc)
```

Example 3: Sample parameters for send-only IMS transaction:

```
CALL SYSPROC.DSNAIMS("SEND", "N", "IMS7GRP", "IMS7TMEM",
                     "IMSCLNM", "", "", "", "", "",
                     "IVTNO     DISPLAY LAST1     "", ims_data_out,
                     "DSNAPIPE", "", "", user_out, error_message, rc)
```

Example 4: Sample parameters for receive-only IMS transaction:

```
CALL SYSPROC.DSNAIMS("RECEIVE", "N", "IMS7GRP", "IMS7TMEM",
                     "IMSCLNM", "", "", "", "", "",
                     "IVTNO     DISPLAY LAST1     "", ims_data_out,
                     "DSNAPIPE", "", "", user_out, error_message, rc)
```

Connecting to multiple IMS subsystems with DSNAIMS

By default DSNAIMS connects to only one IMS subsystem at a time. The first request to DSNAIMS determines to which IMS subsystem the stored procedure connects. DSNAIMS attempts to reconnect to IMS only in the following cases:

- IMS is restarted and the saved connection is no longer valid
- WLM loads another DSNAIMS task

To connect to multiple IMS subsystems simultaneously, perform the following steps:

1. Make a copy of the DB2-supplied job DSNTIJIM and customize it to your environment.
2. Change the procedure name from SYSPROCC.DSNAIMS to another name, such as DSNAIMSB.
3. Do no change the EXTERNAL NAME option. Leave it as DSNAIMS.
4. Run the new job to create a second instance of the stored procedure.

5. To ensure that you connect to the intended IMS target, consistently use the XFC group and member names that you associate with each stored procedure instance.

   Example:

   ```
   CALL SYSPROC.DSNAIMS("SENDRECV", "N", "IMS7GRP", "IMS7TMEM", ...)
   CALL SYSPROC.DSNAIMSB("SENDRECV", "N", "IMS8GRP", "IMS8TMEM", ...)
   ```

The DB2 EXPLAIN stored procedure

The information under this heading is Product-sensitive Programming Interface and Associated Guidance Information, as defined in “Notices” on page 1301.

The DSNAEXP stored procedure is a sample stored procedure that lets users perform an EXPLAIN on an SQL statement without having the authorization to execute that SQL statement.

Note: The DSNAEXP stored procedure replaces the DSN8EXP stored procedure for DB2 Version 8 new function mode. In DB2 Version 8 compatibility mode you must use the DSN8EXP stored procedure.

Environment

DSNAEXP must run in a WLM-established stored procedure address space.

Before you can invoke DSNAEXP, table *sqlid*.PLAN_TABLE must exist. *sqlid* is the value that you specify for the *sqlid* input parameter when you call DSNAEXP.

Job DSNTESC in DSN8810.SDSNSAMP contains a sample CREATE TABLE statement for the PLAN_TABLE.

Authorization required

To execute the CALL DSN8.DSNAEXP statement, the owner of the package or plan that contains the CALL statement must have one or more of the following privileges on each package that the stored procedure uses:

- The EXECUTE privilege on the package for DSNAEXP
- Ownership of the package
- PACKADM authority for the package collection
- SYSADM authority

In addition:

- The SQL authorization ID of the process in which DSNAEXP is called must have the authority to execute SET CURRENT SQLID=*sqlid*.
- The SQL authorization ID of the process must also have one of the following characteristics:
  - Be the owner of a plan table named PLAN_TABLE
  - Have an alias on a plan table named *owner*.PLAN_TABLE and have SELECT and INSERT privileges on the table

DSNAEXP syntax diagram

The following syntax diagram shows the CALL statement for invoking DSNAEXP. Because the linkage convention for DSNAEXP is GENERAL, you cannot pass null values to the stored procedure.

```
►►──CALL──DSNAEXP──(──sqlid,──queryno,──sql-statement,──parse,──qualifier,──sqlcode,──sqlstate,──────►
►─error-message──)──────────────────────────────────────────────────────────────────────────────────►◄
```

DSNAEXP option descriptions

sqlid
Specifies:

- The authorization ID under which the EXPLAIN statement is to be executed
- The qualifier for the table into which EXPLAIN output is written: *sqlid*.PLAN_TABLE
- The implicit qualifier for unqualified objects in the SQL statement on which EXPLAIN is executed, if the value of *parse* is 'N'

sqlid is an input parameter of type CHAR(8).

queryno
Specifies a number that is to be used to identify *sql-statement* in the EXPLAIN output. This number appears in the QUERYNO column in the PLAN_TABLE. *queryno* is an input parameter of type INTEGER.

sql-statement
Specifies the SQL statement on which EXPLAIN is to be executed. *sql-statement* is an input parameter of type CLOB(2M).

parse
Specifies whether DSNAEXP adds a qualifier for unqualified table or view names in the input SQL statement. Valid values are 'Y' and 'N'. If the value of *parse* is 'Y', *qualifier* must contain a valid SQL qualifier name.

If *sql-statement* is insert-within-select and common table expressions, you need to disable the parsing functionality, and add the qualifier manually.

parse is an input parameter of type CHAR(1).

qualifier
Specifies the qualifier that DSNAEXP adds to unqualified table or view names in the input SQL statement. If the value of *parse* is 'N', *qualifier* is ignored.

If the statement on which EXPLAIN is run contains an INSERT within a SELECT or a common table expression, *parse* must be 'N', and table and view qualifiers must be explicitly specified.

qualifier is an input parameter of type CHAR(8).

sqlcode
Contains the SQLCODE from execution of the EXPLAIN statement. *sqlcode* is an output parameter of type INTEGER.

sqlstate
Contains the SQLSTATE from execution of the EXPLAIN statement. *sqlstate* is an output parameter of type CHAR(5).

error-message
Contains information about DSNAEXP execution. If the SQLCODE from

execution of the EXPLAIN statement is not 0, *error-message* contains the error message for the SQLCODE. *error-message* is an output parameter of type VARCHAR(960).

Example of DSNAEXP invocation

The following C example shows how to call DSNAEXP to execute an EXPLAIN statement.

```
EXEC SQL BEGIN DECLARE SECTION;
 char        hvsqlid[9];               /* Qualifier for PLAN_TABLE   */
 long int    hvqueryno;                /* QUERYNO to give the SQL    */
 struct {
   short int hvsql_stmt_len;           /* Input SQL statement length */
             hvsql_stmt_txt SQL TYPE IS CLOB 2M;
                                       /* SQL statement text         */
        }    hvsql_stmt  ;             /* SQL statement to EXPLAIN   */
 char        hvparse[1];               /* Qualify object names       */
                                       /* indicator                  */
 char        hvqualifier[9];           /* Qualifier for unqualified  */
                                       /* objects                    */
 long int    hvsqlcode;                /* SQLCODE from CALL DSNAEXP  */
 char        hvsqlstate[6];            /* SQLSTATE from CALL DSNAEXP */
 struct {
   short int hvmsg_len;                /* Error message length       */
   char      hvmsg_text[961];          /* Error message text         */
        }    hvmsg;                    /* Error message from failed  */
                                       /* CALL DSNAEXP               */
EXEC SQL END DECLARE SECTION;
short int i;
:
/* Set the input parameters */
strcpy(hvsqlid,"ADMF001 ");
hvqueryno=320;
strcpy(hvsql_stmt.hvsql_stmt_text,"SELECT CURRENT DATE FROM SYSDUMMY1");
hvsql_stmt.hvsql_stmt_len=strlen(hvsql_stmt.hvsql_stmt_text);
hvparse[0]='Y';
strcpy(hvqualifier,"SYSIBM");
/* Initialize the output parameters */
hvsqlcode=0;
for (i = 0; i < 5; i++) hvsqlstate[i] = '0';
hvsqlstate[5]='\0';
hvmsg.hvmsg_len=0;
for (i = 0; i < 960; i++) hvmsg.hvmsg_text[i] = ' ';
hvmsg.hvmsg_text[960] = '\0';
/* Call DSNAEXP to do EXPLAIN and put output in ADMF001.PLAN_TABLE */
EXEC SQL
  CALL DSN8.DSNAEXP(:hvsqlid,
                    :hvqueryno,
                    :hvsql_stmt,
                    :hvparse,
                    :hvqualifier,
                    :hvsqlcode,
                    :hvsqlstate,
                    :hvmsg);
```

DSNAEXP output

If DSNAEXP executes successfully, *sqlid*.PLAN_TABLE contains the EXPLAIN output. A user with SELECT authority on *sqlid*.PLAN_TABLE can obtain the results of the EXPLAIN that was executed by DSNAEXP by executing this query:

```
SELECT * FROM sqlid.PLAN_TABLE WHERE QUERYNO='queryno';
```

If DSNAEXP does not execute successfully, *sqlcode*, *sqlstate*, and *error-message* contain error information.

Appendix K. How to use the DB2 library

Titles of books in the library begin with DB2 Universal Database for z/OS Version 8. However, references from one book in the library to another are shortened and do not include the product name, version, and release. Instead, they point directly to the section that holds the information. For a complete list of books in the library, and the sections in each book, see the bibliography at the back of this book.

The most rewarding task associated with a database management system is asking questions of it and getting answers, the task called *end use*. Other tasks are also necessary—defining the parameters of the system, putting the data in place, and so on. The tasks that are associated with DB2 are grouped into the following major categories (but supplemental information relating to all of the following tasks for new releases of DB2 can be found in *DB2 Release Planning Guide*.

Installation: If you are involved with DB2 only to install the system, *DB2 Installation Guide* might be all you need.

If you will be using data sharing capabilities you also need *DB2 Data Sharing: Planning and Administration*, which describes installation considerations for data sharing.

If you want to set up a DB2 subsystem to meet the requirements of the Common Criteria, you need *DB2 Common Criteria Guide*, which contains information that supersedes other information in the DB2 UDB for z/OS library regarding Common Criteria.

End use: End users issue SQL statements to retrieve data. They can also insert, update, or delete data, with SQL statements. They might need an introduction to SQL, detailed instructions for using SPUFI, and an alphabetized reference to the types of SQL statements. This information is found in *DB2 Application Programming and SQL Guide*, and *DB2 SQL Reference*.

End users can also issue SQL statements through the DB2 Query Management Facility (QMF) or some other program, and the library for that licensed program might provide all the instruction or reference material they need. For a list of the titles in the DB2 QMF library, see the bibliography at the end of this book.

Application programming: Some users access DB2 without knowing it, using programs that contain SQL statements. DB2 application programmers write those programs. Because they write SQL statements, they need the same resources that end users do.

Application programmers also need instructions on many other topics:

- How to transfer data between DB2 and a host program—written in Java, C, or COBOL, for example
- How to prepare to compile a program that embeds SQL statements
- How to process data from two systems simultaneously, say DB2 and IMS or DB2 and CICS
- How to write distributed applications across operating systemss
- How to write applications that use Open Database Connectivity (ODBC) to access DB2 servers

© Copyright IBM Corp. 1982, 2008

- How to write applications in the Java programming language to access DB2 servers

The material needed for writing a host program containing SQL is in *DB2 Application Programming and SQL Guide* and in *DB2 Application Programming Guide and Reference for Java*. The material needed for writing applications that use DB2 ODBC or ODBC to access DB2 servers is in *DB2 ODBC Guide and Reference*. For handling errors, see *DB2 Codes*.

If you will be working in a distributed environment, you will need *DB2 Reference for Remote DRDA Requesters and Servers*.

Information about writing applications across operating systems can be found in *IBM DB2 Universal Database SQL Reference for Cross-Platform Development*.

System and database administration: *Administration* covers almost everything else.

If you will be using the RACF access control module for DB2 authorization checking, you will need *DB2 RACF Access Control Module Guide*.

If you are involved with DB2 only to design the database, or plan operational procedures, you need *DB2 Administration Guide*. If you also want to carry out your own plans by creating DB2 objects, granting privileges, running utility jobs, and so on, you also need:

- *DB2 SQL Reference*, which describes the SQL statements you use to create, alter, and drop objects and grant and revoke privileges
- *DB2 Utility Guide and Reference*, which explains how to run utilities
- *DB2 Command Reference*, which explains how to run commands

If you will be using data sharing, you need *DB2 Data Sharing: Planning and Administration*, which describes how to plan for and implement data sharing.

Additional information about system and database administration can be found in *DB2 Messages* and *DB2 Codes*, which list messages and codes issued by DB2, with explanations and suggested responses.

Diagnosis: Diagnosticians detect and describe errors in the DB2 program. They might also recommend or apply a remedy. The documentation for this task is in *DB2 Diagnosis Guide and Reference*, *DB2 Messages*, and *DB2 Codes*.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

© Copyright IBM Corp. 1982, 2008

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Programming Interface Information

This book is intended to help you to plan for and administer DB2 Universal Database for z/OS (DB2 UDB for z/OS).

This book also documents General-use Programming Interface and Associated Guidance Information and Product-sensitive Programming Interface and Associated Guidance Information provided by DB2 Universal Database for z/OS.

General-use Programming Interfaces allow the customer to write programs that obtain the services of DB2 UDB for z/OS.

General-use Programming Interface and Associated Guidance Information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking:

General-use Programming Interface

General-use Programming Interface and Associated Guidance Information ...

End of General-use Programming Interface

Product-sensitive Programming Interfaces allow the customer installation to perform tasks such as diagnosing, modifying, monitoring, repairing, tailoring, or tuning of this IBM software product. Use of such interfaces creates dependencies on the detailed design or implementation of the IBM software product. Product-sensitive Programming Interfaces should be used only for these specialized purposes. Because of their dependencies on detailed design and implementation, it is to be expected that programs written to such interfaces may need to be changed in order to run with new product releases or versions, or as a result of service.

Product-sensitive Programming Interface and Associated Guidance Information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking:

Product-sensitive Programming Interface

Product-sensitive Programming Interface and Associated Guidance Information ...

End of Product-sensitive Programming Interface

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

3090
BookManager
CICS
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Universal Database
DFSMSdfp
DFSMSdss
DFSMShsm
Distributed Relational Database Architecture
DRDA
Enterprise Storage Server
ES/3090
ESCON
eServer
FICON
FlashCopy
IBM
IBM Registry
ibm.com
IMS
iSeries
Language Environment
MQSeries
MVS
MVS/ESA
NetView
OS/390
Parallel Sysplex
PR/SM
QMF
RACF
RAMAC
Redbooks
RETAIN
RMF
SecureWay
SQL/DS
System/390
Tivoli
TME
TotalStorage
VTAM
WebSphere
z/Architecture
z/OS
zSeries

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Glossary

The following terms and abbreviations are defined as they are used in the DB2 library.

A

abend. Abnormal end of task.

abend reason code. A 4-byte hexadecimal code that uniquely identifies a problem with DB2.

abnormal end of task (abend). Termination of a task, job, or subsystem because of an error condition that recovery facilities cannot resolve during execution.

access method services. The facility that is used to define and reproduce VSAM key-sequenced data sets.

access path. The path that is used to locate data that is specified in SQL statements. An access path can be indexed or sequential.

active log. The portion of the DB2 log to which log records are written as they are generated. The active log always contains the most recent log records, whereas the archive log holds those records that are older and no longer fit on the active log.

active member state. A state of a member of a data sharing group. The cross-system coupling facility identifies each active member with a group and associates the member with a particular task, address space, and z/OS system. A member that is not active has either a failed member state or a quiesced member state.

address space. A range of virtual storage pages that is identified by a number (ASID) and a collection of segment and page tables that map the virtual pages to real pages of the computer's memory.

address space connection. The result of connecting an allied address space to DB2. Each address space that contains a task that is connected to DB2 has exactly one address space connection, even though more than one task control block (TCB) can be present. See also *allied address space* and *task control block*.

| **address space identifier (ASID).** A unique
| system-assigned identifier for and address space.

administrative authority. A set of related privileges that DB2 defines. When you grant one of the administrative authorities to a person's ID, the person has all of the privileges that are associated with that administrative authority.

after trigger. A trigger that is defined with the trigger activation time AFTER.

agent. As used in DB2, the structure that associates all processes that are involved in a DB2 unit of work. An *allied agent* is generally synonymous with an *allied thread*. *System agents* are units of work that process tasks that are independent of the allied agent, such as prefetch processing, deferred writes, and service tasks.

aggregate function. An operation that derives its
result by using values from one or more rows. Contrast
with *scalar function*.

alias. An alternative name that can be used in SQL statements to refer to a table or view in the same or a remote DB2 subsystem.

allied address space. An area of storage that is external to DB2 and that is connected to DB2. An allied address space is capable of requesting DB2 services.

allied thread. A thread that originates at the local DB2 subsystem and that can access data at a remote DB2 subsystem.

allocated cursor. A cursor that is defined for stored procedure result sets by using the SQL ALLOCATE CURSOR statement.

already verified. An LU 6.2 security option that allows DB2 to provide the user's verified authorization ID when allocating a conversation. With this option, the user is not validated by the partner DB2 subsystem.

| **ambiguous cursor.** A database cursor that is in a plan
| or package that contains either PREPARE or EXECUTE
| IMMEDIATE SQL statements, and for which the
| following statements are true: the cursor is not defined
| with the FOR READ ONLY clause or the FOR UPDATE
| OF clause; the cursor is not defined on a read-only
| result table; the cursor is not the target of a WHERE
| CURRENT clause on an SQL UPDATE or DELETE
| statement.

American National Standards Institute (ANSI). An organization consisting of producers, consumers, and general interest groups, that establishes the procedures by which accredited organizations create and maintain voluntary industry standards in the United States.

ANSI. American National Standards Institute.

APAR. Authorized program analysis report.

APAR fix corrective service. A temporary correction of an IBM software defect. The correction is temporary,

© Copyright IBM Corp. 1982, 2008

because it is usually replaced at a later date by a more permanent correction, such as a program temporary fix (PTF).

APF. Authorized program facility.

API. Application programming interface.

APPL. A VTAM network definition statement that is used to define DB2 to VTAM as an application program that uses SNA LU 6.2 protocols.

application. A program or set of programs that performs a task; for example, a payroll application.

application-directed connection. A connection that an application manages using the SQL CONNECT statement.

application plan. The control structure that is produced during the bind process. DB2 uses the application plan to process SQL statements that it encounters during statement execution.

application process. The unit to which resources and locks are allocated. An application process involves the execution of one or more programs.

application programming interface (API). A functional interface that is supplied by the operating system or by a separately orderable licensed program that allows an application program that is written in a high-level language to use specific data or functions of the operating system or licensed program.

application requester. The component on a remote system that generates DRDA requests for data on behalf of an application. An application requester accesses a DB2 database server using the DRDA application-directed protocol.

application server. The target of a request from a remote application. In the DB2 environment, the application server function is provided by the distributed data facility and is used to access DB2 data from remote applications.

archive log. The portion of the DB2 log that contains log records that have been copied from the active log.

ASCII. An encoding scheme that is used to represent strings in many environments, typically on PCs and workstations. Contrast with *EBCDIC* and *Unicode*.

ASID. Address space identifier.

attachment facility. An interface between DB2 and TSO, IMS, CICS, or batch address spaces. An attachment facility allows application programs to access DB2.

attribute. A characteristic of an entity. For example, in database design, the phone number of an employee is one of that employee's attributes.

authorization ID. A string that can be verified for connection to DB2 and to which a set of privileges is allowed. It can represent an individual, an organizational group, or a function, but DB2 does not determine this representation.

authorized program analysis report (APAR). A report of a problem that is caused by a suspected defect in a current release of an IBM supplied program.

authorized program facility (APF). A facility that permits the identification of programs that are authorized to use restricted functions.

automatic query rewrite. A process that examines an SQL statement that refers to one or more base tables, and, if appropriate, rewrites the query so that it performs better. This process can also determine whether to rewrite a query so that it refers to one or more materialized query tables that are derived from the source tables.

auxiliary index. An index on an auxiliary table in which each index entry refers to a LOB.

auxiliary table. A table that stores columns outside the table in which they are defined. Contrast with *base table*.

B

backout. The process of undoing uncommitted changes that an application process made. This might be necessary in the event of a failure on the part of an application process, or as a result of a deadlock situation.

backward log recovery. The fourth and final phase of restart processing during which DB2 scans the log in a backward direction to apply UNDO log records for all aborted changes.

base table. (1) A table that is created by the SQL CREATE TABLE statement and that holds persistent data. Contrast with *result table* and *temporary table*.

(2) A table containing a LOB column definition. The actual LOB column data is not stored with the base table. The base table contains a row identifier for each row and an indicator column for each of its LOB columns. Contrast with *auxiliary table*.

base table space. A table space that contains base tables.

basic predicate. A predicate that compares two values.

basic sequential access method (BSAM). An access method for storing or retrieving data blocks in a continuous sequence, using either a sequential-access or a direct-access device.

batch message processing program. In IMS, an application program that can perform batch-type processing online and can access the IMS input and output message queues.

before trigger. A trigger that is defined with the trigger activation time BEFORE.

binary integer. A basic data type that can be further classified as small integer or large integer.

binary large object (BLOB). A sequence of bytes in which the size of the value ranges from 0 bytes to 2 GB–1. Such a string has a CCSID value of 65535.

binary string. A sequence of bytes that is not associated with a CCSID. For example, the BLOB data type is a binary string.

bind. The process by which the output from the SQL precompiler is converted to a usable control structure, often called an access plan, application plan, or package. During this process, access paths to the data are selected and some authorization checking is performed. The types of bind are:

automatic bind. (More correctly, *automatic rebind*) A process by which SQL statements are bound automatically (without a user issuing a BIND command) when an application process begins execution and the bound application plan or package it requires is not valid.

dynamic bind. A process by which SQL statements are bound as they are entered.

incremental bind. A process by which SQL statements are bound during the execution of an application process.

static bind. A process by which SQL statements are bound after they have been precompiled. All static SQL statements are prepared for execution at the same time.

bit data. Data that is character type CHAR or VARCHAR and has a CCSID value of 65535.

BLOB. Binary large object.

block fetch. A capability in which DB2 can retrieve, or fetch, a large set of rows together. Using block fetch can significantly reduce the number of messages that are being sent across the network. Block fetch applies only to cursors that do not update data.

BMP. Batch Message Processing (IMS). See *batch message processing program*.

bootstrap data set (BSDS). A VSAM data set that contains name and status information for DB2, as well as RBA range specifications, for all active and archive log data sets. It also contains passwords for the DB2 directory and catalog, and lists of conditional restart and checkpoint records.

BSAM. Basic sequential access method.

BSDS. Bootstrap data set.

buffer pool. Main storage that is reserved to satisfy the buffering requirements for one or more table spaces or indexes.

built-in data type. A data type that IBM supplies. Among the built-in data types for DB2 UDB for z/OS are string, numeric, ROWID, and datetime. Contrast with *distinct type*.

built-in function. A function that DB2 supplies. Contrast with *user-defined function*.

business dimension. A category of data, such as products or time periods, that an organization might want to analyze.

C

cache structure. A coupling facility structure that stores data that can be available to all members of a Sysplex. A DB2 data sharing group uses cache structures as group buffer pools.

CAF. Call attachment facility.

call attachment facility (CAF). A DB2 attachment facility for application programs that run in TSO or z/OS batch. The CAF is an alternative to the DSN command processor and provides greater control over the execution environment.

call-level interface (CLI). A callable application programming interface (API) for database access, which is an alternative to using embedded SQL. In contrast to embedded SQL, DB2 ODBC (which is based on the CLI architecture) does not require the user to precompile or bind applications, but instead provides a standard set of functions to process SQL statements and related services at run time.

cascade delete. The way in which DB2 enforces referential constraints when it deletes all descendent rows of a deleted parent row.

CASE expression. An expression that is selected based on the evaluation of one or more conditions.

cast function. A function that is used to convert instances of a (source) data type into instances of a different (target) data type. In general, a cast function has the name of the target data type. It has one single argument whose type is the source data type; its return type is the target data type.

castout. The DB2 process of writing changed pages from a group buffer pool to disk.

castout owner. The DB2 member that is responsible for casting out a particular page set or partition.

catalog. In DB2, a collection of tables that contains descriptions of objects such as tables, views, and indexes.

catalog table. Any table in the DB2 catalog.

CCSID. Coded character set identifier.

CDB. Communications database.

CDRA. Character Data Representation Architecture.

CEC. Central electronic complex. See *central processor complex*.

central electronic complex (CEC). See *central processor complex*.

central processor (CP). The part of the computer that contains the sequencing and processing facilities for instruction execution, initial program load, and other machine operations.

central processor complex (CPC). A physical collection of hardware (such as an ES/3090™) that consists of main storage, one or more central processors, timers, and channels.

CFRM. Coupling facility resource management.

CFRM policy. A declaration by a z/OS administrator regarding the allocation rules for a coupling facility structure.

character conversion. The process of changing characters from one encoding scheme to another.

Character Data Representation Architecture (CDRA). An architecture that is used to achieve consistent representation, processing, and interchange of string data.

character large object (CLOB). A sequence of bytes representing single-byte characters or a mixture of single- and double-byte characters where the size of the value can be up to 2 GB–1. In general, character large object values are used whenever a character string might exceed the limits of the VARCHAR type.

character set. A defined set of characters.

character string. A sequence of bytes that represent bit data, single-byte characters, or a mixture of single-byte and multibyte characters.

check constraint. A user-defined constraint that specifies the values that specific columns of a base table can contain.

check integrity. The condition that exists when each row in a table conforms to the check constraints that are defined on that table. Maintaining check integrity requires DB2 to enforce check constraints on operations that add or change data.

check pending. A state of a table space or partition that prevents its use by some utilities and by some SQL statements because of rows that violate referential constraints, check constraints, or both.

checkpoint. A point at which DB2 records internal status information on the DB2 log; the recovery process uses this information if DB2 abnormally terminates.

child lock. For explicit hierarchical locking, a lock that is held on either a table, page, row, or a large object (LOB). Each child lock has a parent lock. See also *parent lock*.

CI. Control interval.

CICS. Represents (in this publication): CICS Transaction Server for z/OS: Customer Information Control System Transaction Server for z/OS.

CICS attachment facility. A DB2 subcomponent that uses the z/OS subsystem interface (SSI) and cross-storage linkage to process requests from CICS to DB2 and to coordinate resource commitment.

CIDF. Control interval definition field.

claim. A notification to DB2 that an object is being accessed. Claims prevent drains from occurring until the claim is released, which usually occurs at a commit point. Contrast with *drain*.

claim class. A specific type of object access that can be one of the following isolation levels:
Cursor stability (CS)
Repeatable read (RR)
Write

claim count. A count of the number of agents that are accessing an object.

class of service. A VTAM term for a list of routes through a network, arranged in an order of preference for their use.

class word. A single word that indicates the nature of a data attribute. For example, the class word PROJ indicates that the attribute identifies a project.

clause. In SQL, a distinct part of a statement, such as a SELECT clause or a WHERE clause.

CLI. Call- level interface.

client. See *requester*.

CLIST. Command list. A language for performing TSO tasks.

CLOB. Character large object.

closed application. An application that requires exclusive use of certain statements on certain DB2

objects, so that the objects are managed solely through the application's external interface.

CLPA. Create link pack area.

clustering index. An index that determines how rows are physically ordered (*clustered*) in a table space. If a clustering index on a partitioned table is not a partitioning index, the rows are ordered in cluster sequence within each data partition instead of spanning partitions. Prior to Version 8 of DB2 UDB for z/OS, the partitioning index was required to be the clustering index.

coded character set. A set of unambiguous rules that establish a character set and the one-to-one relationships between the characters of the set and their coded representations.

coded character set identifier (CCSID). A 16-bit number that uniquely identifies a coded representation of graphic characters. It designates an encoding scheme identifier and one or more pairs consisting of a character set identifier and an associated code page identifier.

code page. (1) A set of assignments of characters to code points. In EBCDIC, for example, the character 'A' is assigned code point X'C1' (2) , and character 'B' is assigned code point X'C2'. Within a code page, each code point has only one specific meaning.

code point. In CDRA, a unique bit pattern that represents a character in a code page.

code unit. The fundamental binary width in a computer architecture that is used for representing character data, such as 7 bits, 8 bits, 16 bits, or 32 bits. Depending on the character encoding form that is used, each code point in a coded character set can be represented internally by one or more code units.

coexistence. During migration, the period of time in which two releases exist in the same data sharing group.

cold start. A process by which DB2 restarts without processing any log records. Contrast with *warm start*.

collection. A group of packages that have the same qualifier.

column. The vertical component of a table. A column has a name and a particular data type (for example, character, decimal, or integer).

column function. See *aggregate function*.

"come from" checking. An LU 6.2 security option that defines a list of authorization IDs that are allowed to connect to DB2 from a partner LU.

command. A DB2 operator command or a DSN subcommand. A command is distinct from an SQL statement.

command prefix. A one- to eight-character command identifier. The command prefix distinguishes the command as belonging to an application or subsystem rather than to MVS.

command recognition character (CRC). A character that permits a z/OS console operator or an IMS subsystem user to route DB2 commands to specific DB2 subsystems.

command scope. The scope of command operation in a data sharing group. If a command has *member scope*, the command displays information only from the one member or affects only non-shared resources that are owned locally by that member. If a command has *group scope*, the command displays information from all members, affects non-shared resources that are owned locally by all members, displays information on sharable resources, or affects sharable resources.

commit. The operation that ends a unit of work by releasing locks so that the database changes that are made by that unit of work can be perceived by other processes.

commit point. A point in time when data is considered consistent.

committed phase. The second phase of the multisite update process that requests all participants to commit the effects of the logical unit of work.

common service area (CSA). In z/OS, a part of the common area that contains data areas that are addressable by all address spaces.

communications database (CDB). A set of tables in the DB2 catalog that are used to establish conversations with remote database management systems.

comparison operator. A token (such as =, >, or <) that is used to specify a relationship between two values.

composite key. An ordered set of key columns of the same table.

compression dictionary. The dictionary that controls the process of compression and decompression. This dictionary is created from the data in the table space or table space partition.

concurrency. The shared use of resources by more than one application process at the same time.

conditional restart. A DB2 restart that is directed by a user-defined conditional restart control record (CRCR).

connection. In SNA, the existence of a communication path between two partner LUs that allows information

to be exchanged (for example, two DB2 subsystems that are connected and communicating by way of a conversation).

connection context. In SQLJ, a Java object that represents a connection to a data source.

connection declaration clause. In SQLJ, a statement that declares a connection to a data source.

connection handle. The data object containing information that is associated with a connection that DB2 ODBC manages. This includes general status information, transaction status, and diagnostic information.

connection ID. An identifier that is supplied by the attachment facility and that is associated with a specific address space connection.

consistency token. A timestamp that is used to generate the version identifier for an application. See also *version*.

constant. A language element that specifies an unchanging value. Constants are classified as string constants or numeric constants. Contrast with *variable*.

constraint. A rule that limits the values that can be inserted, deleted, or updated in a table. See *referential constraint*, *check constraint*, and *unique constraint*.

context. The application's logical connection to the data source and associated internal DB2 ODBC connection information that allows the application to direct its operations to a data source. A DB2 ODBC context represents a DB2 thread.

contracting conversion. A process that occurs when the length of a converted string is smaller than that of the source string. For example, this process occurs when an EBCDIC mixed-data string that contains DBCS characters is converted to ASCII mixed data; the converted string is shorter because of the removal of the shift codes.

control interval (CI). A fixed-length area or disk in which VSAM stores records and creates distributed free space. Also, in a key-sequenced data set or file, the set of records that an entry in the sequence-set index record points to. The control interval is the unit of information that VSAM transmits to or from disk. A control interval always includes an integral number of physical records.

control interval definition field (CIDF). In VSAM, a field that is located in the 4 bytes at the end of each control interval; it describes the free space, if any, in the control interval.

conversation. Communication, which is based on LU 6.2 or Advanced Program-to-Program Communication (APPC), between an application and a remote transaction program over an SNA logical unit-to-logical unit (LU-LU) session that allows communication while processing a transaction.

coordinator. The system component that coordinates the commit or rollback of a unit of work that includes work that is done on one or more other systems.

copy pool. A named set of SMS storage groups that contains data that is to be copied collectively. A copy pool is an SMS construct that lets you define which storage groups are to be copied by using FlashCopy functions. HSM determines which volumes belong to a copy pool.

copy target. A named set of SMS storage groups that are to be used as containers for copy pool volume copies. A copy target is an SMS construct that lets you define which storage groups are to be used as containers for volumes that are copied by using FlashCopy functions.

copy version. A point-in-time FlashCopy copy that is managed by HSM. Each copy pool has a version parameter that specifies how many copy versions are maintained on disk.

correlated columns. A relationship between the value of one column and the value of another column.

correlated subquery. A subquery (part of a WHERE or HAVING clause) that is applied to a row or group of rows of a table or view that is named in an outer subselect statement.

correlation ID. An identifier that is associated with a specific thread. In TSO, it is either an authorization ID or the job name.

correlation name. An identifier that designates a table, a view, or individual rows of a table or view within a single SQL statement. It can be defined in any FROM clause or in the first clause of an UPDATE or DELETE statement.

cost category. A category into which DB2 places cost estimates for SQL statements at the time the statement is bound. A cost estimate can be placed in either of the following cost categories:

- A: Indicates that DB2 had enough information to make a cost estimate without using default values.
- B: Indicates that some condition exists for which DB2 was forced to use default values for its estimate.

The cost category is externalized in the COST_CATEGORY column of the DSN_STATEMNT_TABLE when a statement is explained.

coupling facility. A special PR/SM LPAR logical partition that runs the coupling facility control program and provides high-speed caching, list processing, and locking functions in a Parallel Sysplex.

coupling facility resource management. A component of z/OS that provides the services to manage coupling facility resources in a Parallel Sysplex. This management includes the enforcement of CFRM policies to ensure that the coupling facility and structure requirements are satisfied.

CP. Central processor.

CPC. Central processor complex.

C++ member. A data object or function in a structure, union, or class.

C++ member function. An operator or function that is declared as a member of a class. A member function has access to the private and protected data members and to the member functions of objects in its class. Member functions are also called methods.

C++ object. (1) A region of storage. An object is created when a variable is defined or a new function is invoked. (2) An instance of a class.

CRC. Command recognition character.

CRCR. Conditional restart control record. See also *conditional restart*.

create link pack area (CLPA). An option that is used during IPL to initialize the link pack pageable area.

created temporary table. A table that holds temporary data and is defined with the SQL statement CREATE GLOBAL TEMPORARY TABLE. Information about created temporary tables is stored in the DB2 catalog, so this kind of table is persistent and can be shared across application processes. Contrast with *declared temporary table*. See also *temporary table*.

cross-memory linkage. A method for invoking a program in a different address space. The invocation is synchronous with respect to the caller.

cross-system coupling facility (XCF). A component of z/OS that provides functions to support cooperation between authorized programs that run within a Sysplex.

cross-system extended services (XES). A set of z/OS services that allow multiple instances of an application or subsystem, running on different systems in a Sysplex environment, to implement high-performance, high-availability data sharing by using a coupling facility.

CS. Cursor stability.

CSA. Common service area.

CT. Cursor table.

current data. Data within a host structure that is current with (identical to) the data within the base table.

current SQL ID. An ID that, at a single point in time, holds the privileges that are exercised when certain dynamic SQL statements run. The current SQL ID can be a primary authorization ID or a secondary authorization ID.

current status rebuild. The second phase of restart processing during which the status of the subsystem is reconstructed from information on the log.

cursor. A named control structure that an application program uses to point to a single row or multiple rows within some ordered set of rows of a result table. A cursor can be used to retrieve, update, or delete rows from a result table.

cursor sensitivity. The degree to which database updates are visible to the subsequent FETCH statements in a cursor. A cursor can be sensitive to changes that are made with positioned update and delete statements specifying the name of that cursor. A cursor can also be sensitive to changes that are made with searched update or delete statements, or with cursors other than this cursor. These changes can be made by this application process or by another application process.

cursor stability (CS). The isolation level that provides maximum concurrency without the ability to read uncommitted data. With cursor stability, a unit of work holds locks only on its uncommitted changes and on the current row of each of its cursors.

cursor table (CT). The copy of the skeleton cursor table that is used by an executing application process.

cycle. A set of tables that can be ordered so that each table is a descendent of the one before it, and the first table is a descendent of the last table. A self-referencing table is a cycle with a single member.

D

DAD. See *Document access definition.*

disk. A direct-access storage device that records data magnetically.

database. A collection of tables, or a collection of table spaces and index spaces.

database access thread. A thread that accesses data at the local subsystem on behalf of a remote subsystem.

database administrator (DBA). An individual who is responsible for designing, developing, operating, safeguarding, maintaining, and using a database.

database alias. The name of the target server if different from the location name. The database alias name is used to provide the name of the database server as it is known to the network. When a database alias name is defined, the location name is used by the application to reference the server, but the database alias name is used to identify the database server to be accessed. Any fully qualified object names within any SQL statements are not modified and are sent unchanged to the database server.

database descriptor (DBD). An internal representation of a DB2 database definition, which reflects the data definition that is in the DB2 catalog. The objects that are defined in a database descriptor are table spaces, tables, indexes, index spaces, relationships, check constraints, and triggers. A DBD also contains information about accessing tables in the database.

database exception status. An indication that something is wrong with a database. All members of a data sharing group must know and share the exception status of databases.

database identifier (DBID). An internal identifier of the database.

database management system (DBMS). A software system that controls the creation, organization, and modification of a database and the access to the data that is stored within it.

database request module (DBRM). A data set member that is created by the DB2 precompiler and that contains information about SQL statements. DBRMs are used in the bind process.

database server. The target of a request from a local application or an intermediate database server. In the DB2 environment, the database server function is provided by the distributed data facility to access DB2 data from local applications, or from a remote database server that acts as an intermediate database server.

data currency. The state in which data that is retrieved into a host variable in your program is a copy of data in the base table.

data definition name (ddname). The name of a data definition (DD) statement that corresponds to a data control block containing the same name.

data dictionary. A repository of information about an organization's application programs, databases, logical data models, users, and authorizations. A data dictionary can be manual or automated.

data-driven business rules. Constraints on particular data values that exist as a result of requirements of the business.

Data Language/I (DL/I). The IMS data manipulation language; a common high-level interface between a user application and IMS.

data mart. A small data warehouse that applies to a single department or team. See also *data warehouse*.

data mining. The process of collecting critical business information from a data warehouse, correlating it, and uncovering associations, patterns, and trends.

data partition. A VSAM data set that is contained within a partitioned table space.

data-partitioned secondary index (DPSI). A secondary index that is partitioned. The index is partitioned according to the underlying data.

data sharing. The ability of two or more DB2 subsystems to directly access and change a single set of data.

data sharing group. A collection of one or more DB2 subsystems that directly access and change the same data while maintaining data integrity.

data sharing member. A DB2 subsystem that is assigned by XCF services to a data sharing group.

data source. A local or remote relational or non-relational data manager that is capable of supporting data access via an ODBC driver that supports the ODBC APIs. In the case of DB2 UDB for z/OS, the data sources are always relational database managers.

data space. In releases prior to DB2 UDB for z/OS, Version 8, a range of up to 2 GB of contiguous virtual storage addresses that a program can directly manipulate. Unlike an address space, a data space can hold only data; it does not contain common areas, system data, or programs.

data type. An attribute of columns, literals, host variables, special registers, and the results of functions and expressions.

data warehouse. A system that provides critical business information to an organization. The data warehouse system cleanses the data for accuracy and currency, and then presents the data to decision makers so that they can interpret and use it effectively and efficiently.

date. A three-part value that designates a day, month, and year.

date duration. A decimal integer that represents a number of years, months, and days.

datetime value. A value of the data type DATE, TIME, or TIMESTAMP.

DBA. Database administrator.

DBCLOB. Double-byte character large object.

DBCS. Double-byte character set.

DBD. Database descriptor.

DBID. Database identifier.

DBMS. Database management system.

DBRM. Database request module.

DB2 catalog. Tables that are maintained by DB2 and contain descriptions of DB2 objects, such as tables, views, and indexes.

DB2 command. An instruction to the DB2 subsystem that a user enters to start or stop DB2, to display information on current users, to start or stop databases, to display information on the status of databases, and so on.

DB2 for VSE & VM. The IBM DB2 relational database management system for the VSE and VM operating systems.

DB2I. DB2 Interactive.

DB2 Interactive (DB2I). The DB2 facility that provides for the execution of SQL statements, DB2 (operator) commands, programmer commands, and utility invocation.

DB2I Kanji Feature. The tape that contains the panels and jobs that allow a site to display DB2I panels in Kanji.

DB2 PM. DB2 Performance Monitor.

DB2 thread. The DB2 structure that describes an application's connection, traces its progress, processes resource functions, and delimits its accessibility to DB2 resources and services.

DCLGEN. Declarations generator.

DDF. Distributed data facility.

ddname. Data definition name.

deadlock. Unresolvable contention for the use of a resource, such as a table or an index.

declarations generator (DCLGEN). A subcomponent of DB2 that generates SQL table declarations and COBOL, C, or PL/I data structure declarations that conform to the table. The declarations are generated from DB2 system catalog information. DCLGEN is also a DSN subcommand.

declared temporary table. A table that holds temporary data and is defined with the SQL statement DECLARE GLOBAL TEMPORARY TABLE. Information about declared temporary tables is not stored in the DB2 catalog, so this kind of table is not persistent and can be used only by the application process that issued the DECLARE statement. Contrast with *created temporary table*. See also *temporary table*.

default value. A predetermined value, attribute, or option that is assumed when no other is explicitly specified.

deferred embedded SQL. SQL statements that are neither fully static nor fully dynamic. Like static statements, they are embedded within an application, but like dynamic statements, they are prepared during the execution of the application.

deferred write. The process of asynchronously writing changed data pages to disk.

degree of parallelism. The number of concurrently executed operations that are initiated to process a query.

delete-connected. A table that is a dependent of table P or a dependent of a table to which delete operations from table P cascade.

delete hole. The location on which a cursor is positioned when a row in a result table is refetched and the row no longer exists on the base table, because another cursor deleted the row between the time the cursor first included the row in the result table and the time the cursor tried to refetch it.

delete rule. The rule that tells DB2 what to do to a dependent row when a parent row is deleted. For each relationship, the rule might be CASCADE, RESTRICT, SET NULL, or NO ACTION.

delete trigger. A trigger that is defined with the triggering SQL operation DELETE.

delimited identifier. A sequence of characters that are enclosed within double quotation marks ("). The sequence must consist of a letter followed by zero or more characters, each of which is a letter, digit, or the underscore character (_).

delimiter token. A string constant, a delimited identifier, an operator symbol, or any of the special characters that are shown in DB2 syntax diagrams.

denormalization. A key step in the task of building a physical relational database design. Denormalization is the intentional duplication of columns in multiple tables, and the consequence is increased data redundancy. Denormalization is sometimes necessary to minimize performance problems. Contrast with *normalization*.

dependent. An object (row, table, or table space) that has at least one parent. The object is also said to be a dependent (row, table, or table space) of its parent. See also *parent row*, *parent table*, *parent table space*.

dependent row. A row that contains a foreign key that matches the value of a primary key in the parent row.

dependent table. A table that is a dependent in at least one referential constraint.

DES-based authenticator. An authenticator that is generated using the DES algorithm.

descendent. An object that is a dependent of an object or is the dependent of a descendent of an object.

descendent row. A row that is dependent on another row, or a row that is a descendent of a dependent row.

descendent table. A table that is a dependent of another table, or a table that is a descendent of a dependent table.

deterministic function. A user-defined function whose result is dependent on the values of the input arguments. That is, successive invocations with the same input values produce the same answer. Sometimes referred to as a *not-variant* function. Contrast this with an *nondeterministic function* (sometimes called a *variant function*), which might not always produce the same result for the same inputs.

DFP. Data Facility Product (in z/OS).

DFSMS. Data Facility Storage Management Subsystem (in z/OS). Also called *Storage Management Subsystem (SMS)*.

| **DFSMSdss.** The data set services (dss) component of
| DFSMS (in z/OS).

| **DFSMShsm.** The hierarchical storage manager (hsm)
| component of DFSMS (in z/OS).

dimension. A data category such as time, products, or markets. The elements of a dimension are referred to as members. Dimensions offer a very concise, intuitive way of organizing and selecting data for retrieval, exploration, and analysis. See also *dimension table*.

dimension table. The representation of a dimension in a star schema. Each row in a dimension table represents all of the attributes for a particular member of the dimension. See also *dimension*, *star schema*, and *star join*.

directory. The DB2 system database that contains internal objects such as database descriptors and skeleton cursor tables.

distinct predicate. In SQL, a predicate that ensures
that two row values are not equal, and that both row
values are not null.

distinct type. A user-defined data type that is internally represented as an existing type (its source type), but is considered to be a separate and incompatible type for semantic purposes.

distributed data. Data that resides on a DBMS other than the local system.

distributed data facility (DDF). A set of DB2 components through which DB2 communicates with another relational database management system.

Distributed Relational Database Architecture (DRDA). A connection protocol for distributed relational database processing that is used by IBM's relational database products. DRDA includes protocols for communication between an application and a remote relational database management system, and for communication between relational database management systems. See also *DRDA access.*

DL/I. Data Language/I.

DNS. Domain name server.

| **document access definition (DAD).** Used to define
| the indexing scheme for an XML column or the
| mapping scheme of an XML collection. It can be used
| to enable an XML Extender column of an XML
| collection, which is XML formatted.

domain. The set of valid values for an attribute.

domain name. The name by which TCP/IP applications refer to a TCP/IP host within a TCP/IP network.

domain name server (DNS). A special TCP/IP network server that manages a distributed directory that is used to map TCP/IP host names to IP addresses.

double-byte character large object (DBCLOB). A sequence of bytes representing double-byte characters where the size of the values can be up to 2 GB. In general, DBCLOB values are used whenever a double-byte character string might exceed the limits of the VARGRAPHIC type.

double-byte character set (DBCS). A set of characters, which are used by national languages such as Japanese and Chinese, that have more symbols than can be represented by a single byte. Each character is 2 bytes in length. Contrast with *single-byte character set* and *multibyte character set*.

double-precision floating point number. A 64-bit approximate representation of a real number.

downstream. The set of nodes in the syncpoint tree that is connected to the local DBMS as a participant in the execution of a two-phase commit.

| **DPSI.** Data-partitioned secondary index.

drain. The act of acquiring a locked resource by quiescing access to that object.

drain lock. A lock on a claim class that prevents a claim from occurring.

DRDA. Distributed Relational Database Architecture.

DRDA access. An open method of accessing distributed data that you can use to can connect to another database server to execute packages that were previously bound at the server location. You use the SQL CONNECT statement or an SQL statement with a three-part name to identify the server. Contrast with *private protocol access*.

DSN. (1) The default DB2 subsystem name. (2) The name of the TSO command processor of DB2. (3) The first three characters of DB2 module and macro names.

duration. A number that represents an interval of time. See also *date duration*, *labeled duration*, and *time duration*.

dynamic cursor. A named control structure that an application program uses to change the size of the result table and the order of its rows after the cursor is opened. Contrast with *static cursor.*

dynamic dump. A dump that is issued during the execution of a program, usually under the control of that program.

dynamic SQL. SQL statements that are prepared and executed within an application program while the program is executing. In dynamic SQL, the SQL source is contained in host language variables rather than being coded into the application program. The SQL statement can change several times during the application program's execution.

dynamic statement cache pool. A cache, located above the 2-GB storage line, that holds dynamic statements.

E

EA-enabled table space. A table space or index space that is enabled for extended addressability and that contains individual partitions (or pieces, for LOB table spaces) that are greater than 4 GB.

EB. See *exabyte*.

EBCDIC. Extended binary coded decimal interchange code. An encoding scheme that is used to represent character data in the z/OS, VM, VSE, and iSeries™ environments. Contrast with *ASCII* and *Unicode*.

e-business. The transformation of key business processes through the use of Internet technologies.

EDM pool. A pool of main storage that is used for database descriptors, application plans, authorization cache, application packages.

EID. Event identifier.

embedded SQL. SQL statements that are coded within an application program. See *static SQL*.

enclave. In Language Environment , an independent collection of routines, one of which is designated as the main routine. An enclave is similar to a program or run unit.

encoding scheme. A set of rules to represent character data (ASCII, EBCDIC, or Unicode).

entity. A significant object of interest to an organization.

enumerated list. A set of DB2 objects that are defined with a LISTDEF utility control statement in which pattern-matching characters (*, %, _ or ?) are not used.

environment. A collection of names of logical and physical resources that are used to support the performance of a function.

environment handle. In DB2 ODBC, the data object that contains global information regarding the state of the application. An environment handle must be allocated before a connection handle can be allocated. Only one environment handle can be allocated per application.

EOM. End of memory.

EOT. End of task.

equijoin. A join operation in which the join-condition has the form *expression* = *expression*.

error page range. A range of pages that are considered to be physically damaged. DB2 does not allow users to access any pages that fall within this range.

escape character. The symbol that is used to enclose an SQL delimited identifier. The escape character is the double quotation mark ("), except in COBOL applications, where the user assigns the symbol, which is either a double quotation mark or an apostrophe (').

ESDS. Entry sequenced data set.

ESMT. External subsystem module table (in IMS).

EUR. IBM European Standards.

exabyte. For processor, real and virtual storage capacities and channel volume:
1 152 921 504 606 846 976 bytes or 2^{60}.

exception table. A table that holds rows that violate referential constraints or check constraints that the CHECK DATA utility finds.

exclusive lock. A lock that prevents concurrently executing application processes from reading or changing data. Contrast with *share lock*.

executable statement. An SQL statement that can be embedded in an application program, dynamically prepared and executed, or issued interactively.

execution context. In SQLJ, a Java object that can be used to control the execution of SQL statements.

exit routine. A user-written (or IBM-provided default) program that receives control from DB2 to perform specific functions. Exit routines run as extensions of DB2.

expanding conversion. A process that occurs when the length of a converted string is greater than that of the source string. For example, this process occurs when an ASCII mixed-data string that contains DBCS characters is converted to an EBCDIC mixed-data string; the converted string is longer because of the addition of shift codes.

explicit hierarchical locking. Locking that is used to make the parent-child relationship between resources known to IRLM. This kind of locking avoids global locking overhead when no inter-DB2 interest exists on a resource.

exposed name. A correlation name or a table or view name for which a correlation name is not specified. Names that are specified in a FROM clause are exposed or non-exposed.

expression. An operand or a collection of operators and operands that yields a single value.

extended recovery facility (XRF). A facility that minimizes the effect of failures in z/OS, VTAM , the host processor, or high-availability applications during sessions between high-availability applications and designated terminals. This facility provides an alternative subsystem to take over sessions from the failing subsystem.

Extensible Markup Language (XML). A standard metalanguage for defining markup languages that is a subset of Standardized General Markup Language (SGML). The less complex nature of XML makes it easier to write applications that handle document types, to author and manage structured information, and to transmit and share structured information across diverse computing environments.

external function. A function for which the body is written in a programming language that takes scalar argument values and produces a scalar result for each invocation. Contrast with *sourced function*, *built-in function*, and *SQL function*.

external procedure. A user-written application program that can be invoked with the SQL CALL statement, which is written in a programming language. Contrast with *SQL procedure*.

external routine. A user-defined function or stored procedure that is based on code that is written in an external programming language.

external subsystem module table (ESMT). In IMS, the table that specifies which attachment modules must be loaded.

F

failed member state. A state of a member of a data sharing group. When a member fails, the XCF permanently records the failed member state. This state usually means that the member's task, address space, or z/OS system terminated before the state changed from active to quiesced.

fallback. The process of returning to a previous release of DB2 after attempting or completing migration to a current release.

false global lock contention. A contention indication from the coupling facility when multiple lock names are hashed to the same indicator and when no real contention exists.

fan set. A direct physical access path to data, which is provided by an index, hash, or link; a fan set is the means by which the data manager supports the ordering of data.

federated database. The combination of a DB2 Universal Database server (in Linux, UNIX, and Windows environments) and multiple data sources to which the server sends queries. In a federated database system, a client application can use a single SQL statement to join data that is distributed across multiple database management systems and can view the data as if it were local.

fetch orientation. The specification of the desired placement of the cursor as part of a FETCH statement (for example, BEFORE, AFTER, NEXT, PRIOR, CURRENT, FIRST, LAST, ABSOLUTE, and RELATIVE).

field procedure. A user-written exit routine that is designed to receive a single value and transform (encode or decode) it in any way the user can specify.

filter factor. A number between zero and one that estimates the proportion of rows in a table for which a predicate is true.

fixed-length string. A character or graphic string whose length is specified and cannot be changed. Contrast with *varying-length string*.

FlashCopy. A function on the IBM Enterprise Storage Server that can create a point-in-time copy of data while an application is running.

foreign key. A column or set of columns in a dependent table of a constraint relationship. The key must have the same number of columns, with the same descriptions, as the primary key of the parent table.

Each foreign key value must either match a parent key value in the related parent table or be null.

forest. An ordered set of subtrees of XML nodes.

forget. In a two-phase commit operation, (1) the vote that is sent to the prepare phase when the participant has not modified any data. The forget vote allows a participant to release locks and forget about the logical unit of work. This is also referred to as the read-only vote. (2) The response to the *committed* request in the second phase of the operation.

forward log recovery. The third phase of restart processing during which DB2 processes the log in a forward direction to apply all REDO log records.

free space. The total amount of unused space in a page; that is, the space that is not used to store records or control information is free space.

full outer join. The result of a join operation that includes the matched rows of both tables that are being joined and preserves the unmatched rows of both tables. See also *join*.

fullselect. A subselect, a values-clause, or a number of both that are combined by set operators. *Fullselect* specifies a result table. If UNION is not used, the result of the fullselect is the result of the specified subselect.

fully escaped mapping. A mapping from an SQL identifier to an XML name when the SQL identifier is a column name.

function. A mapping, which is embodied as a program (the function body) that is invocable by means of zero or more input values (arguments) to a single value (the result). See also *aggregate function* and *scalar function*.

Functions can be user-defined, built-in, or generated by DB2. (See also *built-in function*, *cast function*, *external function*, *sourced function*, *SQL function*, and *user-defined function*.)

function definer. The authorization ID of the owner of the schema of the function that is specified in the CREATE FUNCTION statement.

function implementer. The authorization ID of the owner of the function program and function package.

function package. A package that results from binding the DBRM for a function program.

function package owner. The authorization ID of the user who binds the function program's DBRM into a function package.

function resolution. The process, internal to the DBMS, by which a function invocation is bound to a particular function instance. This process uses the function name, the data types of the arguments, and a list of the applicable schema names (called the *SQL path*) to make the selection. This process is sometimes called *function selection*.

function selection. See *function resolution*.

function signature. The logical concatenation of a fully qualified function name with the data types of all of its parameters.

G

GB. Gigabyte (1 073 741 824 bytes).

GBP. Group buffer pool.

GBP-dependent. The status of a page set or page set partition that is dependent on the group buffer pool. Either read/write interest is active among DB2 subsystems for this page set, or the page set has changed pages in the group buffer pool that have not yet been cast out to disk.

generalized trace facility (GTF). A z/OS service program that records significant system events such as I/O interrupts, SVC interrupts, program interrupts, or external interrupts.

generic resource name. A name that VTAM uses to represent several application programs that provide the same function in order to handle session distribution and balancing in a Sysplex environment.

getpage. An operation in which DB2 accesses a data page.

global lock. A lock that provides concurrency control within and among DB2 subsystems. The scope of the lock is across all DB2 subsystems of a data sharing group.

global lock contention. Conflicts on locking requests between different DB2 members of a data sharing group when those members are trying to serialize shared resources.

governor. See *resource limit facility*.

graphic string. A sequence of DBCS characters.

gross lock. The *shared*, *update*, or *exclusive* mode locks on a table, partition, or table space.

group buffer pool (GBP). A coupling facility cache structure that is used by a data sharing group to cache data and to ensure that the data is consistent for all members.

group buffer pool duplexing. The ability to write data to two instances of a group buffer pool structure: a *primary group buffer pool* and a *secondary group buffer*

pool. z/OS publications refer to these instances as the "old" (for primary) and "new" (for secondary) structures.

group level. The release level of a data sharing group, which is established when the first member migrates to a new release.

group name. The z/OS XCF identifier for a data sharing group.

group restart. A restart of at least one member of a data sharing group after the loss of either locks or the shared communications area.

GTF. Generalized trace facility.

H

handle. In DB2 ODBC, a variable that refers to a data structure and associated resources. See also *statement handle*, *connection handle*, and *environment handle*.

help panel. A screen of information that presents tutorial text to assist a user at the workstation or terminal.

heuristic damage. The inconsistency in data between one or more participants that results when a heuristic decision to resolve an indoubt LUW at one or more participants differs from the decision that is recorded at the coordinator.

heuristic decision. A decision that forces indoubt resolution at a participant by means other than automatic resynchronization between coordinator and participant.

hole. A row of the result table that cannot be accessed because of a delete or an update that has been performed on the row. See also *delete hole* and *update hole*.

home address space. The area of storage that z/OS currently recognizes as *dispatched*.

host. The set of programs and resources that are available on a given TCP/IP instance.

host expression. A Java variable or expression that is referenced by SQL clauses in an SQLJ application program.

host identifier. A name that is declared in the host program.

host language. A programming language in which you can embed SQL statements.

host program. An application program that is written in a host language and that contains embedded SQL statements.

host structure. In an application program, a structure that is referenced by embedded SQL statements.

host variable. In an application program, an application variable that is referenced by embedded SQL statements.

host variable array. An array of elements, each of which corresponds to a value for a column. The dimension of the array determines the maximum number of rows for which the array can be used.

HSM. Hierarchical storage manager.

HTML. Hypertext Markup Language, a standard method for presenting Web data to users.

HTTP. Hypertext Transfer Protocol, a communication protocol that the Web uses.

I

ICF. Integrated catalog facility.

IDCAMS. An IBM program that is used to process access method services commands. It can be invoked as a job or jobstep, from a TSO terminal, or from within a user's application program.

IDCAMS LISTCAT. A facility for obtaining information that is contained in the access method services catalog.

identify. A request that an attachment service program in an address space that is separate from DB2 issues thorough the z/OS subsystem interface to inform DB2 of its existence and to initiate the process of becoming connected to DB2.

identity column. A column that provides a way for DB2 to automatically generate a numeric value for each row. The generated values are unique if cycling is not used. Identity columns are defined with the AS IDENTITY clause. Uniqueness of values can be ensured by defining a unique index that contains only the identity column. A table can have no more than one identity column.

IFCID. Instrumentation facility component identifier.

IFI. Instrumentation facility interface.

IFI call. An invocation of the instrumentation facility interface (IFI) by means of one of its defined functions.

IFP. IMS Fast Path.

image copy. An exact reproduction of all or part of a table space. DB2 provides utility programs to make full image copies (to copy the entire table space) or incremental image copies (to copy only those pages that have been modified since the last image copy).

implied forget. In the presumed-abort protocol, an implied response of *forget* to the second-phase *committed* request from the coordinator. The response is implied when the participant responds to any subsequent request from the coordinator.

IMS. Information Management System.

IMS attachment facility. A DB2 subcomponent that uses z/OS subsystem interface (SSI) protocols and cross-memory linkage to process requests from IMS to DB2 and to coordinate resource commitment.

IMS DB. Information Management System Database.

IMS TM. Information Management System Transaction Manager.

in-abort. A status of a unit of recovery. If DB2 fails after a unit of recovery begins to be rolled back, but before the process is completed, DB2 continues to back out the changes during restart.

in-commit. A status of a unit of recovery. If DB2 fails after beginning its phase 2 commit processing, it "knows," when restarted, that changes made to data are consistent. Such units of recovery are termed *in-commit*.

independent. An object (row, table, or table space) that is neither a parent nor a dependent of another object.

index. A set of pointers that are logically ordered by the values of a key. Indexes can provide faster access to data and can enforce uniqueness on the rows in a table.

index-controlled partitioning. A type of partitioning in which partition boundaries for a partitioned table are controlled by values that are specified on the CREATE INDEX statement. Partition limits are saved in the LIMITKEY column of the SYSIBM.SYSINDEXPART catalog table.

index key. The set of columns in a table that is used to determine the order of index entries.

index partition. A VSAM data set that is contained within a partitioning index space.

index space. A page set that is used to store the entries of one index.

indicator column. A 4-byte value that is stored in a base table in place of a LOB column.

indicator variable. A variable that is used to represent the null value in an application program. If the value for the selected column is null, a negative value is placed in the indicator variable.

indoubt. A status of a unit of recovery. If DB2 fails after it has finished its phase 1 commit processing and before it has started phase 2, only the commit coordinator knows if an individual unit of recovery is to be committed or rolled back. At emergency restart, if DB2 lacks the information it needs to make this decision, the status of the unit of recovery is *indoubt* until DB2 obtains this information from the coordinator. More than one unit of recovery can be indoubt at restart.

indoubt resolution. The process of resolving the status of an indoubt logical unit of work to either the committed or the rollback state.

inflight. A status of a unit of recovery. If DB2 fails before its unit of recovery completes phase 1 of the commit process, it merely backs out the updates of its unit of recovery at restart. These units of recovery are termed *inflight*.

inheritance. The passing downstream of class resources or attributes from a parent class in the class hierarchy to a child class.

initialization file. For DB2 ODBC applications, a file containing values that can be set to adjust the performance of the database manager.

inline copy. A copy that is produced by the LOAD or REORG utility. The data set that the inline copy produces is logically equivalent to a full image copy that is produced by running the COPY utility with read-only access (SHRLEVEL REFERENCE).

inner join. The result of a join operation that includes only the matched rows of both tables that are being joined. See also *join*.

inoperative package. A package that cannot be used because one or more user-defined functions or procedures that the package depends on were dropped. Such a package must be explicitly rebound. Contrast with *invalid package.*

insensitive cursor. A cursor that is not sensitive to inserts, updates, or deletes that are made to the underlying rows of a result table after the result table has been materialized.

insert trigger. A trigger that is defined with the triggering SQL operation INSERT.

install. The process of preparing a DB2 subsystem to operate as a z/OS subsystem.

installation verification scenario. A sequence of operations that exercises the main DB2 functions and tests whether DB2 was correctly installed.

instrumentation facility component identifier (IFCID). A value that names and identifies a trace record of an event that can be traced. As a parameter on the START TRACE and MODIFY TRACE commands, it specifies that the corresponding event is to be traced.

instrumentation facility interface (IFI). A programming interface that enables programs to obtain online trace data about DB2, to submit DB2 commands, and to pass data to DB2.

Interactive System Productivity Facility (ISPF). An IBM licensed program that provides interactive dialog services in a z/OS environment.

inter-DB2 R/W interest. A property of data in a table space, index, or partition that has been opened by more than one member of a data sharing group and that has been opened for writing by at least one of those members.

intermediate database server. The target of a request from a local application or a remote application requester that is forwarded to another database server. In the DB2 environment, the remote request is forwarded transparently to another database server if the object that is referenced by a three-part name does not reference the local location.

internationalization. The support for an encoding scheme that is able to represent the code points of characters from many different geographies and languages. To support all geographies, the Unicode standard requires more than 1 byte to represent a single character. See also *Unicode*.

internal resource lock manager (IRLM). A z/OS subsystem that DB2 uses to control communication and database locking.

International Organization for Standardization. An international body charged with creating standards to facilitate the exchange of goods and services as well as cooperation in intellectual, scientific, technological, and economic activity.

invalid package. A package that depends on an object (other than a user-defined function) that is dropped. Such a package is implicitly rebound on invocation. Contrast with *inoperative package.*

invariant character set. (1) A character set, such as the syntactic character set, whose code point assignments do not change from code page to code page. (2) A minimum set of characters that is available as part of all character sets.

IP address. A 4-byte value that uniquely identifies a TCP/IP host.

IRLM. Internal resource lock manager.

ISO. International Organization for Standardization.

isolation level. The degree to which a unit of work is isolated from the updating operations of other units of work. See also *cursor stability*, *read stability*, *repeatable read*, and *uncommitted read*.

ISPF. Interactive System Productivity Facility.

ISPF/PDF. Interactive System Productivity Facility/Program Development Facility.

iterator. In SQLJ, an object that contains the result set of a query. An iterator is equivalent to a cursor in other host languages.

iterator declaration clause. In SQLJ, a statement that generates an iterator declaration class. An iterator is an object of an iterator declaration class.

J

Japanese Industrial Standard. An encoding scheme that is used to process Japanese characters.

JAR. Java Archive.

Java Archive (JAR). A file format that is used for aggregating many files into a single file.

JCL. Job control language.

JDBC. A Sun Microsystems database application programming interface (API) for Java that allows programs to access database management systems by using callable SQL. JDBC does not require the use of an SQL preprocessor. In addition, JDBC provides an architecture that lets users add modules called *database drivers*, which link the application to their choice of database management systems at run time.

JES. Job Entry Subsystem.

JIS. Japanese Industrial Standard.

job control language (JCL). A control language that is used to identify a job to an operating system and to describe the job's requirements.

Job Entry Subsystem (JES). An IBM licensed program that receives jobs into the system and processes all output data that is produced by the jobs.

join. A relational operation that allows retrieval of data from two or more tables based on matching column values. See also *equijoin, full outer join, inner join, left outer join, outer join, and right outer join*.

K

KB. Kilobyte (1024 bytes).

Kerberos. A network authentication protocol that is designed to provide strong authentication for client/server applications by using secret-key cryptography.

Kerberos ticket. A transparent application mechanism that transmits the identity of an initiating principal to its target. A simple ticket contains the principal's

identity, a session key, a timestamp, and other information, which is sealed using the target's secret key.

key. A column or an ordered collection of columns that is identified in the description of a table, index, or referential constraint. The same column can be part of more than one key.

key-sequenced data set (KSDS). A VSAM file or data set whose records are loaded in key sequence and controlled by an index.

keyword. In SQL, a name that identifies an option that is used in an SQL statement.

KSDS. Key-sequenced data set.

L

labeled duration. A number that represents a duration of years, months, days, hours, minutes, seconds, or microseconds.

large object (LOB). A sequence of bytes representing bit data, single-byte characters, double-byte characters, or a mixture of single- and double-byte characters. A LOB can be up to 2 GB–1 byte in length. See also *BLOB*, *CLOB*, and *DBCLOB*.

last agent optimization. An optimized commit flow for either presumed-nothing or presumed-abort protocols in which the last agent, or final participant, becomes the commit coordinator. This flow saves at least one message.

latch. A DB2 internal mechanism for controlling concurrent events or the use of system resources.

LCID. Log control interval definition.

LDS. Linear data set.

leaf page. A page that contains pairs of keys and RIDs and that points to actual data. Contrast with *nonleaf page*.

left outer join. The result of a join operation that includes the matched rows of both tables that are being joined, and that preserves the unmatched rows of the first table. See also *join*.

limit key. The highest value of the index key for a partition.

linear data set (LDS). A VSAM data set that contains data but no control information. A linear data set can be accessed as a byte-addressable string in virtual storage.

linkage editor. A computer program for creating load modules from one or more object modules or load modules by resolving cross references among the modules and, if necessary, adjusting addresses.

link-edit. The action of creating a loadable computer program using a linkage editor.

list. A type of object, which DB2 utilities can process, that identifies multiple table spaces, multiple index spaces, or both. A list is defined with the LISTDEF utility control statement.

list structure. A coupling facility structure that lets data be shared and manipulated as elements of a queue.

LLE. Load list element.

L-lock. Logical lock.

load list element. A z/OS control block that controls the loading and deleting of a particular load module based on entry point names.

load module. A program unit that is suitable for loading into main storage for execution. The output of a linkage editor.

LOB. Large object.

LOB locator. A mechanism that allows an application program to manipulate a large object value in the database system. A LOB locator is a fullword integer value that represents a single LOB value. An application program retrieves a LOB locator into a host variable and can then apply SQL operations to the associated LOB value using the locator.

LOB lock. A lock on a LOB value.

LOB table space. A table space in an auxiliary table that contains all the data for a particular LOB column in the related base table.

local. A way of referring to any object that the local DB2 subsystem maintains. A *local table*, for example, is a table that is maintained by the local DB2 subsystem. Contrast with *remote*.

locale. The definition of a subset of a user's environment that combines a CCSID and characters that are defined for a specific language and country.

local lock. A lock that provides intra-DB2 concurrency control, but not inter-DB2 concurrency control; that is, its scope is a single DB2.

local subsystem. The unique relational DBMS to which the user or application program is directly connected (in the case of DB2, by one of the DB2 attachment facilities).

location. The unique name of a database server. An application uses the location name to access a DB2

database server. A database alias can be used to override the location name when accessing a remote server.

location alias. Another name by which a database server identifies itself in the network. Applications can use this name to access a DB2 database server.

lock. A means of controlling concurrent events or access to data. DB2 locking is performed by the IRLM.

lock duration. The interval over which a DB2 lock is held.

lock escalation. The promotion of a lock from a row, page, or LOB lock to a table space lock because the number of page locks that are concurrently held on a given resource exceeds a preset limit.

locking. The process by which the integrity of data is ensured. Locking prevents concurrent users from accessing inconsistent data.

lock mode. A representation for the type of access that concurrently running programs can have to a resource that a DB2 lock is holding.

lock object. The resource that is controlled by a DB2 lock.

lock promotion. The process of changing the size or mode of a DB2 lock to a higher, more restrictive level.

lock size. The amount of data that is controlled by a DB2 lock on table data; the value can be a row, a page, a LOB, a partition, a table, or a table space.

lock structure. A coupling facility data structure that is composed of a series of lock entries to support shared and exclusive locking for logical resources.

log. A collection of records that describe the events that occur during DB2 execution and that indicate their sequence. The information thus recorded is used for recovery in the event of a failure during DB2 execution.

log control interval definition. A suffix of the physical log record that tells how record segments are placed in the physical control interval.

logical claim. A claim on a logical partition of a nonpartitioning index.

logical data modeling. The process of documenting the comprehensive business information requirements in an accurate and consistent format. Data modeling is the first task of designing a database.

logical drain. A drain on a logical partition of a nonpartitioning index.

logical index partition. The set of all keys that reference the same data partition.

logical lock (L-lock). The lock type that transactions use to control intra- and inter-DB2 data concurrency between transactions. Contrast with *physical lock (P-lock)*.

logically complete. A state in which the concurrent copy process is finished with the initialization of the target objects that are being copied. The target objects are available for update.

logical page list (LPL). A list of pages that are in error and that cannot be referenced by applications until the pages are recovered. The page is in *logical error* because the actual media (coupling facility or disk) might not contain any errors. Usually a connection to the media has been lost.

logical partition. A set of key or RID pairs in a nonpartitioning index that are associated with a particular partition.

logical recovery pending (LRECP). The state in which the data and the index keys that reference the data are inconsistent.

logical unit (LU). An access point through which an application program accesses the SNA network in order to communicate with another application program.

logical unit of work (LUW). The processing that a program performs between synchronization points.

logical unit of work identifier (LUWID). A name that uniquely identifies a thread within a network. This name consists of a fully-qualified LU network name, an LUW instance number, and an LUW sequence number.

log initialization. The first phase of restart processing during which DB2 attempts to locate the current end of the log.

log record header (LRH). A prefix, in every logical record, that contains control information.

log record sequence number (LRSN). A unique identifier for a log record that is associated with a data sharing member. DB2 uses the LRSN for recovery in the data sharing environment.

log truncation. A process by which an explicit starting RBA is established. This RBA is the point at which the next byte of log data is to be written.

LPL. Logical page list.

LRECP. Logical recovery pending.

LRH. Log record header.

LRSN. Log record sequence number.

LU. Logical unit.

LU name. Logical unit name, which is the name by which VTAM refers to a node in a network. Contrast with *location name*.

LUW. Logical unit of work.

LUWID. Logical unit of work identifier.

M

mapping table. A table that the REORG utility uses to map the associations of the RIDs of data records in the original copy and in the shadow copy. This table is created by the user.

mass delete. The deletion of all rows of a table.

master terminal. The IMS logical terminal that has complete control of IMS resources during online operations.

master terminal operator (MTO). See *master terminal.*

materialize. (1) The process of putting rows from a view or nested table expression into a work file for additional processing by a query.

(2) The placement of a LOB value into contiguous storage. Because LOB values can be very large, DB2 avoids materializing LOB data until doing so becomes absolutely necessary.

materialized query table. A table that is used to contain information that is derived and can be summarized from one or more source tables.

MB. Megabyte (1 048 576 bytes).

MBCS. Multibyte character set. UTF-8 is an example of an MBCS. Characters in UTF-8 can range from 1 to 4 bytes in DB2.

member name. The z/OS XCF identifier for a particular DB2 subsystem in a data sharing group.

menu. A displayed list of available functions for selection by the operator. A menu is sometimes called a *menu panel*.

metalanguage. A language that is used to create other specialized languages.

migration. The process of converting a subsystem with a previous release of DB2 to an updated or current release. In this process, you can acquire the functions of the updated or current release without losing the data that you created on the previous release.

mixed data string. A character string that can contain both single-byte and double-byte characters.

MLPA. Modified link pack area.

MODEENT. A VTAM macro instruction that associates a logon mode name with a set of parameters representing session protocols. A set of MODEENT macro instructions defines a logon mode table.

modeling database. A DB2 database that you create on your workstation that you use to model a DB2 UDB for z/OS subsystem, which can then be evaluated by the Index Advisor.

mode name. A VTAM name for the collection of physical and logical characteristics and attributes of a session.

modify locks. An L-lock or P-lock with a MODIFY attribute. A list of these active locks is kept at all times in the coupling facility lock structure. If the requesting DB2 subsystem fails, that DB2 subsystem's modify locks are converted to retained locks.

MPP. Message processing program (in IMS).

MTO. Master terminal operator.

multibyte character set (MBCS). A character set that represents single characters with more than a single byte. Contrast with *single-byte character set* and *double-byte character set*. See also *Unicode*.

multidimensional analysis. The process of assessing and evaluating an enterprise on more than one level.

Multiple Virtual Storage. An element of the z/OS operating system. This element is also called the Base Control Program (BCP).

multisite update. Distributed relational database processing in which data is updated in more than one location within a single unit of work.

multithreading. Multiple TCBs that are executing one copy of DB2 ODBC code concurrently (sharing a processor) or in parallel (on separate central processors).

must-complete. A state during DB2 processing in which the entire operation must be completed to maintain data integrity.

mutex. Pthread mutual exclusion; a lock. A Pthread mutex variable is used as a locking mechanism to allow serialization of critical sections of code by temporarily blocking the execution of all but one thread.

MVS. See *Multiple Virtual Storage*.

N

negotiable lock. A lock whose mode can be downgraded, by agreement among contending users, to be compatible to all. A physical lock is an example of a negotiable lock.

nested table expression. A fullselect in a FROM clause (surrounded by parentheses).

network identifier (NID). The network ID that is assigned by IMS or CICS, or if the connection type is RRSAF, the RRS unit of recovery ID (URID).

NID. Network identifier.

nonleaf page. A page that contains keys and page numbers of other pages in the index (either leaf or nonleaf pages). Nonleaf pages never point to actual data.

nonpartitioned index. An index that is not physically partitioned. Both partitioning indexes and secondary indexes can be nonpartitioned.

nonscrollable cursor. A cursor that can be moved only in a forward direction. Nonscrollable cursors are sometimes called forward-only cursors or serial cursors.

normalization. A key step in the task of building a logical relational database design. Normalization helps you avoid redundancies and inconsistencies in your data. An entity is normalized if it meets a set of constraints for a particular normal form (first normal form, second normal form, and so on). Contrast with *denormalization*.

nondeterministic function. A user-defined function whose result is not solely dependent on the values of the input arguments. That is, successive invocations with the same argument values can produce a different answer. this type of function is sometimes called a *variant* function. Contrast this with a *deterministic function* (sometimes called a *not-variant function*), which always produces the same result for the same inputs.

not-variant function. See *deterministic function*.

NPSI. See *nonpartitioned secondary index.*

NRE. Network recovery element.

NUL. The null character ('\0'), which is represented by the value X'00'. In C, this character denotes the end of a string.

null. A special value that indicates the absence of information.

NULLIF. A scalar function that evaluates two passed expressions, returning either NULL if the arguments are equal or the value of the first argument if they are not.

null-terminated host variable. A varying-length host variable in which the end of the data is indicated by a null terminator.

null terminator. In C, the value that indicates the end of a string. For EBCDIC, ASCII, and Unicode UTF-8 strings, the null terminator is a single-byte value (X'00'). For Unicode UCS-2 (wide) strings, the null terminator is a double-byte value (X'0000').

O

OASN (origin application schedule number). In IMS, a 4-byte number that is assigned sequentially to each IMS schedule since the last cold start of IMS. The OASN is used as an identifier for a unit of work. In an 8-byte format, the first 4 bytes contain the schedule number and the last 4 bytes contain the number of IMS sync points (*commit points*) during the current schedule. The OASN is part of the NID for an IMS connection.

ODBC. Open Database Connectivity.

ODBC driver. A dynamically-linked library (DLL) that implements ODBC function calls and interacts with a data source.

OBID. Data object identifier.

Open Database Connectivity (ODBC). A Microsoft® database application programming interface (API) for C that allows access to database management systems by using callable SQL. ODBC does not require the use of an SQL preprocessor. In addition, ODBC provides an architecture that lets users add modules called *database drivers*, which link the application to their choice of database management systems at run time. This means that applications no longer need to be directly linked to the modules of all the database management systems that are supported.

ordinary identifier. An uppercase letter followed by zero or more characters, each of which is an uppercase letter, a digit, or the underscore character. An ordinary identifier must not be a reserved word.

ordinary token. A numeric constant, an ordinary identifier, a host identifier, or a keyword.

originating task. In a parallel group, the primary agent that receives data from other execution units (referred to as *parallel tasks*) that are executing portions of the query in parallel.

OS/390. Operating System/390®.

outer join. The result of a join operation that includes the matched rows of both tables that are being joined and preserves some or all of the unmatched rows of the tables that are being joined. See also *join*.

overloaded function. A function name for which multiple function instances exist.

P

package. An object containing a set of SQL statements that have been statically bound and that is available for processing. A package is sometimes also called an *application package*.

package list. An ordered list of package names that may be used to extend an application plan.

package name. The name of an object that is created by a BIND PACKAGE or REBIND PACKAGE command. The object is a bound version of a database request module (DBRM). The name consists of a location name, a collection ID, a package ID, and a version ID.

page. A unit of storage within a table space (4 KB, 8 KB, 16 KB, or 32 KB) or index space (4 KB). In a table space, a page contains one or more rows of a table. In a LOB table space, a LOB value can span more than one page, but no more than one LOB value is stored on a page.

page set. Another way to refer to a table space or index space. Each page set consists of a collection of VSAM data sets.

page set recovery pending (PSRCP). A restrictive state of an index space. In this case, the entire page set must be recovered. Recovery of a logical part is prohibited.

panel. A predefined display image that defines the locations and characteristics of display fields on a display surface (for example, a *menu panel*).

parallel complex. A cluster of machines that work together to handle multiple transactions and applications.

parallel group. A set of consecutive operations that execute in parallel and that have the same number of parallel tasks.

parallel I/O processing. A form of I/O processing in which DB2 initiates multiple concurrent requests for a single user query and performs I/O processing concurrently (in *parallel*) on multiple data partitions.

parallelism assistant. In Sysplex query parallelism, a DB2 subsystem that helps to process parts of a parallel query that originates on another DB2 subsystem in the data sharing group.

parallelism coordinator. In Sysplex query parallelism, the DB2 subsystem from which the parallel query originates.

Parallel Sysplex. A set of z/OS systems that communicate and cooperate with each other through certain multisystem hardware components and software services to process customer workloads.

parallel task. The execution unit that is dynamically created to process a query in parallel. A parallel task is implemented by a z/OS service request block.

parameter marker. A question mark (?) that appears in a statement string of a dynamic SQL statement. The question mark can appear where a host variable could appear if the statement string were a static SQL statement.

parameter-name. An SQL identifier that designates a parameter in an SQL procedure or an SQL function.

parent key. A primary key or unique key in the parent table of a referential constraint. The values of a parent key determine the valid values of the foreign key in the referential constraint.

parent lock. For explicit hierarchical locking, a lock that is held on a resource that might have child locks that are lower in the hierarchy. A parent lock is usually the table space lock or the partition intent lock. See also *child lock*.

parent row. A row whose primary key value is the foreign key value of a dependent row.

parent table. A table whose primary key is referenced by the foreign key of a dependent table.

parent table space. A table space that contains a parent table. A table space containing a dependent of that table is a dependent table space.

participant. An entity other than the commit coordinator that takes part in the commit process. The term participant is synonymous with *agent* in SNA.

partition. A portion of a page set. Each partition corresponds to a single, independently extendable data set. Partitions can be extended to a maximum size of 1, 2, or 4 GB, depending on the number of partitions in the partitioned page set. All partitions of a given page set have the same maximum size.

partitioned data set (PDS). A data set in disk storage that is divided into partitions, which are called members. Each partition can contain a program, part of a program, or data. The term partitioned data set is synonymous with program library.

partitioned index. An index that is physically partitioned. Both partitioning indexes and secondary indexes can be partitioned.

partitioned page set. A partitioned table space or an index space. Header pages, space map pages, data pages, and index pages reference data only within the scope of the partition.

partitioned table space. A table space that is subdivided into parts (based on index key range), each of which can be processed independently by utilities.

partitioning index. An index in which the leftmost columns are the partitioning columns of the table. The index can be partitioned or nonpartitioned.

partition pruning. The removal from consideration of inapplicable partitions through setting up predicates in a query on a partitioned table to access only certain partitions to satisfy the query.

partner logical unit. An access point in the SNA network that is connected to the local DB2 subsystem by way of a VTAM conversation.

path. See *SQL path*.

PCT. Program control table (in CICS).

PDS. Partitioned data set.

piece. A data set of a nonpartitioned page set.

physical claim. A claim on an entire nonpartitioning index.

physical consistency. The state of a page that is not in a partially changed state.

physical drain. A drain on an entire nonpartitioning index.

physical lock (P-lock). A type of lock that DB2 acquires to provide consistency of data that is cached in different DB2 subsystems. Physical locks are used only in data sharing environments. Contrast with *logical lock (L-lock)*.

physical lock contention. Conflicting states of the requesters for a physical lock. See also *negotiable lock*.

physically complete. The state in which the concurrent copy process is completed and the output data set has been created.

plan. See *application plan*.

plan allocation. The process of allocating DB2 resources to a plan in preparation for execution.

plan member. The bound copy of a DBRM that is identified in the member clause.

plan name. The name of an application plan.

plan segmentation. The dividing of each plan into sections. When a section is needed, it is independently brought into the EDM pool.

P-lock. Physical lock.

PLT. Program list table (in CICS).

point of consistency. A time when all recoverable data that an application accesses is consistent with other data. The term point of consistency is synonymous with *sync point* or *commit point*.

policy. See *CFRM policy*.

Portable Operating System Interface (POSIX). The IEEE operating system interface standard, which defines the Pthread standard of threading. See also *Pthread*.

POSIX. Portable Operating System Interface.

postponed abort UR. A unit of recovery that was inflight or in-abort, was interrupted by system failure or cancellation, and did not complete backout during restart.

PPT. (1) Processing program table (in CICS). (2) Program properties table (in z/OS).

precision. In SQL, the total number of digits in a decimal number (called the *size* in the C language). In the C language, the number of digits to the right of the decimal point (called the *scale* in SQL). The DB2 library uses the SQL terms.

precompilation. A processing of application programs containing SQL statements that takes place before compilation. SQL statements are replaced with statements that are recognized by the host language compiler. Output from this precompilation includes source code that can be submitted to the compiler and the database request module (DBRM) that is input to the bind process.

predicate. An element of a search condition that expresses or implies a comparison operation.

prefix. A code at the beginning of a message or record.

preformat. The process of preparing a VSAM ESDS for DB2 use, by writing specific data patterns.

prepare. The first phase of a two-phase commit process in which all participants are requested to prepare for commit.

prepared SQL statement. A named object that is the executable form of an SQL statement that has been processed by the PREPARE statement.

presumed-abort. An optimization of the presumed-nothing two-phase commit protocol that reduces the number of recovery log records, the duration of state maintenance, and the number of messages between coordinator and participant. The optimization also modifies the indoubt resolution responsibility.

presumed-nothing. The standard two-phase commit protocol that defines coordinator and participant responsibilities, relative to logical unit of work states, recovery logging, and indoubt resolution.

primary authorization ID. The authorization ID that is used to identify the application process to DB2.

primary group buffer pool. For a duplexed group buffer pool, the structure that is used to maintain the coherency of cached data. This structure is used for page registration and cross-invalidation. The z/OS equivalent is *old* structure. Compare with *secondary group buffer pool*.

primary index. An index that enforces the uniqueness of a primary key.

primary key. In a relational database, a unique, nonnull key that is part of the definition of a table. A table cannot be defined as a parent unless it has a unique key or primary key.

principal. An entity that can communicate securely with another entity. In Kerberos, principals are represented as entries in the Kerberos registry database and include users, servers, computers, and others.

principal name. The name by which a principal is known to the DCE security services.

private connection. A communications connection that is specific to DB2.

private protocol access. A method of accessing distributed data by which you can direct a query to another DB2 system. Contrast with *DRDA access*.

private protocol connection. A DB2 private connection of the application process. See also *private connection*.

privilege. The capability of performing a specific function, sometimes on a specific object. The types of privileges are:

explicit privileges, which have names and are held as the result of SQL GRANT and REVOKE statements. For example, the SELECT privilege.

implicit privileges, which accompany the ownership of an object, such as the privilege to drop a synonym that one owns, or the holding of an authority, such as the privilege of SYSADM authority to terminate any utility job.

privilege set. For the installation SYSADM ID, the set of all possible privileges. For any other authorization ID, the set of all privileges that are recorded for that ID in the DB2 catalog.

process. In DB2, the unit to which DB2 allocates resources and locks. Sometimes called an *application process*, a process involves the execution of one or more programs. The execution of an SQL statement is always associated with some process. The means of initiating and terminating a process are dependent on the environment.

program. A single, compilable collection of executable statements in a programming language.

program temporary fix (PTF). A solution or bypass of a problem that is diagnosed as a result of a defect in a current unaltered release of a licensed program. An authorized program analysis report (APAR) fix is corrective service for an existing problem. A PTF is preventive service for problems that might be encountered by other users of the product. A PTF is *temporary*, because a permanent fix is usually not incorporated into the product until its next release.

protected conversation. A VTAM conversation that supports two-phase commit flows.

PSRCP. Page set recovery pending.

PTF. Program temporary fix.

Pthread. The POSIX threading standard model for splitting an application into subtasks. The Pthread standard includes functions for creating threads, terminating threads, synchronizing threads through locking, and other thread control facilities.

Q

QMF. Query Management Facility.

QSAM. Queued sequential access method.

query. A component of certain SQL statements that specifies a result table.

query block. The part of a query that is represented by one of the FROM clauses. Each FROM clause can have multiple query blocks, depending on DB2's internal processing of the query.

query CP parallelism. Parallel execution of a single query, which is accomplished by using multiple tasks. See also *Sysplex query parallelism*.

query I/O parallelism. Parallel access of data, which is accomplished by triggering multiple I/O requests within a single query.

queued sequential access method (QSAM). An extended version of the basic sequential access method (BSAM). When this method is used, a queue of data blocks is formed. Input data blocks await processing, and output data blocks await transfer to auxiliary storage or to an output device.

quiesce point. A point at which data is consistent as a result of running the DB2 QUIESCE utility.

quiesced member state. A state of a member of a data sharing group. An active member becomes quiesced when a STOP DB2 command takes effect without a failure. If the member's task, address space, or z/OS system fails before the command takes effect, the member state is failed.

R

RACF. Resource Access Control Facility, which is a component of the z/OS Security Server.

RAMAC. IBM family of enterprise disk storage system products.

RBA. Relative byte address.

RCT. Resource control table (in CICS attachment facility).

RDB. Relational database.

RDBMS. Relational database management system.

RDBNAM. Relational database name.

RDF. Record definition field.

read stability (RS). An isolation level that is similar to repeatable read but does not completely isolate an application process from all other concurrently executing application processes. Under level RS, an application that issues the same query more than once might read additional rows that were inserted and committed by a concurrently executing application process.

rebind. The creation of a new application plan for an application program that has been bound previously. If, for example, you have added an index for a table that your application accesses, you must rebind the application in order to take advantage of that index.

rebuild. The process of reallocating a coupling facility structure. For the shared communications area (SCA) and lock structure, the structure is repopulated; for the group buffer pool, changed pages are usually cast out to disk, and the new structure is populated only with changed pages that were not successfully cast out.

RECFM. Record format.

record. The storage representation of a row or other data.

record identifier (RID). A unique identifier that DB2 uses internally to identify a row of data in a table. Compare with *row ID*.

record identifier (RID) pool. An area of main storage that is used for sorting record identifiers during list-prefetch processing.

record length. The sum of the length of all the columns in a table, which is the length of the data as it is physically stored in the database. Records can be fixed length or varying length, depending on how the columns are defined. If all columns are fixed-length columns, the record is a fixed-length record. If one or more columns are varying-length columns, the record is a varying-length column.

Recoverable Resource Manager Services attachment facility (RRSAF). A DB2 subcomponent that uses Resource Recovery Services to coordinate resource commitment between DB2 and all other resource managers that also use RRS in a z/OS system.

recovery. The process of rebuilding databases after a system failure.

recovery log. A collection of records that describes the events that occur during DB2 execution and indicates their sequence. The recorded information is used for recovery in the event of a failure during DB2 execution.

recovery manager. (1) A subcomponent that supplies coordination services that control the interaction of DB2 resource managers during commit, abort, checkpoint, and restart processes. The recovery manager also supports the recovery mechanisms of other subsystems (for example, IMS) by acting as a participant in the other subsystem's process for protecting data that has reached a point of consistency. (2) A coordinator or a participant (or both), in the execution of a two-phase commit, that can access a recovery log that maintains the state of the logical unit of work and names the immediate upstream coordinator and downstream participants.

recovery pending (RECP). A condition that prevents SQL access to a table space that needs to be recovered.

recovery token. An identifier for an element that is used in recovery (for example, NID or URID).

RECP. Recovery pending.

redo. A state of a unit of recovery that indicates that changes are to be reapplied to the disk media to ensure data integrity.

reentrant. Executable code that can reside in storage as one shared copy for all threads. Reentrant code is not self-modifying and provides separate storage areas for each thread. Reentrancy is a compiler and operating system concept, and reentrancy alone is not enough to guarantee logically consistent results when multithreading. See also *threadsafe*.

referential constraint. The requirement that nonnull values of a designated foreign key are valid only if they equal values of the primary key of a designated table.

referential integrity. The state of a database in which all values of all foreign keys are valid. Maintaining referential integrity requires the enforcement of referential constraints on all operations that change the data in a table on which the referential constraints are defined.

referential structure. A set of tables and relationships that includes at least one table and, for every table in the set, all the relationships in which that table participates and all the tables to which it is related.

refresh age. The time duration between the current time and the time during which a materialized query table was last refreshed.

registry. See *registry database*.

registry database. A database of security information about principals, groups, organizations, accounts, and security policies.

relational database (RDB). A database that can be perceived as a set of tables and manipulated in accordance with the relational model of data.

relational database management system (RDBMS). A collection of hardware and software that organizes and provides access to a relational database.

relational database name (RDBNAM). A unique identifier for an RDBMS within a network. In DB2, this must be the value in the LOCATION column of table SYSIBM.LOCATIONS in the CDB. DB2 publications refer to the name of another RDBMS as a LOCATION value or a location name.

relationship. A defined connection between the rows of a table or the rows of two tables. A relationship is the internal representation of a referential constraint.

relative byte address (RBA). The offset of a data record or control interval from the beginning of the storage space that is allocated to the data set or file to which it belongs.

remigration. The process of returning to a current release of DB2 following a fallback to a previous release. This procedure constitutes another migration process.

remote. Any object that is maintained by a remote DB2 subsystem (that is, by a DB2 subsystem other than the local one). A *remote view*, for example, is a view that is maintained by a remote DB2 subsystem. Contrast with *local*.

remote attach request. A request by a remote location to attach to the local DB2 subsystem. Specifically, the request that is sent is an SNA Function Management Header 5.

remote subsystem. Any relational DBMS, except the *local subsystem*, with which the user or application can communicate. The subsystem need not be remote in any physical sense, and might even operate on the same processor under the same z/OS system.

reoptimization. The DB2 process of reconsidering the access path of an SQL statement at run time; during reoptimization, DB2 uses the values of host variables, parameter markers, or special registers.

REORG pending (REORP). A condition that restricts SQL access and most utility access to an object that must be reorganized.

REORP. REORG pending.

repeatable read (RR). The isolation level that provides maximum protection from other executing application programs. When an application program executes with repeatable read protection, rows that the program references cannot be changed by other programs until the program reaches a commit point.

repeating group. A situation in which an entity includes multiple attributes that are inherently the same. The presence of a repeating group violates the requirement of first normal form. In an entity that satisfies the requirement of first normal form, each attribute is independent and unique in its meaning and its name. See also *normalization*.

replay detection mechanism. A method that allows a principal to detect whether a request is a valid request from a source that can be trusted or whether an untrustworthy entity has captured information from a previous exchange and is replaying the information exchange to gain access to the principal.

request commit. The vote that is submitted to the prepare phase if the participant has modified data and is prepared to commit or roll back.

requester. The source of a request to access data at a remote server. In the DB2 environment, the requester function is provided by the distributed data facility.

resource. The object of a lock or claim, which could be a table space, an index space, a data partition, an index partition, or a logical partition.

resource allocation. The part of plan allocation that deals specifically with the database resources.

resource control table (RCT). A construct of the CICS attachment facility, created by site-provided macro parameters, that defines authorization and access attributes for transactions or transaction groups.

resource definition online. A CICS feature that you use to define CICS resources online without assembling tables.

resource limit facility (RLF). A portion of DB2 code that prevents dynamic manipulative SQL statements from exceeding specified time limits. The resource limit facility is sometimes called the governor.

resource limit specification table (RLST). A site-defined table that specifies the limits to be enforced by the resource limit facility.

resource manager. (1) A function that is responsible for managing a particular resource and that guarantees the consistency of all updates made to recoverable resources within a logical unit of work. The resource that is being managed can be physical (for example, disk or main storage) or logical (for example, a particular type of system service). (2) A participant, in the execution of a two-phase commit, that has recoverable resources that could have been modified. The resource manager has access to a recovery log so that it can commit or roll back the effects of the logical unit of work to the recoverable resources.

restart pending (RESTP). A restrictive state of a page set or partition that indicates that restart (backout) work needs to be performed on the object. All access to the page set or partition is denied except for access by the:
- RECOVER POSTPONED command
- Automatic online backout (which DB2 invokes after restart if the system parameter LBACKOUT=AUTO)

RESTP. Restart pending.

result set. The set of rows that a stored procedure returns to a client application.

result set locator. A 4-byte value that DB2 uses to uniquely identify a query result set that a stored procedure returns.

result table. The set of rows that are specified by a SELECT statement.

retained lock. A MODIFY lock that a DB2 subsystem was holding at the time of a subsystem failure. The lock is retained in the coupling facility lock structure across a DB2 failure.

RID. Record identifier.

RID pool. Record identifier pool.

right outer join. The result of a join operation that includes the matched rows of both tables that are being joined and preserves the unmatched rows of the second join operand. See also *join*.

RLF. Resource limit facility.

RLST. Resource limit specification table.

RMID. Resource manager identifier.

RO. Read-only access.

rollback. The process of restoring data that was changed by SQL statements to the state at its last commit point. All locks are freed. Contrast with *commit*.

root page. The index page that is at the highest level (or the beginning point) in an index.

routine. A term that refers to either a user-defined function or a stored procedure.

row. The horizontal component of a table. A row consists of a sequence of values, one for each column of the table.

ROWID. Row identifier.

row identifier (ROWID). A value that uniquely identifies a row. This value is stored with the row and never changes.

row lock. A lock on a single row of data.

| **rowset.** A set of rows for which a cursor position is
| established.

| **rowset cursor.** A cursor that is defined so that one or
| more rows can be returned as a rowset for a single
| FETCH statement, and the cursor is positioned on the
| set of rows that is fetched.

| **rowset-positioned access.** The ability to retrieve
| multiple rows from a single FETCH statement.

| **row-positioned access.** The ability to retrieve a single
| row from a single FETCH statement.

row trigger. A trigger that is defined with the trigger granularity FOR EACH ROW.

RRE. Residual recovery entry (in IMS).

RRSAF. Recoverable Resource Manager Services attachment facility.

RS. Read stability.

RTT. Resource translation table.

RURE. Restart URE.

S

savepoint. A named entity that represents the state of data and schemas at a particular point in time within a unit of work. SQL statements exist to set a savepoint, release a savepoint, and restore data and schemas to the state that the savepoint represents. The restoration of data and schemas to a savepoint is usually referred to as *rolling back to a savepoint*.

SBCS. Single-byte character set.

SCA. Shared communications area.

scalar function. An SQL operation that produces a
single value from another value and is expressed as a
function name, followed by a list of arguments that are
enclosed in parentheses. Contrast with *aggregate*
function.

scale. In SQL, the number of digits to the right of the decimal point (called the *precision* in the C language). The DB2 library uses the SQL definition.

schema. (1) The organization or structure of a database. (2) A logical grouping for user-defined functions, distinct types, triggers, and stored procedures. When an object of one of these types is created, it is assigned to one schema, which is determined by the name of the object. For example, the following statement creates a distinct type T in schema C:

```
CREATE DISTINCT TYPE C.T ...
```

scrollability. The ability to use a cursor to fetch in either a forward or backward direction. The FETCH statement supports multiple fetch orientations to indicate the new position of the cursor. See also *fetch orientation*.

scrollable cursor. A cursor that can be moved in both a forward and a backward direction.

SDWA. System diagnostic work area.

search condition. A criterion for selecting rows from a table. A search condition consists of one or more predicates.

secondary authorization ID. An authorization ID that has been associated with a primary authorization ID by an authorization exit routine.

secondary group buffer pool. For a duplexed group buffer pool, the structure that is used to back up changed pages that are written to the primary group buffer pool. No page registration or cross-invalidation occurs using the secondary group buffer pool. The z/OS equivalent is *new* structure.

secondary index. A nonpartitioning index on a partitioned table.

section. The segment of a plan or package that contains the executable structures for a single SQL statement. For most SQL statements, one section in the plan exists for each SQL statement in the source program. However, for cursor-related statements, the DECLARE, OPEN, FETCH, and CLOSE statements reference the same section because they each refer to the SELECT statement that is named in the DECLARE CURSOR statement. SQL statements such as COMMIT, ROLLBACK, and some SET statements do not use a section.

segment. A group of pages that holds rows of a single table. See also *segmented table space*.

segmented table space. A table space that is divided into equal-sized groups of pages called segments. Segments are assigned to tables so that rows of different tables are never stored in the same segment.

self-referencing constraint. A referential constraint that defines a relationship in which a table is a dependent of itself.

self-referencing table. A table with a self-referencing constraint.

sensitive cursor. A cursor that is sensitive to changes that are made to the database after the result table has been materialized.

sequence. A user-defined object that generates a sequence of numeric values according to user specifications.

sequential data set. A non-DB2 data set whose records are organized on the basis of their successive physical positions, such as on magnetic tape. Several of the DB2 database utilities require sequential data sets.

sequential prefetch. A mechanism that triggers consecutive asynchronous I/O operations. Pages are fetched before they are required, and several pages are read with a single I/O operation.

serial cursor. A cursor that can be moved only in a forward direction.

serialized profile. A Java object that contains SQL statements and descriptions of host variables. The SQLJ translator produces a serialized profile for each connection context.

server. The target of a request from a remote requester. In the DB2 environment, the server function is provided by the distributed data facility, which is used to access DB2 data from remote applications.

server-side programming. A method for adding DB2 data into dynamic Web pages.

service class. An eight-character identifier that is used by the z/OS Workload Manager to associate user performance goals with a particular DDF thread or stored procedure. A service class is also used to classify work on parallelism assistants.

service request block. A unit of work that is scheduled to execute in another address space.

session. A link between two nodes in a VTAM network.

session protocols. The available set of SNA communication requests and responses.

shared communications area (SCA). A coupling facility list structure that a DB2 data sharing group uses for inter-DB2 communication.

share lock. A lock that prevents concurrently executing application processes from changing data, but not from reading data. Contrast with *exclusive lock*.

shift-in character. A special control character (X'0F') that is used in EBCDIC systems to denote that the subsequent bytes represent SBCS characters. See also *shift-out character*.

shift-out character. A special control character (X'0E') that is used in EBCDIC systems to denote that the subsequent bytes, up to the next shift-in control character, represent DBCS characters. See also *shift-in character*.

sign-on. A request that is made on behalf of an individual CICS or IMS application process by an attachment facility to enable DB2 to verify that it is authorized to use DB2 resources.

simple page set. A nonpartitioned page set. A simple page set initially consists of a single data set (page set piece). If and when that data set is extended to 2 GB, another data set is created, and so on, up to a total of 32 data sets. DB2 considers the data sets to be a single contiguous linear address space containing a maximum of 64 GB. Data is stored in the next available location within this address space without regard to any partitioning scheme.

simple table space. A table space that is neither partitioned nor segmented.

single-byte character set (SBCS). A set of characters in which each character is represented by a single byte. Contrast with *double-byte character set* or *multibyte character set*.

single-precision floating point number. A 32-bit approximate representation of a real number.

size. In the C language, the total number of digits in a decimal number (called the *precision* in SQL). The DB2 library uses the SQL term.

SMF. System Management Facilities.

SMP/E. System Modification Program/Extended.

SMS. Storage Management Subsystem.

SNA. Systems Network Architecture.

SNA network. The part of a network that conforms to the formats and protocols of Systems Network Architecture (SNA).

socket. A callable TCP/IP programming interface that TCP/IP network applications use to communicate with remote TCP/IP partners.

sourced function. A function that is implemented by another built-in or user-defined function that is already known to the database manager. This function can be a scalar function or a column (aggregating) function; it returns a single value from a set of values (for example, MAX or AVG). Contrast with *built-in function, external function,* and *SQL function*.

source program. A set of host language statements and SQL statements that is processed by an SQL precompiler.

source table. A table that can be a base table, a view, a table expression, or a user-defined table function.

source type. An existing type that DB2 uses to internally represent a distinct type.

space. A sequence of one or more blank characters.

special register. A storage area that DB2 defines for an application process to use for storing information that can be referenced in SQL statements. Examples of special registers are USER and CURRENT DATE.

specific function name. A particular user-defined function that is known to the database manager by its specific name. Many specific user-defined functions can have the same function name. When a user-defined function is defined to the database, every function is assigned a specific name that is unique within its schema. Either the user can provide this name, or a default name is used.

SPUFI. SQL Processor Using File Input.

SQL. Structured Query Language.

SQL authorization ID (SQL ID). The authorization ID that is used for checking dynamic SQL statements in some situations.

SQLCA. SQL communication area.

SQL communication area (SQLCA). A structure that is used to provide an application program with information about the execution of its SQL statements.

SQL connection. An association between an application process and a local or remote application server or database server.

SQLDA. SQL descriptor area.

SQL descriptor area (SQLDA). A structure that describes input variables, output variables, or the columns of a result table.

SQL escape character. The symbol that is used to enclose an SQL delimited identifier. This symbol is the double quotation mark ("). See also *escape character*.

SQL function. A user-defined function in which the CREATE FUNCTION statement contains the source code. The source code is a single SQL expression that evaluates to a single value. The SQL user-defined function can return only one parameter.

SQL ID. SQL authorization ID.

SQLJ. Structured Query Language (SQL) that is embedded in the Java programming language.

SQL path. An ordered list of schema names that are used in the resolution of unqualified references to user-defined functions, distinct types, and stored procedures. In dynamic SQL, the current path is found in the CURRENT PATH special register. In static SQL, it is defined in the PATH bind option.

SQL procedure. A user-written program that can be invoked with the SQL CALL statement. Contrast with *external procedure*.

SQL processing conversation. Any conversation that requires access of DB2 data, either through an application or by dynamic query requests.

SQL Processor Using File Input (SPUFI). A facility of the TSO attachment subcomponent that enables the DB2I user to execute SQL statements without embedding them in an application program.

SQL return code. Either SQLCODE or SQLSTATE.

SQL routine. A user-defined function or stored procedure that is based on code that is written in SQL.

SQL statement coprocessor. An alternative to the DB2 precompiler that lets the user process SQL statements at compile time. The user invokes an SQL statement coprocessor by specifying a compiler option.

SQL string delimiter. A symbol that is used to enclose an SQL string constant. The SQL string delimiter is the apostrophe ('), except in COBOL applications, where the user assigns the symbol, which is either an apostrophe or a double quotation mark (").

SRB. Service request block.

SSI. Subsystem interface (in z/OS).

SSM. Subsystem member (in IMS).

stand-alone. An attribute of a program that means that it is capable of executing separately from DB2, without using DB2 services.

star join. A method of joining a dimension column of a fact table to the key column of the corresponding dimension table. See also *join*, *dimension*, and *star schema*.

star schema. The combination of a fact table (which contains most of the data) and a number of dimension tables. See also *star join*, *dimension*, and *dimension table*.

statement handle. In DB2 ODBC, the data object that contains information about an SQL statement that is managed by DB2 ODBC. This includes information such as dynamic arguments, bindings for dynamic arguments and columns, cursor information, result values, and status information. Each statement handle is associated with the connection handle.

statement string. For a dynamic SQL statement, the character string form of the statement.

statement trigger. A trigger that is defined with the trigger granularity FOR EACH STATEMENT.

static cursor. A named control structure that does not change the size of the result table or the order of its rows after an application opens the cursor. Contrast with *dynamic cursor*.

static SQL. SQL statements, embedded within a program, that are prepared during the program preparation process (before the program is executed). After being prepared, the SQL statement does not change (although values of host variables that are specified by the statement might change).

storage group. A named set of disks on which DB2 data can be stored.

stored procedure. A user-written application program that can be invoked through the use of the SQL CALL statement.

string. See *character string* or *graphic string*.

strong typing. A process that guarantees that only user-defined functions and operations that are defined on a distinct type can be applied to that type. For example, you cannot directly compare two currency types, such as Canadian dollars and U.S. dollars. But you can provide a user-defined function to convert one currency to the other and then do the comparison.

structure. (1) A name that refers collectively to different types of DB2 objects, such as tables, databases, views, indexes, and table spaces. (2) A construct that uses z/OS to map and manage storage on a coupling facility. See also *cache structure*, *list structure*, or *lock structure*.

Structured Query Language (SQL). A standardized language for defining and manipulating data in a relational database.

structure owner. In relation to group buffer pools, the DB2 member that is responsible for the following activities:

- Coordinating rebuild, checkpoint, and damage assessment processing
- Monitoring the group buffer pool threshold and notifying castout owners when the threshold has been reached

subcomponent. A group of closely related DB2 modules that work together to provide a general function.

subject table. The table for which a trigger is created. When the defined triggering event occurs on this table, the trigger is activated.

subpage. The unit into which a physical index page can be divided.

subquery. A SELECT statement within the WHERE or HAVING clause of another SQL statement; a nested SQL statement.

subselect. That form of a query that does not include an ORDER BY clause, an UPDATE clause, or UNION operators.

substitution character. A unique character that is substituted during character conversion for any characters in the source program that do not have a match in the target coding representation.

subsystem. A distinct instance of a relational database management system (RDBMS).

surrogate pair. A coded representation for a single character that consists of a sequence of two 16-bit code units, in which the first value of the pair is a high-surrogate code unit in the range U+D800 through U+DBFF, and the second value is a low-surrogate code unit in the range U+DC00 through U+DFFF. Surrogate pairs provide an extension mechanism for encoding 917 476 characters without requiring the use of 32-bit characters.

SVC dump. A dump that is issued when a z/OS or a DB2 functional recovery routine detects an error.

sync point. See *commit point*.

syncpoint tree. The tree of recovery managers and resource managers that are involved in a logical unit of work, starting with the recovery manager, that make the final commit decision.

synonym. In SQL, an alternative name for a table or view. Synonyms can be used to refer only to objects at the subsystem in which the synonym is defined.

syntactic character set. A set of 81 graphic characters that are registered in the IBM registry as character set 00640. This set was originally recommended to the programming language community to be used for syntactic purposes toward maximizing portability and interchangeability across systems and country boundaries. It is contained in most of the primary registered character sets, with a few exceptions. See also *invariant character set*.

Sysplex. See *Parallel Sysplex*.

Sysplex query parallelism. Parallel execution of a single query that is accomplished by using multiple tasks on more than one DB2 subsystem. See also *query CP parallelism*.

system administrator. The person at a computer installation who designs, controls, and manages the use of the computer system.

system agent. A work request that DB2 creates internally such as prefetch processing, deferred writes, and service tasks.

system conversation. The conversation that two DB2 subsystems must establish to process system messages before any distributed processing can begin.

system diagnostic work area (SDWA). The data that is recorded in a SYS1.LOGREC entry that describes a program or hardware error.

system-directed connection. A connection that a relational DBMS manages by processing SQL statements with three-part names.

System Modification Program/Extended (SMP/E). A z/OS tool for making software changes in programming systems (such as DB2) and for controlling those changes.

Systems Network Architecture (SNA). The description of the logical structure, formats, protocols, and operational sequences for transmitting information through and controlling the configuration and operation of networks.

SYS1.DUMPxx data set. A data set that contains a system dump (in z/OS).

SYS1.LOGREC. A service aid that contains important information about program and hardware errors (in z/OS).

T

table. A named data object consisting of a specific number of columns and some number of unordered rows. See also *base table* or *temporary table*.

table-controlled partitioning. A type of partitioning in which partition boundaries for a partitioned table are controlled by values that are defined in the CREATE TABLE statement. Partition limits are saved in the LIMITKEY_INTERNAL column of the SYSIBM.SYSTABLEPART catalog table.

table function. A function that receives a set of arguments and returns a table to the SQL statement that references the function. A table function can be referenced only in the FROM clause of a subselect.

table locator. A mechanism that allows access to trigger transition tables in the FROM clause of SELECT statements, in the subselect of INSERT statements, or from within user-defined functions. A table locator is a fullword integer value that represents a transition table.

table space. A page set that is used to store the records in one or more tables.

table space set. A set of table spaces and partitions that should be recovered together for one of these reasons:

- Each of them contains a table that is a parent or descendent of a table in one of the others.
- The set contains a base table and associated auxiliary tables.

A table space set can contain both types of relationships.

task control block (TCB). A z/OS control block that is used to communicate information about tasks within an address space that are connected to DB2. See also *address space connection*.

TB. Terabyte (1 099 511 627 776 bytes).

TCB. Task control block (in z/OS).

TCP/IP. A network communication protocol that computer systems use to exchange information across telecommunication links.

TCP/IP port. A 2-byte value that identifies an end user or a TCP/IP network application within a TCP/IP host.

template. A DB2 utilities output data set descriptor that is used for dynamic allocation. A template is defined by the TEMPLATE utility control statement.

temporary table. A table that holds temporary data. Temporary tables are useful for holding or sorting intermediate results from queries that contain a large number of rows. The two types of temporary table, which are created by different SQL statements, are the created temporary table and the declared temporary table. Contrast with *result table*. See also *created temporary table* and *declared temporary table*.

Terminal Monitor Program (TMP). A program that provides an interface between terminal users and command processors and has access to many system services (in z/OS).

thread. The DB2 structure that describes an application's connection, traces its progress, processes resource functions, and delimits its accessibility to DB2 resources and services. Most DB2 functions execute under a thread structure. See also *allied thread* and *database access thread*.

threadsafe. A characteristic of code that allows multithreading both by providing private storage areas for each thread, and by properly serializing shared (global) storage areas.

three-part name. The full name of a table, view, or alias. It consists of a location name, authorization ID, and an object name, separated by a period.

time. A three-part value that designates a time of day in hours, minutes, and seconds.

time duration. A decimal integer that represents a number of hours, minutes, and seconds.

timeout. Abnormal termination of either the DB2 subsystem or of an application because of the unavailability of resources. Installation specifications are set to determine both the amount of time DB2 is to wait for IRLM services after starting, and the amount of time IRLM is to wait if a resource that an application requests is unavailable. If either of these time specifications is exceeded, a timeout is declared.

Time-Sharing Option (TSO). An option in MVS that provides interactive time sharing from remote terminals.

timestamp. A seven-part value that consists of a date and time. The timestamp is expressed in years, months, days, hours, minutes, seconds, and microseconds.

TMP. Terminal Monitor Program.

to-do. A state of a unit of recovery that indicates that the unit of recovery's changes to recoverable DB2 resources are indoubt and must either be applied to the disk media or backed out, as determined by the commit coordinator.

trace. A DB2 facility that provides the ability to monitor and collect DB2 monitoring, auditing, performance, accounting, statistics, and serviceability (global) data.

transaction lock. A lock that is used to control concurrent execution of SQL statements.

transaction program name. In SNA LU 6.2 conversations, the name of the program at the remote logical unit that is to be the other half of the conversation.

transient XML data type. A data type for XML values that exists only during query processing.

transition table. A temporary table that contains all the affected rows of the subject table in their state before or after the triggering event occurs. Triggered SQL statements in the trigger definition can reference the table of changed rows in the old state or the new state.

transition variable. A variable that contains a column value of the affected row of the subject table in its state before or after the triggering event occurs. Triggered SQL statements in the trigger definition can reference the set of old values or the set of new values.

tree structure. A data structure that represents entities in nodes, with a most one parent node for each node, and with only one root node.

trigger. A set of SQL statements that are stored in a DB2 database and executed when a certain event occurs in a DB2 table.

trigger activation. The process that occurs when the trigger event that is defined in a trigger definition is executed. Trigger activation consists of the evaluation of the triggered action condition and conditional execution of the triggered SQL statements.

trigger activation time. An indication in the trigger definition of whether the trigger should be activated before or after the triggered event.

trigger body. The set of SQL statements that is executed when a trigger is activated and its triggered action condition evaluates to true. A trigger body is also called *triggered SQL statements*.

trigger cascading. The process that occurs when the triggered action of a trigger causes the activation of another trigger.

triggered action. The SQL logic that is performed when a trigger is activated. The triggered action consists of an optional triggered action condition and a set of triggered SQL statements that are executed only if the condition evaluates to true.

triggered action condition. An optional part of the triggered action. This Boolean condition appears as a WHEN clause and specifies a condition that DB2 evaluates to determine if the triggered SQL statements should be executed.

triggered SQL statements. The set of SQL statements that is executed when a trigger is activated and its triggered action condition evaluates to true. Triggered SQL statements are also called the *trigger body*.

trigger granularity. A characteristic of a trigger, which determines whether the trigger is activated:
- Only once for the triggering SQL statement
- Once for each row that the SQL statement modifies

triggering event. The specified operation in a trigger definition that causes the activation of that trigger. The triggering event is comprised of a triggering operation (INSERT, UPDATE, or DELETE) and a subject table on which the operation is performed.

triggering SQL operation. The SQL operation that causes a trigger to be activated when performed on the subject table.

trigger package. A package that is created when a CREATE TRIGGER statement is executed. The package is executed when the trigger is activated.

TSO. Time-Sharing Option.

TSO attachment facility. A DB2 facility consisting of the DSN command processor and DB2I. Applications that are not written for the CICS or IMS environments can run under the TSO attachment facility.

typed parameter marker. A parameter marker that is specified along with its target data type. It has the general form:

```
CAST(? AS data-type)
```

type 1 indexes. Indexes that were created by a release of DB2 before DB2 Version 4 or that are specified as type 1 indexes in Version 4. Contrast with *type 2 indexes*. As of Version 8, type 1 indexes are no longer supported.

type 2 indexes. Indexes that are created on a release of DB2 after Version 7 or that are specified as type 2 indexes in Version 4 or later.

U

UCS-2. Universal Character Set, coded in 2 octets, which means that characters are represented in 16-bits per character.

UDF. User-defined function.

UDT. User-defined data type. In DB2 UDB for z/OS, the term *distinct type* is used instead of user-defined data type. See *distinct type*.

uncommitted read (UR). The isolation level that allows an application to read uncommitted data.

underlying view. The view on which another view is directly or indirectly defined.

undo. A state of a unit of recovery that indicates that the changes that the unit of recovery made to recoverable DB2 resources must be backed out.

Unicode. A standard that parallels the ISO-10646 standard. Several implementations of the Unicode standard exist, all of which have the ability to represent a large percentage of the characters that are contained in the many scripts that are used throughout the world.

uniform resource locator (URL). A Web address, which offers a way of naming and locating specific items on the Web.

union. An SQL operation that combines the results of two SELECT statements. Unions are often used to merge lists of values that are obtained from several tables.

unique constraint. An SQL rule that no two values in a primary key, or in the key of a unique index, can be the same.

unique index. An index that ensures that no identical key values are stored in a column or a set of columns in a table.

unit of recovery. A recoverable sequence of operations within a single resource manager, such as an instance of DB2. Contrast with *unit of work*.

unit of recovery identifier (URID). The LOGRBA of the first log record for a unit of recovery. The URID also appears in all subsequent log records for that unit of recovery.

unit of work. A recoverable sequence of operations within an application process. At any time, an application process is a single unit of work, but the life of an application process can involve many units of work as a result of commit or rollback operations. In a *multisite update* operation, a single unit of work can include several *units of recovery*. Contrast with *unit of recovery*.

Universal Unique Identifier (UUID). An identifier that is immutable and unique across time and space (in z/OS).

unlock. The act of releasing an object or system resource that was previously locked and returning it to general availability within DB2.

untyped parameter marker. A parameter marker that is specified without its target data type. It has the form of a single question mark (?).

updatability. The ability of a cursor to perform positioned updates and deletes. The updatability of a cursor can be influenced by the SELECT statement and the cursor sensitivity option that is specified on the DECLARE CURSOR statement.

update hole. The location on which a cursor is positioned when a row in a result table is fetched again and the new values no longer satisfy the search condition. DB2 marks a row in the result table as an update hole when an update to the corresponding row in the database causes that row to no longer qualify for the result table.

update trigger. A trigger that is defined with the triggering SQL operation UPDATE.

upstream. The node in the syncpoint tree that is responsible, in addition to other recovery or resource managers, for coordinating the execution of a two-phase commit.

UR. Uncommitted read.

URE. Unit of recovery element.

URID . Unit of recovery identifier.

URL. Uniform resource locator.

user-defined data type (UDT). See *distinct type*.

user-defined function (UDF). A function that is defined to DB2 by using the CREATE FUNCTION statement and that can be referenced thereafter in SQL statements. A user-defined function can be an *external function,* a *sourced function,* or an *SQL function.* Contrast with *built-in function*.

user view. In logical data modeling, a model or representation of critical information that the business requires.

UTF-8. Unicode Transformation Format, 8-bit encoding form, which is designed for ease of use with existing ASCII-based systems. The CCSID value for data in UTF-8 format is 1208. DB2 UDB for z/OS supports UTF-8 in mixed data fields.

UTF-16. Unicode Transformation Format, 16-bit encoding form, which is designed to provide code values for over a million characters and a superset of UCS-2. The CCSID value for data in UTF-16 format is 1200. DB2 UDB for z/OS supports UTF-16 in graphic data fields.

UUID. Universal Unique Identifier.

V

value. The smallest unit of data that is manipulated in SQL.

variable. A data element that specifies a value that can be changed. A COBOL elementary data item is an example of a variable. Contrast with *constant*.

variant function. See *nondeterministic function*.

varying-length string. A character or graphic string whose length varies within set limits. Contrast with *fixed-length string*.

version. A member of a set of similar programs, DBRMs, packages, or LOBs.

A version of a program is the source code that is produced by precompiling the program. The program version is identified by the program name and a timestamp (consistency token).

A version of a DBRM is the DBRM that is produced by precompiling a program. The DBRM version is identified by the same program name and timestamp as a corresponding program version.

A version of a package is the result of binding a DBRM within a particular database system. The package version is identified by the same program name and consistency token as the DBRM.

A version of a LOB is a copy of a LOB value at a point in time. The version number for a LOB is stored in the auxiliary index entry for the LOB.

view. An alternative representation of data from one or more tables. A view can include all or some of the columns that are contained in tables on which it is defined.

view check option. An option that specifies whether every row that is inserted or updated through a view must conform to the definition of that view. A view check option can be specified with the WITH CASCADED CHECK OPTION, WITH CHECK OPTION, or WITH LOCAL CHECK OPTION clauses of the CREATE VIEW statement.

Virtual Storage Access Method (VSAM). An access method for direct or sequential processing of fixed- and varying-length records on disk devices. The records in a VSAM data set or file can be organized in logical sequence by a key field (key sequence), in the physical sequence in which they are written on the data set or file (entry-sequence), or by relative-record number (in z/OS).

Virtual Telecommunications Access Method (VTAM). An IBM licensed program that controls communication and the flow of data in an SNA network (in z/OS).

| **volatile table.** A table for which SQL operations
| choose index access whenever possible.

VSAM. Virtual Storage Access Method.

VTAM. Virtual Telecommunication Access Method (in z/OS).

W

warm start. The normal DB2 restart process, which involves reading and processing log records so that data that is under the control of DB2 is consistent. Contrast with *cold start*.

WLM application environment. A z/OS Workload Manager attribute that is associated with one or more stored procedures. The WLM application environment determines the address space in which a given DB2 stored procedure runs.

write to operator (WTO). An optional user-coded service that allows a message to be written to the system console operator informing the operator of errors and unusual system conditions that might need to be corrected (in z/OS).

WTO. Write to operator.

WTOR. Write to operator (WTO) with reply.

X

XCF. See *cross-system coupling facility*.

XES. See *cross-system extended services*.

| **XML.** See *Extensible Markup Language*.

| **XML attribute.** A name-value pair within a tagged
| XML element that modifies certain features of the
| element.

XML element. A logical structure in an XML
document that is delimited by a start and an end tag.
Anything between the start tag and the end tag is the
content of the element.

| **XML node.** The smallest unit of valid, complete
| structure in a document. For example, a node can
| represent an element, an attribute, or a text string.

| **XML publishing functions.** Functions that return
| XML values from SQL values.

X/Open. An independent, worldwide open systems organization that is supported by most of the world's largest information systems suppliers, user organizations, and software companies. X/Open's goal is to increase the portability of applications by combining existing and emerging standards.

XRF. Extended recovery facility.

Z

| **z/OS.** An operating system for the eServer™ product
| line that supports 64-bit real and virtual storage.

z/OS Distributed Computing Environment (z/OS DCE). A set of technologies that are provided by the Open Software Foundation to implement distributed computing.

Bibliography

DB2 Universal Database for z/OS Version 8 product information:

- *DB2 Administration Guide*, SC18-7413
- *DB2 Application Programming and SQL Guide*, SC18-7415
- *DB2 Application Programming Guide and Reference for Java*, SC18-7414
- *DB2 Codes*, GC18-9603
- *DB2 Command Reference*, SC18-7416
- *DB2 Common Criteria Guide*, SC18-9672
- *DB2 Data Sharing: Planning and Administration*, SC18-7417
- *DB2 Diagnosis Guide and Reference*, LY37-3201
- *DB2 Diagnostic Quick Reference Card*, LY37-3202
- *DB2 Image, Audio, and Video Extenders Administration and Programming*, SC26-9947
- *DB2 Installation Guide*, GC18-7418
- *DB2 Licensed Program Specifications*, GC18-7420
- *DB2 Management Clients Package Program Directory*, GI10-8567
- *DB2 Messages*, GC18-9602
- *DB2 ODBC Guide and Reference*, SC18-7423
- *The Official Introduction to DB2 UDB for z/OS*
- *DB2 Program Directory*, GI10-8566
- *DB2 RACF Access Control Module Guide*, SC18-7433
- *DB2 Reference for Remote DRDA Requesters and Servers*, SC18-7424
- *DB2 Reference Summary*, SX26-3853
- *DB2 Release Planning Guide*, SC18-7425
- *DB2 SQL Reference*, SC18-7426
- *DB2 Text Extender Administration and Programming*, SC26-9948
- *DB2 Utility Guide and Reference*, SC18-7427
- *DB2 What's New?*, GC18-7428
- *DB2 XML Extender for z/OS Administration and Programming*, SC18-7431

Books and resources about related products:

APL2®

- *APL2 Programming Guide*, SH21-1072
- *APL2 Programming: Language Reference*, SH21-1061
- *APL2 Programming: Using Structured Query Language (SQL)*, SH21-1057

BookManager® READ/MVS

- *BookManager READ/MVS V1R3: Installation Planning & Customization*, SC38-2035

C language: IBM C/C++ for z/OS

- *z/OS C/C++ Programming Guide*, SC09-4765
- *z/OS C/C++ Run-Time Library Reference*, SA22-7821

Character Data Representation Architecture

- *Character Data Representation Architecture Overview*, GC09-2207
- *Character Data Representation Architecture Reference and Registry*, SC09-2190

CICS Transaction Server for z/OS

The publication order numbers below are for Version 2 Release 2 and Version 2 Release 3 (with the release 2 number listed first).

- *CICS Transaction Server for z/OS Information Center*, SK3T-6903 or SK3T-6957.
- *CICS Transaction Server for z/OS Application Programming Guide*, SC34-5993 or SC34-6231
- *CICS Transaction Server for z/OS Application Programming Reference*, SC34-5994 or SC34-6232
- *CICS Transaction Server for z/OS CICS-RACF Security Guide*, SC34-6011 or SC34-6249
- *CICS Transaction Server for z/OS CICS Supplied Transactions*, SC34-5992 or SC34-6230
- *CICS Transaction Server for z/OS Customization Guide*, SC34-5989 or SC34-6227
- *CICS Transaction Server for z/OS Data Areas*, LY33-6100 or LY33-6103
- *CICS Transaction Server for z/OS DB2 Guide*, SC34-6014 or SC34-6252
- *CICS Transaction Server for z/OS External Interfaces Guide*, SC34-6006 or SC34-6244
- *CICS Transaction Server for z/OS Installation Guide*, GC34-5985 or GC34-6224
- *CICS Transaction Server for z/OS Intercommunication Guide*, SC34-6005 or SC34-6243
- *CICS Transaction Server for z/OS Messages and Codes*, GC34-6003 or GC34-6241
- *CICS Transaction Server for z/OS Operations and Utilities Guide*, SC34-5991 or SC34-6229

© Copyright IBM Corp. 1982, 2008

- *CICS Transaction Server for z/OS Performance Guide*, SC34-6009 or SC34-6247
- *CICS Transaction Server for z/OS Problem Determination Guide*, SC34-6002 or SC34-6239
- *CICS Transaction Server for z/OS Release Guide*, GC34-5983 or GC34-6218
- *CICS Transaction Server for z/OS Resource Definition Guide*, SC34-5990 or SC34-6228
- *CICS Transaction Server for z/OS System Definition Guide*, SC34-5988 or SC34–6226
- *CICS Transaction Server for z/OS System Programming Reference*, SC34-5595 or SC34–6233

CICS Transaction Server for OS/390
- *CICS Transaction Server for OS/390 Application Programming Guide*, SC33-1687
- *CICS Transaction Server for OS/390 DB2 Guide*, SC33-1939
- *CICS Transaction Server for OS/390 External Interfaces Guide*, SC33-1944
- *CICS Transaction Server for OS/390 Resource Definition Guide*, SC33-1684

COBOL:
- *IBM COBOL Language Reference*, SC27-1408
- *Enterprise COBOL for z/OS Programming Guide*, SC27-1412

Database Design
- *DB2 for z/OS and OS/390 Development for Performance Volume I* by Gabrielle Wiorkowski, Gabrielle & Associates, ISBN 0-96684-605-2
- *DB2 for z/OS and OS/390 Development for Performance Volume II* by Gabrielle Wiorkowski, Gabrielle & Associates, ISBN 0-96684-606-0
- *Handbook of Relational Database Design* by C. Fleming and B. Von Halle, Addison Wesley, ISBN 0-20111-434-8

DB2 Administration Tool
- *DB2 Administration Tool for z/OS User's Guide and Reference*, available on the Web at www.ibm.com/software/data/db2imstools/library.html

DB2 Buffer Pool Analyzer for z/OS
- *DB2 Buffer Pool Tool for z/OS User's Guide and Reference,* available on the Web at www.ibm.com/software/data/db2imstools/library.html

DB2 Connect
- *IBM DB2 Connect Quick Beginnings for DB2 Connect Enterprise Edition*, GC09-4833
- *IBM DB2 Connect Quick Beginnings for DB2 Connect Personal Edition*, GC09-4834
- *IBM DB2 Connect User's Guide*, SC09-4835

DB2 DataPropagator
- *DB2 Universal Database Replication Guide and Reference*, SC27-1121

DB2 Performance Expert for z/OS, Version 1

The following books are part of the DB2 Performance Expert library. Some of these books include information about the following tools: IBM DB2 Performance Expert for z/OS; IBM DB2 Performance Monitor for z/OS; and DB2 Buffer Pool Analyzer for z/OS.
- *OMEGAMON Buffer Pool Analyzer User's Guide*, SC18-7972
- *OMEGAMON Configuration and Customization*, SC18-7973
- *OMEGAMON Messages*, SC18-7974
- *OMEGAMON Monitoring Performance from ISPF*, SC18-7975
- *OMEGAMON Monitoring Performance from Performance Expert Client*, SC18-7976
- *OMEGAMON Program Directory*, GI10-8549
- *OMEGAMON Report Command Reference*, SC18-7977
- *OMEGAMON Report Reference*, SC18-7978
- *Using IBM Tivoli OMEGAMON XE on z/OS*, SC18-7979

DB2 Query Management Facility (QMF) Version 8.1
- *DB2 Query Management Facility: DB2 QMF High Performance Option User's Guide for TSO/CICS*, SC18-7450
- *DB2 Query Management Facility: DB2 QMF Messages and Codes*, GC18-7447
- *DB2 Query Management Facility: DB2 QMF Reference*, SC18-7446
- *DB2 Query Management Facility: Developing DB2 QMF Applications*, SC18-7651
- *DB2 Query Management Facility: Getting Started with DB2 QMF for Windows and DB2 QMF for WebSphere*, SC18-7449
- *DB2 Query Management Facility: Getting Started with DB2 QMF Query Miner*, GC18-7451
- *DB2 Query Management Facility: Installing and Managing DB2 QMF for TSO/CICS*, GC18-7444
- *DB2 Query Management Facility: Installing and Managing DB2 QMF for Windows and DB2 QMF for WebSphere*, GC18-7448

- *DB2 Query Management Facility: Introducing DB2 QMF*, GC18-7443
- *DB2 Query Management Facility: Using DB2 QMF*, SC18-7445
- *DB2 Query Management Facility: DB2 QMF Visionary Developer's Guide*, SC18-9093
- *DB2 Query Management Facility: DB2 QMF Visionary Getting Started Guide*, GC18-9092

DB2 Redbooks™

For access to all IBM Redbooks about DB2, see the IBM Redbooks Web page at www.ibm.com/redbooks

DB2 Server for VSE & VM
- *DB2 Server for VM: DBS Utility*, SC09-2983

DB2 Universal Database Cross-Platform information
- *IBM DB2 Universal Database SQL Reference for Cross-Platform Development*, available at www.ibm.com/software/data/developer/cpsqlref/

DB2 Universal Database for iSeries

The following books are available at www.ibm.com/iseries/infocenter
- *DB2 Universal Database for iSeries Performance and Query Optimization*
- *DB2 Universal Database for iSeries Database Programming*
- *DB2 Universal Database for iSeries SQL Programming Concepts*
- *DB2 Universal Database for iSeries SQL Programming with Host Languages*
- *DB2 Universal Database for iSeries SQL Reference*
- *DB2 Universal Database for iSeries Distributed Data Management*
- *DB2 Universal Database for iSeries Distributed Database Programming*

DB2 Universal Database for Linux, UNIX, and Windows:
- *DB2 Universal Database Administration Guide: Planning*, SC09-4822
- *DB2 Universal Database Administration Guide: Implementation*, SC09-4820
- *DB2 Universal Database Administration Guide: Performance*, SC09-4821
- *DB2 Universal Database Administrative API Reference*, SC09-4824
- *DB2 Universal Database Application Development Guide: Building and Running Applications*, SC09-4825
- *DB2 Universal Database Call Level Interface Guide and Reference, Volumes 1 and 2*, SC09-4849 and SC09-4850
- *DB2 Universal Database Command Reference*, SC09-4828
- *DB2 Universal Database SQL Reference Volume 1*, SC09-4844
- *DB2 Universal Database SQL Reference Volume 2*, SC09-4845

Device Support Facilities
- *Device Support Facilities User's Guide and Reference*, GC35-0033

DFSMS

These books provide information about a variety of components of DFSMS, including z/OS DFSMS, z/OS DFSMSdfp, z/OS DFSMSdss, z/OS DFSMShsm, and z/OS DFP.
- *z/OS DFSMS Access Method Services for Catalogs*, SC26-7394
- *z/OS DFSMSdss Storage Administration Guide*, SC35-0423
- *z/OS DFSMSdss Storage Administration Reference*, SC35-0424
- *z/OS DFSMShsm Managing Your Own Data*, SC35-0420
- *z/OS DFSMSdfp: Using DFSMSdfp in the z/OS Environment*, SC26-7473
- *z/OS DFSMSdfp Diagnosis Reference*, GY27-7618
- *z/OS DFSMS: Implementing System-Managed Storage*, SC27-7407
- *z/OS DFSMS: Macro Instructions for Data Sets*, SC26-7408
- *z/OS DFSMS: Managing Catalogs*, SC26-7409
- *z/OS MVS: Program Management User's Guide and Reference*, SA22-7643
- *z/OS MVS Program Management: Advanced Facilities*, SA22-7644
- *z/OS DFSMSdfp Storage Administration Reference*, SC26-7402
- *z/OS DFSMS: Using Data Sets*, SC26-7410
- *DFSMSdfp Advanced Services* , SC26-7400
- *DFSMS/MVS: Utilities*, SC26-7414

DFSORT™
- *DFSORT Application Programming: Guide*, SC33-4035
- *DFSORT Installation and Customization*, SC33-4034

Distributed Relational Database Architecture

- *Open Group Technical Standard*; the Open Group presently makes the following DRDA books available through its Web site at www.opengroup.org
  - *Open Group Technical Standard, DRDA Version 3 Vol. 1: Distributed Relational Database Architecture*
  - *Open Group Technical Standard, DRDA Version 3 Vol. 2: Formatted Data Object Content Architecture*
  - *Open Group Technical Standard, DRDA Version 3 Vol. 3: Distributed Data Management Architecture*

Domain Name System
- *DNS and BIND, Third Edition, Paul Albitz and Cricket Liu, O'Reilly*, ISBN 0-59600-158-4

Education
- Information about IBM educational offerings is available on the Web at http://www.ibm.com/software/sw-training/
- A collection of glossaries of IBM terms is available on the IBM Terminology Web site at www.ibm.com/ibm/terminology/index.html

eServer zSeries
- *IBM eServer zSeries Processor Resource/System Manager Planning Guide*, SB10-7033

Fortran: VS Fortran
- *VS Fortran Version 2: Language and Library Reference*, SC26-4221
- *VS Fortran Version 2: Programming Guide for CMS and MVS*, SC26-4222

High Level Assembler
- *High Level Assembler for MVS and VM and VSE Language Reference*, SC26-4940
- *High Level Assembler for MVS and VM and VSE Programmer's Guide*, SC26-4941

ICSF
- *z/OS ICSF Overview*, SA22-7519
- *Integrated Cryptographic Service Facility Administrator's Guide*, SA22-7521

IMS Version 8

IMS product information is available on the IMS Library Web page, which you can find at www.ibm.com/ims
- *IMS Administration Guide: System*, SC27-1284
- *IMS Administration Guide: Transaction Manager*, SC27-1285
- *IMS Application Programming: Database Manager*, SC27-1286
- *IMS Application Programming: Design Guide*, SC27-1287
- *IMS Application Programming: Transaction Manager*, SC27-1289
- *IMS Command Reference*, SC27-1291
- *IMS Customization Guide*, SC27-1294
- *IMS Install Volume 1: Installation Verification*, GC27-1297
- *IMS Install Volume 2: System Definition and Tailoring*, GC27-1298
- *IMS Messages and Codes Volumes 1 and 2*, GC27-1301 and GC27-1302
- *IMS Open Transaction Manager Access Guide and Reference*, SC18-7829
- *IMS Utilities Reference: System*, SC27-1309

General information about IMS Batch Terminal Simulator for z/OS is available on the Web at www.ibm.com/software/data/db2imstools/library.html

IMS DataPropagator
- *IMS DataPropagator for z/OS Administrator's Guide for Log*, SC27-1216
- *IMS DataPropagator: An Introduction*, GC27-1211
- *IMS DataPropagator for z/OS Reference*, SC27-1210

ISPF
- *z/OS ISPF Dialog Developer's Guide*, SC23-4821
- *z/OS ISPF Messages and Codes*, SC34-4815
- *z/OS ISPF Planning and Customizing*, GC34-4814
- *z/OS ISPF User's Guide Volumes 1 and 2*, SC34-4822 and SC34-4823

Language Environment
- *Debug Tool User's Guide and Reference*, SC18-7171
- *Debug Tool for z/OS and OS/390 Reference and Messages*, SC18-7172
- *z/OS Language Environment Concepts Guide*, SA22-7567
- *z/OS Language Environment Customization*, SA22-7564
- *z/OS Language Environment Debugging Guide*, GA22-7560
- *z/OS Language Environment Programming Guide*, SA22-7561
- *z/OS Language Environment Programming Reference*, SA22-7562

MQSeries
- *MQSeries Application Messaging Interface*, SC34-5604

- *MQSeries for OS/390 Concepts and Planning Guide*, GC34-5650
- *MQSeries for OS/390 System Setup Guide*, SC34-5651

National Language Support
- *National Language Design Guide Volume 1*, SE09-8001
- *IBM National Language Support Reference Manual Volume 2*, SE09-8002

NetView
- *Tivoli NetView for z/OS Installation: Getting Started*, SC31-8872
- *Tivoli NetView for z/OS User's Guide*, GC31-8849

Microsoft ODBC

Information about Microsoft ODBC is available at http://msdn.microsoft.com/library/

Parallel Sysplex Library
- *System/390 9672 Parallel Transaction Server, 9672 Parallel Enterprise Server, 9674 Coupling Facility System Overview For R1/R2/R3 Based Models*, SB10-7033
- *z/OS Parallel Sysplex Application Migration*, SA22-7662
- *z/OS Parallel Sysplex Overview: An Introduction to Data Sharing and Parallelism*, SA22-7661
- *z/OS Parallel Sysplex Test Report*, SA22-7663

The *Parallel Sysplex Configuration Assistant* is available at www.ibm.com/s390/pso/psotool

PL/I: Enterprise PL/I for z/OS
- *IBM Enterprise PL/I for z/OS Language Reference*, SC27-1460
- *IBM Enterprise PL/I for z/OS Programming Guide*, SC27-1457

PL/I: PL/I for MVS & VM
- *PL/I for MVS & VM Programming Guide*, SC26-3113

SMP/E
- *SMP/E for z/OS and OS/390 Reference*, SA22-7772
- *SMP/E for z/OS and OS/390 User's Guide*, SA22-7773

Storage Management
- *z/OS DFSMS: Implementing System-Managed Storage*, SC26-7407
- *MVS/ESA Storage Management Library: Managing Data*, SC26-7397
- *MVS/ESA Storage Management Library: Managing Storage Groups*, SC35-0421
- *MVS Storage Management Library: Storage Management Subsystem Migration Planning Guide*, GC26-7398

System Network Architecture (SNA)
- *SNA Formats*, GA27-3136
- *SNA LU 6.2 Peer Protocols Reference*, SC31-6808
- *SNA Transaction Programmer's Reference Manual for LU Type 6.2*, GC30-3084
- *SNA/Management Services Alert Implementation Guide*, GC31-6809

TCP/IP
- *IBM TCP/IP for MVS: Customization & Administration Guide*, SC31-7134
- *IBM TCP/IP for MVS: Diagnosis Guide*, LY43-0105
- *IBM TCP/IP for MVS: Messages and Codes*, SC31-7132
- *IBM TCP/IP for MVS: Planning and Migration Guide*, SC31-7189

TotalStorage™ Enterprise Storage Server
- *RAMAC Virtual Array: Implementing Peer-to-Peer Remote Copy*, SG24-5680
- *Enterprise Storage Server Introduction and Planning*, GC26-7444
- *IBM RAMAC Virtual Array*, SG24-6424

Unicode
- *z/OS Support for Unicode: Using Conversion Services*, SA22-7649

Information about Unicode, the Unicode consortium, the Unicode standard, and standards conformance requirements is available at www.unicode.org

VTAM
- *Planning for NetView, NCP, and VTAM*, SC31-8063
- *VTAM for MVS/ESA Diagnosis*, LY43-0078
- *VTAM for MVS/ESA Messages and Codes*, GC31-8369
- *VTAM for MVS/ESA Network Implementation Guide*, SC31-8370
- *VTAM for MVS/ESA Operation*, SC31-8372
- *z/OS Communications Server SNA Programming*, SC31-8829
- *z/OS Communicatons Server SNA Programmer's LU 6.2 Reference*, SC31-8810
- *VTAM for MVS/ESA Resource Definition Reference*, SC31-8377

WebSphere family

- *WebSphere MQ Integrator Broker: Administration Guide*, SC34-6171
- *WebSphere MQ Integrator Broker for z/OS: Customization and Administration Guide*, SC34-6175
- *WebSphere MQ Integrator Broker: Introduction and Planning*, GC34-5599
- *WebSphere MQ Integrator Broker: Using the Control Center*, SC34-6168

z/Architecture

- *z/Architecture Principles of Operation*, SA22-7832

z/OS

- *z/OS C/C++ Programming Guide*, SC09-4765
- *z/OS C/C++ Run-Time Library Reference*, SA22-7821
- *z/OS C/C++ User's Guide*, SC09-4767
- *z/OS Communications Server: IP Configuration Guide*, SC31-8875
- *z/OS DCE Administration Guide*, SC24-5904
- *z/OS DCE Introduction*, GC24-5911
- *z/OS DCE Messages and Codes*, SC24-5912
- *z/OS Information Roadmap*, SA22-7500
- *z/OS Introduction and Release Guide*, GA22-7502
- *z/OS JES2 Initialization and Tuning Guide*, SA22-7532
- *z/OS JES3 Initialization and Tuning Guide*, SA22-7549
- *z/OS Language Environment Concepts Guide*, SA22-7567
- *z/OS Language Environment Customization*, SA22-7564
- *z/OS Language Environment Debugging Guide*, GA22-7560
- *z/OS Language Environment Programming Guide*, SA22-7561
- *z/OS Language Environment Programming Reference*, SA22-7562
- *z/OS Managed System Infrastructure for Setup User's Guide*, SC33-7985
- *z/OS MVS Diagnosis: Procedures*, GA22-7587
- *z/OS MVS Diagnosis: Reference*, GA22-7588
- *z/OS MVS Diagnosis: Tools and Service Aids*, GA22-7589
- *z/OS MVS Initialization and Tuning Guide*, SA22-7591
- *z/OS MVS Initialization and Tuning Reference*, SA22-7592
- *z/OS MVS Installation Exits*, SA22-7593
- *z/OS MVS JCL Reference*, SA22-7597
- *z/OS MVS JCL User's Guide*, SA22-7598
- *z/OS MVS Planning: Global Resource Serialization*, SA22-7600
- *z/OS MVS Planning: Operations*, SA22-7601
- *z/OS MVS Planning: Workload Management*, SA22-7602
- *z/OS MVS Programming: Assembler Services Guide*, SA22-7605
- *z/OS MVS Programming: Assembler Services Reference, Volumes 1 and 2*, SA22-7606 and SA22-7607
- *z/OS MVS Programming: Authorized Assembler Services Guide*, SA22-7608
- *z/OS MVS Programming: Authorized Assembler Services Reference Volumes 1-4*, SA22-7609, SA22-7610, SA22-7611, and SA22-7612
- *z/OS MVS Programming: Callable Services for High-Level Languages*, SA22-7613
- *z/OS MVS Programming: Extended Addressability Guide*, SA22-7614
- *z/OS MVS Programming: Sysplex Services Guide*, SA22-7617
- *z/OS MVS Programming: Sysplex Services Reference*, SA22-7618
- *z/OS MVS Programming: Workload Management Services*, SA22-7619
- *z/OS MVS Recovery and Reconfiguration Guide*, SA22-7623
- *z/OS MVS Routing and Descriptor Codes*, SA22-7624
- *z/OS MVS Setting Up a Sysplex*, SA22-7625
- *z/OS MVS System Codes* SA22-7626
- *z/OS MVS System Commands*, SA22-7627
- *z/OS MVS System Messages Volumes 1-10*, SA22-7631, SA22-7632, SA22-7633, SA22-7634, SA22-7635, SA22-7636, SA22-7637, SA22-7638, SA22-7639, and SA22-7640
- *z/OS MVS Using the Subsystem Interface*, SA22-7642
- *z/OS Planning for Multilevel Security and the Common Criteria*, SA22-7509
- *z/OS RMF User's Guide*, SC33-7990
- *z/OS Security Server Network Authentication Server Administration*, SC24-5926
- *z/OS Security Server RACF Auditor's Guide*, SA22-7684
- *z/OS Security Server RACF Command Language Reference*, SA22-7687
- *z/OS Security Server RACF Macros and Interfaces*, SA22-7682
- *z/OS Security Server RACF Security Administrator's Guide*, SA22-7683
- *z/OS Security Server RACF System Programmer's Guide*, SA22-7681
- *z/OS Security Server RACROUTE Macro Reference*, SA22-7692
- *z/OS Support for Unicode: Using Conversion Services*, SA22-7649
- *z/OS TSO/E CLISTs*, SA22-7781
- *z/OS TSO/E Command Reference*, SA22-7782

- *z/OS TSO/E Customization*, SA22-7783
- *z/OS TSO/E Messages*, SA22-7786
- *z/OS TSO/E Programming Guide*, SA22-7788
- *z/OS TSO/E Programming Services*, SA22-7789
- *z/OS TSO/E REXX Reference*, SA22-7790
- *z/OS TSO/E User's Guide*, SA22-7794
- *z/OS UNIX System Services Command Reference*, SA22-7802
- *z/OS UNIX System Services Messages and Codes*, SA22-7807
- *z/OS UNIX System Services Planning*, GA22-7800
- *z/OS UNIX System Services Programming: Assembler Callable Services Reference*, SA22-7803
- *z/OS UNIX System Services User's Guide*, SA22-7801

Index

Special characters

_ (underscore)
  in DDL registration tables 219
% (percent sign)
  in DDL registration tables 219

Numerics

16-KB page size 113
32-KB page size 113
8-KB page size 113

A

abend
  AEY9 525
  after SQLCODE -923 530
  ASP7 525
  backward log recovery 607
  CICS
    abnormal termination of application 525
    scenario 530
    transaction abends when disconnecting from DB2 386, 387
    waits 525
  current status rebuild 595
  disconnects DB2 396
  DXR122E 517
  effects of 444
  forward log recovery 602
  IMS
    U3047 524
    U3051 524
  IMS, scenario 522, 524
  IRLM
    scenario 517
    stop command 378
    stop DB2 377
  log
    damage 591
    initialization 594
    lost information 613
  page problem 612
  restart 593
  starting DB2 after 326
  VVDS (VSAM volume data set)
    destroyed 547
    out of space 547
acceptance option 243
access control
  authorization exit routine 1061
  closed application 215, 227
  DB2 subsystem
    local 131, 232
    process overview 231
    RACF 131
    remote 132, 238
  external DB2 data sets 132
  field level 143
  internal DB2 data 129
access method services
  bootstrap data set definition 437
  commands
    ALTER 38, 550
    ALTER ADDVOLUMES 31, 38
    ALTER REMOVEVOLUMES 31
    DEFINE 38, 613
    DEFINE CLUSTER 38, 40, 657
    DELETE CLUSTER 38
    EXPORT 483
    IMPORT 107, 611
    PRINT 504
    REPRO 504, 539
  data set management 29, 38
  delete damaged BSDS 538
  redefine user work file 493
  rename damaged BSDS 538
  table space re-creation 613
access path
  direct row access 942
  hints 801
  index access 916
  index-only access 942
  low cluster ratio
    effects of 917
    suggests table space scan 947
    with list prefetch 971
  multiple index access
    description 952
    disabling 683
    PLAN_TABLE 940
  selection
    influencing with SQL 788
    problems 745
    queries containing host variables 774
    Visual Explain 789, 927
  table space scan 947
  unique index with matching value 954
access profile, in RACF 266
accounting
  elapsed times 662
  trace
    description 1190
ACQUIRE
  option of BIND PLAN subcommand
    locking tables and table spaces 843
    thread creation 730
active log
  data set
    changing high-level qualifier for 99
    copying with IDCAMS REPRO statement 438
    effect of stopped 535
    offloaded to archive log 425
    placement 692
    VSAM linear 1116
  description 8
  dual logging 426
  offloading 426
  problems 531
  recovery scenario 531

© Copyright IBM Corp. 1982, 2008

active log *(continued)*
size
determining 698
tuning considerations 698
truncation 427
writing 425
activity sample table 1031
ADD VOLUMES clause of ALTER STOGROUP statement 68
address space
DB2 13
started-task 267
stored procedures 268
ADMIN_TASK_ADD 339
ADMIN_TASK_REMOVE 349
administrative authority 138
administrative task scheduler
adding a task 335
ADMIN_TASK_ADD 339
ADMIN_TASK_REMOVE 349
architecture 355
architecture in a data sharing environment 356
data sharing
specifying a scheduler 339
synchronization 351
disabling tracing 352
enabling tracing 352
how stored procedures are executed 360
interacting 335
JCL jobs 361
life cycle 356
listing scheduled tasks 347
listing task status 347
multi-threaded execution of tasks 359
overview 335
protecting resources 358
protecting the interface 357
recovering the task list 353
removing a task 348
sample schedule definitions 337
scheduling capabilities 335
secure execution of tasks 358
security 357
SQL code returned by stored procedure 354
SQL code returned by user-defined table function 354
starting 351
stopping 351
task did not execute 353
task execution 359
task execution in a data sharing environment 361
task lists 356
troubleshooting 352
Unicode restrictions 361
user roles 357
alias
ownership 146
qualified name 146
ALL
clause of GRANT statement 134
ALL PRIVILEGES clause
GRANT statement 137
allocating space
effect on INSERTs 658
preformatting 658
table 37
allocating storage
dictionary 116
table 112
already-verified acceptance option 244
ALTER command of access method services 550
ALTER command, access method services
FOR option 38
TO option 38
ALTER DATABASE statement
usage 68
ALTER FUNCTION statement
usage 96
ALTER privilege
description 134
ALTER PROCEDURE statement 96
ALTER STOGROUP statement 67
ALTER TABLE statement
AUDIT clause 288
description 71
ALTER TABLESPACE statement
description 69
ALTERIN privilege
description 136
ambiguous cursor 856, 1008
APPL statement
options
SECACPT 243
application plan
controlling application connections 385
controlling use of DDL 215, 227
inoperative, when privilege is revoked 186
invalidated
dropping a table 89
dropping a view 96
dropping an index 94
when privilege is revoked 186
list of dependent objects 90
monitoring 1197
privileges
explicit 137
of ownership 146
retrieving catalog information 190
application program
coding SQL statements
data communication coding 16
error checking in IMS 329
internal integrity reports 298
recovery scenarios
CICS 525
IMS 524
running
batch 329
CAF (call attachment facility) 330
CICS transactions 329
error recovery scenario 520, 521
IMS 328
RRSAF (Resource Recovery Services attachment facility) 331
TSO online 327
security measures in 153
suspension
description 810
timeout periods 834
application programmer
description 171
privileges 177
application registration table (ART)
See registration tables for DDL
archive log
BSDS 436

archive log *(continued)*
data set
changing high-level qualifier for 99
description 428
offloading 425
types 428
deleting 438
description 8
device type 429
dual logging 429
dynamic allocation of data sets 428
multivolume data sets 430
recovery scenario 535
retention period 438
SMS 430
writing 426
ARCHIVE LOG command
cancels offloads 432
use 430
ARCHIVE LOG FREQ field of panel DSNTIPL 698
ARCHIVE privilege
description 136
archiving to disk volumes 429
ARCHWTOR option of DSNZPxxx module 427
ART (application registration table)
See registration tables for DDL
ASUTIME column
resource limit specification table (RLST) 720
asynchronous data from IFI 1172
attachment facility
description 14
attachment request
come-from check 248
controlling 243
definition 242
translating IDs 247, 260
using secondary IDs 249
AUDIT
clause of ALTER TABLE statement 288
clause of CREATE TABLE statement 288
option of START TRACE command 286
audit trace
class descriptions 287
controlling 286
description 285, 1192
records 290
auditing
access attempts 285, 292
authorization IDs 290
classes of events 287
data 1192
description 127
in sample security plan
attempted access 310
payroll data 304
payroll updates 307
reporting trace records 290
security measures in force 290
table access 288
trace data through IFI 1183
authority
administrative 138
controlling access to
CICS 329
DB2 catalog and directory 144
DB2 commands 323
DB2 functions 323
authority *(continued)*
controlling access to *(continued)*
IMS application program 329
TSO application program 328
description 130, 133
explicitly granted 138, 145
hierarchy 138
level SYS for z/OS command group 320
levels 323
types
DBADM 141
DBCTRL 140
DBMAINT 140
installation SYSADM 142
installation SYSOPR 140
PACKADM 140
SYSADM 142
SYSCTRL 141
SYSOPR 140
authorization
data definition statements, to use 215
exit routines.
See connection exit routine
See sign-on exit routine
authorization ID
auditing 290
checking during thread creation 730
description 134
dynamic SQL, determining 164
exit routine input parameter 1054
inbound from remote location
See remote request
initial
connection processing 233
sign-on processing 236
package execution 149
primary
connection processing 233, 234
description 134
exit routine input 1054
privileges exercised by 161
sign-on processing 236, 237
retrieving catalog information 189
routine, determining 161
secondary
attachment requests 249
connection processing 234
description 134
exit routine output 1056, 1073
identifying RACF groups 272
ownership held by 148
privileges exercised by 161
sign-on processing 237
SQL
changing 134
description 134
exit routine output 1056, 1073
privileges exercised by 161
system-directed access 150
translating
inbound IDs 247
outbound IDs 260
verifying 243
automatic
data management 477
deletion of archive log data sets 438

automatic *(continued)*
rebind
EXPLAIN processing 937
restart function of z/OS 449
automatic query rewrite
definition 881
description of process 891
determining occurrence of 896
enabling query optimization 889
examples 892
exploiting 889
introduction 881, 889
query requirements 891
auxiliary storage 29
auxiliary table
LOCK TABLE statement 864
availability
recovering
data sets 491
page sets 491
recovery planning 471
summary of functions for 12
AVGKEYLEN column
SYSINDEXES catalog table 901
SYSINDEXPART catalog table 902
AVGROWLEN column
SYSTABLEPART catalog table 904
SYSTABLES catalog table
data collected by RUNSTATS utility 905
SYSTABLES_HIST catalog table 911
SYSTABLESPACE catalog table 905
AVGSIZE column
SYSLOBSTATS catalog table 903

B

BACKOUT DURATION field of panel DSNTIPL 450
backup
data set using DFSMShsm 477
database
concepts 471
DSN1COPY 498
image copies 489
planning 471
system procedures 471
BACKUP SYSTEM utility 37
backward log recovery phase
recovery scenario 607
restart 448
base table
distinctions from temporary tables 48
basic direct access method (BDAM)
See BDAM (basic direct access method)
basic sequential access method (BSAM)
See BSAM (basic sequential access method)
batch message processing (BMP) program
See BMP (batch message processing) program
batch processing
TSO 329
BDAM (basic direct access method) 428
BIND PACKAGE subcommand of DSN
options
DISABLE 154
ENABLE 154
ISOLATION 846
OWNER 148
RELEASE 843
BIND PACKAGE subcommand of DSN *(continued)*
options *(continued)*
REOPT(ALWAYS) 774
REOPT(NONE) 774
REOPT(ONCE) 774
privileges for remote bind 154
BIND PLAN subcommand of DSN
options
ACQUIRE 843
DISABLE 154
ENABLE 154
ISOLATION 846
OWNER 148
RELEASE 843
REOPT(ALWAYS) 774
REOPT(NONE) 774
REOPT(ONCE) 774
BIND privilege
description 137
BINDADD privilege
description 136
BINDAGENT privilege
description 136
naming plan or package owner 148
binding
privileges needed 164
bit data
altering subtype 88
blank
column with a field procedure 1091
block fetch
description 1005
enabling 1007
LOB data impact 1007
scrollable cursors 1007
BMP (batch message processing) program
connecting from dependent regions 394
bootstrap data set (BSDS)
See BSDS (bootstrap data set)
BSAM (basic sequential access method)
reading archive log data sets 428
BSDS (bootstrap data set)
archive log information 437
changing high-level qualifier of 99
changing log inventory 438
defining 436
description 9
dual copies 436
dual recovery 539
failure symptoms 593
logging considerations 696
managing 436
recovery scenario 538, 610
registers log data 436
restart use 444
restoring from the archive log 539
single recovery 539
stand-alone log services role 1126
BSDS privilege
description 136
buffer information area used in IFI 1155
buffer pool
advantages of large pools 673
advantages of multiple pools 673
allocating storage 673
altering attributes 675
available pages 666

buffer pool *(continued)*
considerations 707
description 9
displaying current status 675
hit ratio 671
immediate writes 679
in-use pages 666
long-term page fix option 675
monitoring 677, 1196
page-stealing algorithm 674
read operations 666
size 672, 732
statistics 677
thresholds 667, 679
update efficiency 678
updated pages 666
use in logging 425
write efficiency 678
write operations 666
BUFFERPOOL clause
ALTER INDEX statement 667
ALTER TABLESPACE statement 667
CREATE DATABASE statement 667
CREATE INDEX statement 667
CREATE TABLESPACE statement 667
BUFFERPOOL privilege
description 138
built-in functions for encryption
See encryption

C

cache
dynamic SQL
effect of RELEASE(DEALLOCATE) 844
cache controller 696
cache for authorization IDs 152
CAF (call attachment facility)
application program
running 330
submitting 331
description 18
DSNALI language interface module 1153
call attachment facility (CAF)
See CAF (call attachment facility)
CANCEL THREAD command
CICS threads 386
disconnecting from TSO 383
use in controlling DB2 connections 411
capturing changed data
altering a table for 87
CARDF column
SYSCOLDIST catalog table
access path selection 900
data collected by RUNSTATS utility 900
SYSCOLDIST_HIST catalog table 909
SYSINDEXPART catalog table
data collected by RUNSTATS utility 902
SYSINDEXPART_HIST catalog table 910
SYSTABLEPART catalog table
data collected by RUNSTATS utility 904
SYSTABLEPART_HIST catalog table 911
SYSTABLES catalog table
data collected by RUNSTATS utility 905
SYSTABLES_HIST catalog table 911
SYSTABSTATS catalog table
data collected by RUNSTATS utility 906
CARDF column *(continued)*
SYSTABSTATS_HIST catalog table 911
CARDINALITY column of SYSROUTINES catalog table 904
cardinality of user-defined table function
improving query performance 793
Cartesian join 958
catalog statistics
history 909, 913
influencing access paths 799
catalog tables
frequency of image copies 476
historical statistics 909, 913
retrieving information about
multiple grants 188
plans and packages 190
privileges 187
routines 190
SYSCOLAUTH 187
SYSCOLDIST
data collected by RUNSTATS utility 900
SYSCOLDIST_HIST 909
SYSCOLDISTSTATS
data collected by RUNSTATS utility 900
SYSCOLSTATS
data collected by RUNSTATS utility 900
SYSCOLUMNS
column description of a value 1090
data collected by RUNSTATS utility 901
updated by ALTER TABLE statement 71
updated by DROP TABLE 89
SYSCOLUMNS_HIST 909
SYSCOPY
discarding records 510
holds image copy information 480
image copy in log 1113
used by RECOVER utility 474
SYSDBAUTH 187
SYSINDEXES
access path selection 916
data collected by RUNSTATS utility 901
dropping a table 90
SYSINDEXES_HIST 909
SYSINDEXPART
data collected by RUNSTATS utility 902
space allocation information 41
SYSINDEXPART_HIST 910
SYSINDEXSTATS
data collected by RUNSTATS utility 903
SYSINDEXSTATS_HIST 910
SYSLOBSTATS
data collected by RUNSTATS utility 903
SYSLOBSTATS_HIST 910
SYSPACKAUTH 187
SYSPLANAUTH
checked during thread creation 730
plan authorization 187
SYSPLANDEP 90
SYSRESAUTH 187
SYSROUTINES
data collected by RUNSTATS utility 904
using EXTERNAL_SECURITY column of 277
SYSSTOGROUP
storage groups 29
SYSSTRINGS
establishing conversion procedure 1086
SYSSYNONYMS 89

catalog tables *(continued)*
SYSTABAUTH
authorization information 187
dropping a table 90
SYSTABLEPART
data collected by RUNSTATS utility 904
PAGESAVE column 705
table spaces associated with storage group 68
updated by LOAD and REORG utilities for data compression 705
SYSTABLEPART_HIST 911
SYSTABLES
data collected by RUNSTATS utility 905
updated by ALTER TABLE statement 71
updated by DROP TABLE 89
updated by LOAD and REORG for data compression 705
SYSTABLES_HIST 911
SYSTABLESPACE
data collected by RUNSTATS utility 905
implicitly created table spaces 45
SYSTABSTATS
data collected by RUNSTATS utility 906
PCTROWCOMP column 705
SYSUSERAUTH 187
SYSVIEWDEP
view dependencies 90
SYSVOLUMES 29
SYTABSTATS_HIST 911
views of 191
catalog, DB2
authority for access 144
changing high-level qualifier 102
description 7
DSNDB06 database 480
locks 824
point in time recovery 494
recovery 494
recovery scenario 546
statistics
production system 923
querying the catalog 916
tuning 693
CDB (communications database)
backing up 473
changing high-level qualifier 102
description 8
updating tables 247
CHANGE command of IMS
purging residual recovery entries 388
change log inventory utility
changing
BSDS 376, 438
control of data set access 282
change number of sessions (CNOS)
See CNOS (change number of sessions)
CHANGE SUBSYS command of IMS 393
CHARACTER data type
altering 88
CHECK DATA utility
checks referential constraints 297
CHECK INDEX utility
checks consistency of indexes 297
checkpoint
log records 1111, 1115
queue 454
CHECKPOINT FREQ field of panel DSNTIPN 698
CI (control interval)
description 425, 428
reading 1126
CICS
commands
accessing databases 384
DSNC DISCONNECT 386
DSNC DISPLAY PLAN 386
DSNC DISPLAY TRANSACTION 386
DSNC STOP 387
response destination 322
used in DB2 environment 317
connecting to
controlling 388
disconnecting applications 386
thread 386
connecting to DB2
authorization IDs 329
connection processing 233
controlling 383
disconnecting applications 421
sample authorization routines 235
security 279
sign-on processing 236
supplying secondary IDs 233
correlating DB2 and CICS accounting records 650
description, attachment facility 15
disconnecting from DB2 387
dynamic plan selection
exit routine 1103
facilities 1103
diagnostic trace 420
monitoring facility (CMF) 642, 1185
tools 1187
language interface module (DSNCLI)
IFI entry point 1153
running CICS applications 329
operating
entering DB2 commands 321
identify outstanding indoubt units 461
recovery from system failure 16
terminates AEY9 531
planning
DB2 considerations 15
environment 329
programming
applications 329
recovery scenarios
application failure 525
attachment facility failure 530
CICS not operational 525
DB2 connection failure 526
indoubt resolution failure 527
starting a connection 384
statistics 1185
system administration 16
two-phase commit 455
XRF (extended recovery facility) 16
CICS transaction invocation stored procedure 1104
claim
class 865
definition 865
effect of cursor WITH HOLD 857
Class 1 elapsed time 642
CLOSE
clause of CREATE INDEX statement
effect on virtual storage use 706

CLOSE *(continued)*
clause of CREATE TABLESPACE statement
deferred close 691
effect on virtual storage use 706
closed application
controlling access 215, 227
definition 215
cluster ratio
description 917
effects
low cluster ratio 917
table space scan 947
with list prefetch 971
CLUSTERED column of SYSINDEXES catalog table
data collected by RUNSTATS utility 901
CLUSTERING column
SYSINDEXES_HIST catalog table 910
CLUSTERING column of SYSINDEXES catalog table
access path selection 901
CLUSTERRATIO column
SYSINDEXSTATS_HIST catalog table 910
CLUSTERRATIOF column
SYSINDEXES catalog table
data collected by RUNSTATS utility 902
SYSINDEXES_HIST catalog table 910
SYSINDEXSTATS catalog table
access path selection 903
CNOS (change number of sessions)
failure 556
coding
exit routines
general rules 1104
parameters 1106
COLCARD column of SYSCOLSTATS catalog table
data collected by RUNSTATS utility 901
updating 914
COLCARDDATA column of SYSCOLSTATS catalog table 901
COLCARDF column
SYSCOLUMNS catalog table 901
SYSCOLUMNS_HIST catalog table 909
COLCARDF column of SYSCOLUMNS catalog table
statistics not exact 906
updating 914
cold start
See also conditional restart
bypassing the damaged log 592
recovery operations during 453
special situations 613
COLGROUPCOLNO column
SYSCOLDIST catalog table
access path selection 900
SYSCOLDIST_HIST catalog table 909
SYSCOLDISTSTATS catalog table
data collected by RUNSTATS utility 900
collection, package
administrator 171
privileges on 136
column
adding to a table 71
description 6
dropping from a table 88
column description of a value 1090
column value descriptor (CVD) 1092
COLVALUE column
SYSCOLDIST catalog table
access path selection 900
SYSCOLDIST_HIST catalog table 909
COLVALUE column *(continued)*
SYSCOLDISTSTATS catalog table
data collected by RUNSTATS utility 900
come-from check 248
command prefix
messages 332
multi-character 320
usage 320
command recognition character (CRC)
See CRC (command recognition character)
commands
concurrency 809, 864
entering 317
issuing DB2 commands from IFI 1154
operator 318, 324
prefixes 333
commit
two-phase process 455
communications database (CDB)
See CDB (communications database)
compatibility
locks 823
compressing data
See data compression
compression dictionary
See data compression, dictionary
concurrency
commands 809, 864
contention independent of databases 825
control by drains and claims 864
control by locks 810
description 810
effect of
ISOLATION options 851, 852
lock escalation 831
lock size 820
LOCKSIZE options 838
row locks 839
uncommitted read 850
recommendations 812
utilities 809, 864
utility compatibility 867
with real-time statistics 1253
conditional restart
control record
backward log recovery failure 609
current status rebuild failure 601
forward log recovery failure 607
log initialization failure 601
wrap-around queue 454
description 451
excessive loss of active log data, restart procedure 614
total loss of log, restart procedure 613
connection
controlling CICS 383
controlling IMS 388
DB2
controlling commands 383
thread 743
displaying
IMS activity 394
effect of lost, on restart 459
exit routine
See connection exit routine
exit routine.
See connection exit routine

connection *(continued)*
IDs
cited in message DSNR007I 446
outstanding unit of recovery 446
used by IMS 329
used to identify a unit of recovery 521
processing
See connection processing
requests
exit point 1053
initial primary authorization ID 233
invoking RACF 233
local 232
VTAM 267
connection exit routine
debugging 1059
default 233, 234
description 1051
performance considerations 1058
sample
location 1052
provides secondary IDs 234, 1057
secondary authorization ID 234
using 233
writing 1051
connection processing
choosing for remote requests 243
initial primary authorization ID 233, 1056
invoking RACF 233
supplying secondary IDs 234
usage 232
using exit routine 233
continuous block fetch
See block fetch
continuous operation
recovering table spaces and data sets 491
recovery planning 12, 471
control interval (CI)
See CI (control interval)
control interval, sizing 28
control region, IMS 393
conversation acceptance option 243, 244
conversation-level security 243
conversion procedure
description 1086
writing 1086
coordinator
in multi-site update 467
in two-phase commit 455
copy pools
creating 37
SMS construct 37
COPY privilege
description 136
COPY utility
backing up 498
copying data from table space 489
DFSMSdss concurrent copy 481, 491
effect on real-time statistics 1250
restoring data 498
COPY-pending status
resetting 63
copying
a DB2 subsystem 109
a package, privileges for 154, 164
a relational database 109
correlated subqueries 780
correlation ID
CICS 528
duplicate 392, 529
identifier for connections from TSO 382
IMS 392
outstanding unit of recovery 446
RECOVER INDOUBT command 385, 391, 399
COST_CATEGORY_B column of RLST 721
CP processing, disabling parallel operations 669
CRC (command recognition character)
description 321
CREATE DATABASE statement
description 43
privileges required 164
CREATE GLOBAL TEMPORARY TABLE statement
distinctions from base tables 48
CREATE IN privilege
description 136
CREATE INDEX statement
privileges required 164
USING clause 41
CREATE SCHEMA statement 58
CREATE STOGROUP statement
description 29
privileges required 164
VOLUMES('*') attribute 30, 35
CREATE TABLE statement
AUDIT clause 288
creating a table space implicitly 45
privileges required 164
test table 64
CREATE TABLESPACE statement
creating a table space explicitly 44
deferring allocation of data sets 30
DEFINE NO clause 30, 44
DSSIZE option 42
privileges required 164
USING STOGROUP clause 30, 44
CREATE VIEW statement
privileges required 164
CREATEALIAS privilege
description 136
created temporary table
distinctions from base tables 48
table space scan 947
CREATEDBA privilege
description 136
CREATEDBC privilege
description 136
CREATEIN privilege
description 136
CREATESG privilege
description 136
CREATETAB privilege
description 136
CREATETMTAB privilege
description 136
CREATETS privilege
description 136
cron format 346
CS (cursor stability)
claim class 865
distributed environment 846
drain lock 866
effect on locking 846
optimistic concurrency control 848
page and row locking 848

CURRENDATA option of BIND
plan and package options differ 856
CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register 890
CURRENT REFRESH AGE special register 890
current status rebuild
phase of restart 446
recovery scenario 593
CURRENTDATA option
BIND PACKAGE subcommand
enabling block fetch 1008
BIND PLAN subcommand 1008
cursor
ambiguous 856, 1008
defined WITH HOLD, subsystem parameter to release locks 841
WITH HOLD
claims 857
locks 857
Customer Information Control System (CICS)
See CICS
CVD (column value descriptor) 1092, 1094

D

damage, heuristic 464
data
See also mixed data
access control
description 128
field-level 143
using option of START DB2 325
backing up 498
checking consistency of updates 297
coding
conversion procedures 1086
date and time exit routines 1083
edit exit routines 1077
field procedures 1089
compression
See data compression
consistency
ensuring 293
verifying 296, 299
definition control support
See data definition control support
effect of locks on integrity 810
encrypting 1077
improving access 927
loading into tables 61
moving 105
restoring 498
understanding access 927
DATA CAPTURE clause
ALTER TABLE statement 87
data class
assigning data sets to 42
SMS construct 42
data compression
determining effectiveness 704
dictionary
description 116, 703
estimating disk storage 116
estimating virtual storage 117
DSN1COMP utility 704
edit routine 1077
effect on log records 1112
data compression *(continued)*
Huffman 1077
logging 425
performance considerations 702
data definition control support
bypassing 228
controlling by
application name 220
application name with exceptions 221
object name 222
object name with exceptions 224
description 215
registration tables
See registration tables for DDL
restarting 228
stopping 228
Data Facility Product (DFSMSdfp) 107
data management threshold (DMTH) 668
data mirror 576
data set
adding 552
adding groups to control 281
allocation and extension 731
backing up using DFSMS 491
changing high-level qualifier 98
closing 690
control over creating 283
controlling access 281
copying 489
DSMAX value 687
extending 550
generic profiles 281, 283
limit 687
managing
defining your own 27
migrating to DFSMShsm 35
reasons for defining your own 37
reasons for using DB2 27
using access method services 38
using DB2 storage groups 27
using DFSMShsm 27, 35
with DB2 storage groups 27
your own 37
monitoring I/O activity 693
naming convention 38
number 39
open 687, 731
partition of
partitioned partitioning index 38
partitioned secondary index 38
partitioned table space 38
recovering
using non-DB2 dump 504
renaming 486
table space, deferring allocation 30
Data Set Services (DFSMSdss) 107
data set, DB2-managed
extending
avoiding failure 31
conditions for 31
description 31
insufficient space 31
nonpartitioned spaces 31
partitioned spaces 31
primary space allocation 32
example 34
secondary space allocation 32

data set, DB2-managed *(continued)*
example 34
data sharing
real-time statistics 1253
using IFI 1179
data structure
hierarchy 4
types 3
data type
altering 88
codes for numeric data 1110
subtypes 88
database
access thread
differences from allied threads 735
failure 555
security failures in 557
altering
definition 68
design 67
backup
copying data 489
planning 471
balancing 296
controlling access 400
creating 43
default database
See default database (DSNDB04)
description 5
dropping 68
DSNDB07 (work file database).
See work file database
implementing design 43
monitoring 365
operations that lock 43
page set control log records 1116
privileges
administrator 171, 176
controller 176
description 136
ownership 146
reasons for defining 43
recovery
description 491
failure scenarios 542
planning 471
RECOVER TOCOPY 500
RECOVER TOLOGPOINT 500
RECOVER TORBA 501
starting 364
starting and stopping as unit 43
status information 365
stopping 371
TEMP for declared temporary tables 44
users who need their own 44
database controller privileges 176
database descriptor (DBD)
See DBD (database descriptor)
database exception table, log records
exception states 1112
image copies of special table spaces 1112
LPL 1116
WEPR 1116
DataPropagator NonRelational (DPropNR)
See DPropNR (Data Propagator
DataRefresher 65
DATE FORMAT field of panel DSNTIPF 1084
date routine
DATE FORMAT field at installation 1084
description 1083
LOCAL DATE LENGTH field at installation 1084
writing 1083
datetime
exit routine for.
See date routine
See time routine
format
table 1083
DB2 Buffer Pool Analyzer
description 1196
DB2 coded format for numeric data 1110
DB2 commands
authority 323
authorized for SYSOPR 324
commands
RECOVER INDOUBT 465
RESET INDOUBT 466
START DB2 325
START DDF 401
STOP DDF 417
STOP DDF MODE(SUSPEND) 401
description 318
destination of responses 322
entering from
CICS 321
DSN session 328
IMS 320
TSO 321
z/OS 320
issuing from IFI 1154, 1156
users authorized to enter 323
DB2 Connect 18
DB2 data set statistics
obtaining through IFCID 0199 1168
DB2 DataPropagator
altering a table for 87
DB2 decoded procedure for numeric data 1110
DB2 Interactive (DB2I)
See DB2I (DB2 Interactive)
DB2 Performance Monitor (DB2 PM)
See DB2 Performance Expert
DB2 PM (DB2 Performance Monitor)
EXPLAIN 926
DB2 private protocol access
description 1003
resource limit facility 726
DB2-managed objects, changing data set high-level qualifier 104
DB2I (DB2 Interactive)
description 11, 327
panels
description 17
used to connect from TSO 381
DBA (database administrator)
description 171
sample privileges 176
DBADM authority
description 141
DBCTRL authority
description 140
DBD (database descriptor)
contents 8
EDM pool 679, 681
freeing 733

DBD (database descriptor) *(continued)*
load
in EDM pool 731
using ACQUIRE(ALLOCATE) 730
locks on 825
use count 733
DBD01 directory table space
contents 8
placement of data sets 692
quiescing 482
recovery after conditional restart 496
recovery information 480
DBFULTA0 (Fast Path Log Analysis Utility) 1185
DBMAINT authority
description 140
DD limit
See DSMAX
DDCS (data definition control support)
database 9
DDF (distributed data facility)
block fetch 1005
controlling connections 400
description 18
resuming 401
suspending 401
DDL, controlling usage of
See data definition control support
deadlock
description 811
detection scenarios 877
example 811
recommendation for avoiding 814
row vs. page locks 839
wait time calculation 835
with RELEASE(DEALLOCATE) 815
X'00C90088' reason code in SQLCA 812
DEADLOCK TIME field of panel DSNTIPJ 833
DEADLOK option of START irlmproc command 832
decision, heuristic 464
DECLARE GLOBAL TEMPORARY TABLE statement
distinctions from base tables 48
declared temporary table
distinctions from base tables 48
DECRYPT_BIT 209, 210
DECRYPT_CHAR 209, 210
DECRYPT_DB 209, 210
decryption
See encryption
dedicated virtual memory pool 967
default database (DSNDB04)
changing high-level qualifier 102
defining 5
DEFER ALL field of panel DSNTIPS 450
deferred close 687
deferred write threshold (DWQT)
description 669
recommendation for LOBs 671
DEFINE CLUSTER command of access method services 657
DEFINE CLUSTER command, access method services
defining extents 40
example of use 40
LINEAR option 40
REUSE option 40
SHAREOPTIONS 40
DEFINE command of access method services
recreating table space 613
redefine user work file 493
DEFINE command, access method services
FOR option 38
TO option 38
definer, description 155
DELETE
command of access method services 613
statement
validation routine 1081
DELETE privilege
description 134
deleting
archive logs 438
department sample table
description 1032
dependent
regions, disconnecting from 396
DFSLI000 (IMS language interface module) 329, 1153
DFSMS (Data Facility Storage Management Subsystem)
archive log data sets 430
backup 491
concurrent copy
backup 491
description 20
recovery 491
DFSMSdfp (Data Facility Product) 107
DFSMSdfp partitioned data set extended (PDSE) 20
DFSMSdss (Data Set Services) 107
DFSMShsm
assigning indexes to data classes 35
assigning table spaces to data classes 35
migrating data sets 35
recalling archive logs 36
using with BACKUP SYSTEM utility 37
using with RECOVER utility 36
DFSMShsm (Data Facility Hierarchical Storage Manager)
advantages 35
backup 477
moving data sets 107
recovery 477
DFSxxxx messages 332
dictionary
See data compression
direct row access 942
directory
authority for access 144
changing high-level qualifier 102
description 8
frequency of image copies 476
order of recovering
I/O errors 546
point in time recovery 494
recovery 494
SYSLGRNX table
discarding records 510
records log RBA ranges 480
table space names 8
DISABLE option
limits plan and package use 154
disaster recovery
data mirroring 576
preparation 483
scenario 558
using a tracker site 568
disaster, rolling 576
disconnecting
applications 386, 388
CICS from DB2, command 384

disconnecting *(continued)*
DB2 from TSO 383
discretionary access checking 193
disk
altering storage group assignment 67
data set, allocation and extension 701
improving utilization 701
requirements 111
DISPLAY command of IMS
SUBSYS option 388, 394
DISPLAY DATABASE command
displaying LPL entries 369
SPACENAM option 368
status checking 298
DISPLAY DDF command
displays connections to DDF 402
DISPLAY FUNCTION SPECIFIC command
displaying statistics about external user-defined functions 374
DISPLAY LOCATION command
controls connections to DDF 403
DISPLAY NET command of VTAM 412
DISPLAY OASN command of IMS
displaying RREs 393
produces OASN 524
DISPLAY privilege
description 136
DISPLAY PROCEDURE command
example 413
DISPLAY THREAD command
extensions to control DDF connections
DETAIL option 406
LOCATION option 405
LUWID option 409
messages issued 380
options
DETAIL 406
LOCATION 405
LUWID 409
TYPE (INDOUBT) 528
shows IMS threads 389, 394
shows parallel tasks 998
DISPLAY TRACE command
AUDIT option 286
DISPLAY UTILITY command
data set control log record 1111
DISPLAYDB privilege
description 136
displaying
buffer pool information 675
indoubt units of recovery 390, 528
information about
originating threads 382
parallel threads 382
postponed units of recovery 391
distinct type
privileges of ownership 146
DISTINCT TYPE privilege, description 138
distributed data
controlling connections 400
DB2 private protocol access
See DB2 private protocol
DRDA protocol
See DRDA access
operating
displaying status 1168
in an overloaded network 630
distributed data *(continued)*
performance considerations 1004
programming
block fetch 1005
FOR FETCH ONLY 1007
FOR READ ONLY 1007
resource limit facility 725
server-elapsed time monitoring 1017
tuning 1004
distributed data facility (DDF)
See DDF (distributed data facility)
Distributed Relational Database Architecture (DRDA) 18
distribution statistics 915
DL/I
batch
features 17
loading data 65
DL/I BATCH TIMEOUT field of installation panel DSNTIPI 834
DMTH (data management threshold) 668
double-hop situation 150
down-level detection
controlling 544
LEVEL UPDATE FREQ field of panel DSNTIPN 544
down-level page sets 543
DPropNR (DataPropagator NonRelational) 17
DPSI
performance considerations 786
drain
definition 865
DRAIN ALL 868
wait calculation 836
drain lock
description 809, 866
types 866
wait calculation 836
DRDA access
description 1003
resource limit facility 726
security mechanisms 238
DROP
statement
TABLE 89
TABLESPACE 69
DROP privilege
description 136
DROPIN privilege
description 136
dropping
columns from a table 88
database 68
privileges needed for package 164
table spaces 69
tables 89
views 96
volumes from a storage group 67
DSMAX
calculating 688
limit on open data sets 687
DSN command of TSO
command processor
connecting from TSO 381
description 18
invoked by TSO batch work 330
invoking 17
issues commands 328
running TSO programs 327

DSN command of TSO *(continued)*
subcommands
END 383
DSN command processor
See DSN command of TSO
DSN message prefix 331
DSN_PREDICAT_TABLE 1199
DSN_STATEMNT_TABLE table
column descriptions 982
DSN1CHKR utility
control of data set access 282
DSN1COMP utility
description 704
DSN1COPY utility
control of data set access 282
resetting log RBA 621
restoring data 498
DSN1LOGP utility
control of data set access 282
example 601
extract log records 1111
JCL
sample 598
limitations 618
print log records 1111
shows lost work 591
DSN1PRNT utility
description 282
DSN3@ATH connection exit routine.
See connection exit routine
DSN3@SGN sign-on exit routine.
See sign-on exit routine
DSN6SPRM macro
RELCURHL parameter 841
DSN6SYSP macro
PCLOSEN parameter 691
PCLOSET parameter 691
DSN8EAE1 exit routine 1077
DSN8HUFF edit routine 1077
DSNACCOR stored procedure
description 1257
example call 1270
option descriptions 1259
output 1274
syntax diagram 1259
DSNACICS stored procedure
debugging 1287
description 1280
invocation example 1285
invocation syntax 1281
output 1287
parameter descriptions 1281
restrictions 1287
DSNACICX user exit routine
description 1283
parameter list 1284
rules for writing 1283
DSNAEXP stored procedure
description 1295
example call 1297
option descriptions 1296
output 1297
syntax diagram 1295
DSNAIMS
option descriptions 1291
syntax diagram 1291
DSNAIMS stored procedure
description 1291
examples 1294
DSNALI (CAF language interface module)
inserting 1153
DSNC command of CICS
destination 323
prefix 333
DSNC DISCONNECT command of CICS
description 386
terminate DB2 threads 383
DSNC DISPLAY command of CICS
description 383
DSNC DISPLAY PLAN 386
DSNC DISPLAY TRANSACTION 386
DSNC STOP command of CICS
stop DB2 connection to CICS 383
DSNC STRT command of CICS
example 384
processing 386
start DB2 connection to CICS 383
DSNC transaction code
entering DB2 commands 321
DSNCLI (CICS language interface module)
entry point 1153
running CICS applications 329
DSNCRCT (resource control table)
See RCT (resource control table)
DSNDAIDL mapping macro 1054
DSNDB01 database
authority for access 144
DSNDB04 default database
See default database (DSNDB04)
DSNDB06 database
authority for access 144
changing high-level qualifier 102
DSNDB07 database.
See work file database
DSNDDTXP mapping macro 1085
DSNDEDIT mapping macro 1078
DSNDEXPL mapping macro 1106
DSNDFPPB mapping macro 1092
DSNDIFCA mapping macro 1175
DSNDQWIW mapping macro 1181
DSNDROW mapping macro 1108
DSNDRVAL mapping macro 1081
DSNDSLRB mapping macro 1126
DSNDSLRF mapping macro 1132
DSNDWBUF mapping macro 1155
DSNDWQAL mapping macro 1159
DSNDXAPL parameter list 1066
DSNELI (TSO language interface module) 327, 1153
DSNJSLR macro
capturing log records 1111
stand-alone CLOSE 1132
stand-alone sample program 1133
DSNLEUSR
Encrypting inbound IDs 249
Encrypting outbound IDs 262
Encrypting passwords 262
DSNLEUSR stored procedure
description 1287
example call 1289
option descriptions 1288
output 1290
syntax diagram 1288
DSNMxxx messages 332

DSNTEJ1S job 59
DSNTESP data set 920
DSNTIJEX job
exit routines 1052
DSNTIJIC job
improving recovery of inconsistent data 486
DSNTIJSG job
installation 717
DSNUM column
SYSINDEXPART catalog table
data collected by RUNSTATS utility 902
SYSINDEXPART_HIST catalog table 910
SYSTABLEPART catalog table
data collected by RUNSTATS utility 904
SYSTABLEPART_HIST catalog table 911
DSNX@XAC access control authorization exit routine 1061
DSNZPxxx
subsystem parameters module
specifying an alternate 325
dual logging
active log 426
archive logs 429
description 9
restoring 436
synchronization 427
dump
caution about using disk dump and restore 492
duration of locks
controlling 843
description 820
DWQT option of ALTER BUFFERPOOL command 669
dynamic plan selection exit routine 1103
dynamic plan selection in CICS
exit routine.
See plan selection exit routine
dynamic prefetch
description 970
dynamic SQL
authorization 164
caching
effect of RELEASE bind option 844
example 168
privileges required 164
skeletons, EDM pool 679
DYNAMICRULES
description 164
example 168

E

EA-enabled page sets 42
edit procedure, changing 87
edit routine
description 294, 1077
ensuring data accuracy 294
row formats 1106
specified by EDITPROC option 1077
writing 1077
EDITPROC clause
exit points 1078
specifies edit exit routine 1078
EDM pool
DBD freeing 733
description 679
EDPROC column of SYSTABLES catalog tabley 905
employee photo and resume sample table 1037
employee sample table 1034
employee-to-project-activity sample table 1040
ENABLE
option of BIND PLAN subcommand 154
enclave 739
ENCRYPT 209, 210
encrypting
data 1077
passwords from workstation 264
passwords on attachment requests 244, 262
encryption 207
built-in functions for 207
column level 208
data 207
defining columns for 208
non-character values 211
password hints 210, 211
performance recommendations 212
predicate evaluation 211
value level 210
with viewsl 209
END
subcommand of DSN
disconnecting from TSO 383
Enterprise Storage Server
backup 491
environment, operating
CICS 329
DB2 19
IMS 329
TSO 327
z/OS 19
ERRDEST option
DSNC MODIFY 384
unsolicited CICS messages 332
error
application program 520
IFI (instrumentation facility interface) 1184
physical RW 369
SQL query 297
escalation, lock 829
escape character
example 223
in DDL registration tables 219
EVALUATE UNCOMMITTED field of panel DSNTIP4 842
EXCLUSIVE
lock mode
effect on resources 822
LOB 862
page 821
row 821
table, partition, and table space 821
EXECUTE privilege
after BIND REPLACE 153
description 134, 137
effect 148
exit parameter list (EXPL) 1106
exit point
authorization routines 1053
connection routine 1053
conversion procedure 1087
date and time routines 1084
edit routine 1078
field procedure 1091
sign-on routine 1053
validation routine 1081
exit routine
authorization control 1061

exit routine *(continued)*
determining if active 1076
DSNACICX 1283
general considerations 1104
writing 1051
exit routine.
See also connection exit routine
See also conversion procedure
See also date routine
See also edit routine
See also field procedure
See also log capture exit routine
See also sign-on exit routine
See also time routine
See validation routine
EXPL (exit parameter list) 1106
EXPLAIN 1199
report of outer join 957
statement
alternative using IFI 1152
description 927
executing under DB2 QMF 937
index scans 941
interpreting output 939
investigating SQL processing 927
EXPLAIN PROCESSING field of panel DSNTIPO
overhead 937
EXPLAIN tables 1199, 1221, 1222
DSN_DETCOST_TABLE 1212
DSN_FILTER_TABLE 1210
DSN_PGROUP_TABLE 1205
DSN_PTASK_TABLE 1208
DSN_QUERY_TABLE 1224
DSN_SORT_TABLE 1218
DSN_SORTKEY_TABLE 1219
DSN_STRUCT_TABLE 1203
EXPORT command of access method services 107, 483
Extended Remote Copy (XRC) 581
EXTENDED SECURITY field of panel DSNTIPR 239
extending a data set, procedure 550
EXTENTS column
SYSINDEXPART catalog table
data collected by RUNSTATS utility 902
SYSINDEXPART_HIST catalog table 910
SYSTABLEPART catalog table
data collected by RUNSTATS utility 904
SYSTABLEPART_HIST catalog table 911
external storage
See auxiliary storage
EXTERNAL_SECURITY column of SYSIBM.SYSROUTINES catalog table, RACF access to non-DB2 resources 277

F

failure symptoms
abend shows
log problem during restart 607
restart failed 593, 602
BSDS 593
CICS
attachment abends 526
task abends 530
waits 525
IMS
loops 522
waits 522
log 593
failure symptoms *(continued)*
lost log information 613
message
DFH2206 525
DFS555 524
DSNB207I 542
DSNJ 610
DSNJ001I 538
DSNJ004I 533
DSNJ100 610
DSNJ103I 535
DSNJ105I 533
DSNJ106I 533
DSNJ107 610
DSNJ110E 532
DSNJ111E 532
DSNJ114I 536
DSNM002I 522
DSNM004I 522
DSNM005I 523
DSNM3201I 525
DSNP007I 548
DSNP012I 547
DSNU086I 545, 546
no processing is occurring 518
subsystem termination 530
z/OS error recovery program message 536
FARINDREF column
SYSTABLEPART_HIST catalog table 911
FARINDREF column of SYSTABLEPART catalog table
data collected by RUNSTATS utility 904
FAROFFPOSF column
SYSINDEXPART_HIST catalog table 910
FAROFFPOSF column of SYSINDEXPART catalog table
data collected by RUNSTATS utility 902
fast copy function
Enterprise Storage Server FlashCopy 491
RVA SnapShot 491
fast log apply
use during RECOVER processing 488
Fast Path Log Analysis Utility 1185
FETCH FIRST *n* ROWS ONLY clause
effect on OPTIMIZE clause 789
FETCH FIRST n ROW ONLY clause
effect on distributed performance 1011
effect on OPTIMIZE clause 1011
field decoding operation
definition 1089
input 1099
output 1099
field definition operation
definition 1089
input 1095
output 1095
field description of a value 1090
field encoding operation
definition 1089
input 1098
output 1098
field procedure
changing 87
description 294, 1089
ensuring data accuracy 294
specified by the FIELDPROC clause 1090
writing 1089
field procedure information block (FPIB) 1093
field procedure parameter list (FPPL) 1092

field procedure parameter value list (FPPVL) 1092
field value descriptor (FVD) 1092
field-level access control 143
FIELDPROC clause
  ALTER TABLE statement 1090
  CREATE TABLE statement 1090
filter factor
  catalog statistics used for determining 906
  predicate 760
FIRSTKEYCARD column
  SYSINDEXSTATS catalog table
    recommendation for updating 914
FIRSTKEYCARDF column
  SYSINDEXES catalog table
    data collected by RUNSTATS utility 902
    recommendation for updating 914
  SYSINDEXES_HIST catalog table 910
  SYSINDEXSTATS catalog table
    data collected by RUNSTATS utility 903
  SYSINDEXSTATS_HIST catalog table 910
fixed-length records, effect on processor resources 661
FORCE option
  START DATABASE command 365
  STOP DB2 command 396, 444
format
  column 1108
  data passed to FPPVL 1094
  data set names 38
  message 331
  recovery log record 1119
  row 1108
  value descriptors 1087, 1095
forward log recovery
  phase of restart 447
  scenario 602
FPIB (field procedure information block) 1093, 1094
FPPL (field procedure parameter list) 1092
FPPVL (field procedure parameter value list) 1092, 1094
FREE PACKAGE subcommand of DSN
  privileges needed 164
FREE PLAN subcommand of DSN
  privileges needed 164
free space
  description 652
  recommendations 653
FREEPAGE
  clause of ALTER INDEX statement
    effect on DB2 speed 652
  clause of ALTER TABLESPACE statement
    effect on DB2 speed 652
  clause of CREATE INDEX statement
    effect on DB2 speed 652
  clause of CREATE TABLESPACE statement
    effect on DB2 speed 652
FREESPACE column
  SYSLOBSTATS catalog table 903
  SYSLOBSTATS_HIST catalog table 910
FREQUENCYF column
  SYSCOLDIST catalog table
    access path selection 900
  SYSCOLDIST_HIST catalog table 909
  SYSCOLDISTSTATS catalog table 900
full image copy
  use after LOAD 699
  use after REORG 699
FULLKEYCARDDATA column
  SYSINDEXSTATS catalog table 903
FULLKEYCARDF column
  SYSINDEXES catalog table
    data collected by RUNSTATS utility 902
  SYSINDEXES_HIST catalog table 910
  SYSINDEXSTATS catalog table 903
  SYSINDEXSTATS_HIST catalog table 910
function
  column
    when evaluated 946
FUNCTION privilege, description 136
function, user-defined 155
FVD (field value descriptor) 1092, 1094

G

generalized trace facility (GTF).
  See GTF (generalized trace facility)
GETHINT 210
global transaction
  definition of 461
glossary 1305
governor (resource limit facility)
  See resource limit facility (governor)
GRANT statement
  examples 174, 180
  format 174
  privileges required 164
granting privileges and authorities 174
GROUP BY clause
  effect on OPTIMIZE clause 791
GROUP DD statement for stand-alone log services OPEN request 1127
GTF (generalized trace facility)
  event identifiers 1195
  format of trace records 1135
  interpreting trace records 1140
  recording trace records 1195

H

help
  DB2 UTILITIES panel 11
heuristic damage 464
heuristic decision 464
Hierarchical Storage Manager (DFSMShsm)
  See DFSMShsm (Hierarchical Storage Manager)
HIGH2KEY column
  SYSCOLSTATS catalog table 901
  SYSCOLUMNS catalog table
    access path selection 901
    recommendation for updating 915
  SYSCOLUMNS_HIST catalog table 909
HIGHKEY column of SYSCOLSTATS catalog table 901
hints, optimization 801
HMIGRATE command of DFSMShsm (Hierarchical Storage Manager) 107
hop situation 151
host variable
  example query 774
  impact on access path selection 774
  in equal predicate 777
  tuning queries 774
HRECALL command of DFSMShsm (Hierarchical Storage Manager) 107
Huffman compression.
  See also data compression, Huffman

Huffman compression. *(continued)*
exit routine 1077
hybrid join
description 961
disabling 683

I

I/O activity, monitoring by data set 693
I/O error
catalog 546
directory 546
occurrence 436
table spaces 545
I/O processing
minimizing contention 657, 710
parallel
disabling 669
queries 989
identity column
altering attributes 88
loading data into 62
identity columns
conditional restart 452
IEFSSNxx member of SYS1.PARMLIB
IRLM 376
IFCA (instrumentation facility communication area)
command request 1155
description 1174
field descriptions 1175
IFI READS request 1157
READA request of IFI 1171
WRITE request of IFI 1173
IFCID (instrumentation facility component identifier)
0199 677, 693
0330 427, 532
area
description 1179
READS request of IFI 1158
WRITE request of IFI 1173
description 1136
identifiers by number
0001 1013, 1167, 1190
0002 1167
0015 730
0021 732
0032 732
0033 732
0038 732
0039 732
0058 731
0070 732
0073 730
0084 732
0088 733
0089 733
0106 1168
0124 1168
0147 1168, 1192
0148 1168
0149 1168
0150 1168
0185 1168
0199 1168
0202 1168
0217 1168
0221 999
IFCID (instrumentation facility component identifier) *(continued)*
identifiers by number *(continued)*
0222 999
0225 1168
0230 1168
0234 1168
0254 1168
0258 702
0306 1123, 1168
0314 1076
0316 1169
0317 1169
SMF type 1190, 1192
IFI (instrumentation facility interface)
asynchronous data 1172
auditing data 1183
authorization 1157
buffer information area 1155
collecting trace data, example 1152
command request, output example 1182
commands
READA 1171
READS 1156, 1158
data integrity 1183
data sharing group, in a 1179
decompressing log data 1123
dynamic statement cache information 1169
errors 1184
issuing DB2 commands
example 1156
syntax 1154
locking 1183
output area
command request 1155
description 1179
example 1156
passing data to DB2, example 1152
qualification area 1159
READS output 1181
READS request 1158
recovery considerations 1184
return area
command request 1155
description 1178
READA request 1171
READS request 1157
storage requirements 1158, 1171
summary of functions 1152
synchronous data 1167
using stored procedures 1154
WRITE 1173
writer header 1181
ii1 1887 modify bullets ** 10/16/02 629
IMAGCOPY privilege
description 136
image copy
catalog 476
directory 476
frequency vs. recovery speed 476
full
use after LOAD 699
incremental
frequency 476
making after loading a table 63
recovery speed 476
immediate write threshold (IWTH) 668

implementor, description 155
IMPORT command of access method services 107, 611
IMS
  commands
    CHANGE SUBSYS 388, 393
    DISPLAY OASN 393
    DISPLAY SUBSYS 388, 394
    response destination 323
    START REGION 396
    START SUBSYS 388
    STOP REGION 396
    STOP SUBSYS 388, 396
    TRACE SUBSYS 388
    used in DB2 environment 317
  connecting to DB2
    attachment facility 393
    authorization IDs 329
    connection ID 329
    connection processing 233
    controlling 16, 388
    dependent region connections 393
    disconnecting applications 396
    security 279
    sign-on processing 236
    supplying secondary IDs 233
  facilities
    Fast Path 743
    message format 332
    processing limit 716
    regions 742
    tools 1187
  indoubt units of recovery 460
  language interface module (DFSLI000)
    IFI applications 1153
    link-editing 329
  LTERM authorization ID
    for message-driven regions 329
    shown by /DISPLAY SUBSYS 394
    used with GRANT 324
  operating
    batch work 329
    entering DB2 commands 320
    recovery from system failure 16
    running programs 328
    tracing 420
  planning
    design recommendations 742
    environment 329
  programming
    application 16
    error checking 329
  recovery
    resolution of indoubt units of recovery 460
  recovery scenarios 521, 522
  system administration 17
  thread 389, 390
  two-phase commit 455
  using with DB2 16
IMS BMP TIMEOUT field of panel DSNTIPI 834
IMS Performance Analyzer (IMS PA)
  description 1185
  IMS transit times 642
IMS transactions stored procedure
  multiple connections 1294
  option descriptions 1291
  syntax diagram 1291
IMS.PROCLIB library
  connecting from dependent regions 393
inactive connections 736
index
  access methods
    access path selection 950
    by nonmatching index 951
    description 948
    IN-list index scan 951
    matching index columns 941
    matching index description 950
    multiple 952
    one-fetch index scan 953
  altering
    ALTER INDEX statement 92
    effects of dropping 94
  backward index scan 58
  copying 489
  costs 948
  description 6
  evaluating effectiveness 705
  implementing 54
  locking 824
  NOT PADDED
    advantages of using 57
    disadvantages of using 57
    index-only access 57
    varying-length column 57
  ordered data
    backward scan 58
    forward scan 58
  ownership 146
  privileges of ownership 146
  reasons for using 948
  reorganizing 95
  space
    description 6
    estimating size 118, 119
    recovery scenario 545
    storage allocated 41
  structure
    index tree 117
    leaf pages 117
    overall 118
    root page 117
    subpages 117
  types 54
    clustering 55
    data-partitioned secondary 56
    nonpartitioned secondary 56
    partitioning 55
    secondary 56
    unique 54
  versions 94
    recycling version numbers 95
INDEX privilege
  description 134
index structure
  root page
    leaf pages 117
index-only access
  NOT PADDED attribute 57
INDEXSPACESTATS
  contents 1237
  real-time statistics table 1230
indoubt thread
  displaying information about 464

indoubt thread *(continued)*
recovering 465
resetting status 465
resolving 582
information center consultant 171
INITIAL_INSTS column of SYSROUTINES catalog table 904
INITIAL_IOS column of SYSROUTINES catalog table 904
INLISTP 801
INSERT privilege
description 134
INSERT processing, effect of MEMBER CLUSTER option of CREATE TABLESPACE 813
INSERT statement
example 64
load data 61, 63
installation
macros
automatic IRLM start 377
installation SYSADM authority
privileges 142
use of RACF profiles 283
installation SYSOPR authority
privilege 140
use of RACF profiles 283
instrumentation facility communication area (IFCA)
See IFCA (instrumentation facility communication area)
instrumentation facility interface (IFI)
See IFI (instrumentation facility interface)
INSTS_PER_INVOC column of SYSROUTINES catalog table 904
integrated catalog facility
changing alias name for DB2 data sets 98
controlling storage 29
integrity
IFI data 1183
reports 298
INTENT EXCLUSIVE lock mode 822, 862
INTENT SHARE lock mode 822, 862
Interactive System Productivity Facility (ISPF)
See ISPF (Interactive System Productivity Facility)
internal resource lock manager (IRLM)
See IRLM (internal resource lock manager)
invalid LOB, recovering 544
invoker, description 155
invoking
DSN command processor 17
IOS_PER_INVOC column of SYSROUTINES catalog table 904
IRLM
administering 14
description 14
IRLM (internal resource lock manager)
controlling 376
diagnostic trace 420
element name
global mode 379
local mode 379
failure 517
IFI trace records 1167
monitoring connection 377, 378
MVS dispatching priority 813
OMEGAMON locking report 874
recovery scenario 517
starting
automatically 325, 377
manually 377
startup procedure options 832
stopping 378
IRLM (internal resource lock manager) *(continued)*
trace options, effect on performance 661
workload manager 712
ISOLATION
option of BIND PLAN subcommand
effects on locks 846
isolation level
control by SQL statement
example 857
recommendations 815
ISPF (Interactive System Productivity Facility)
DB2 considerations 18
requirement 17
system administration 18
tutorial panels 11
IWTH (immediate write threshold) 668

J

JAR 146
privileges of ownership 146
Java class for a routine 146
privileges of ownership 146
Java class privilege
description 138
JCL jobs
scheduled execution 361
join operation
Cartesian 958
description 955
hybrid
description 961
disabling 683
join sequence 963
merge scan 959
nested loop 957
star join 963
star schema 963
join sequence
definition 752

K

key
adding 78
dropping 79
foreign 78, 79
parent 78, 79
unique 78, 79
KEYCARDDATA column
SYSCOLDIST catalog table
data collected by RUNSTATS utility 900
KEYCOUNTF column
SYSINDEXSTATS catalog table 903
SYSINDEXSTATS_HIST catalog table 910

L

language interface modules
DFSLI000 1153
DSNALI 1153
DSNCLI
description 1153
usage 329
DSNELI 1153
latch 809

LCID (log control interval definition) 1118
LE tokens 707
leaf page
  description 117
  index 117
LEAFDIST column
  SYSINDEXPART catalog table
    data collected by RUNSTATS utility 902
  SYSINDEXPART_HIST catalog table 910
LEAFFAR column
  SYSINDEXPART catalog table 902
    example 921
  SYSINDEXPART_HIST catalog table 910
LEAFNEAR column
  SYSINDEXPART catalog table 903
  SYSINDEXPART_HIST catalog table 910
level of a lock 817
LEVEL UPDATE FREQ field of panel DSNTIPN 544
LIMIT BACKOUT field of panel DSNTIPN 450
limited block fetch
  See block fetch
limited partition scan 944
LIMITKEY column
  SYSINDEXPART catalog table 903
list prefetch
  description 970
  disabling 683
  thresholds 971
LOAD privilege
  description 135
LOAD utility
  availability of tables when using 62
  CCSID option 62
  delimited files 62
  effect on real-time statistics 1244
  loading DB2 tables 61
  making corrections 63
  moving data 106
loading
  data
    DL/I 65
    sequential data sets 61
    SQL INSERT statement 63
  tables 61
LOB
  lock
    concurrency with UR readers 853
    description 860
LOB (large object)
  block fetching 1007
  lock duration 862
  LOCK TABLE statement 864
  locking 860
  LOCKSIZE clause of CREATE or ALTER
    TABLESPACE 864
  modes of LOB locks 862
  modes of table space locks 862
  recommendations for buffer pool DWQT threshold 671
  recovering invalid 544
  when to reorganize 923
local attachment request 242
LOCAL DATE LENGTH
  field of panel DSNTIPF 1084
LOCAL TIME LENGTH
  field of panel DSNTIPF 1084
lock
  avoidance 842, 855
lock *(continued)*
  benefits 810
  class
    drain 809
    transaction 809
  compatibility 823
  DB2 installation options 833
  description 809
  drain
    description 866
    types 866
    wait calculation 836
  duration
    controlling 843
    description 820
    LOBs 862
    page locks 732
  effect of cursor WITH HOLD 857
  effects
    deadlock 811
    deadlock wait calculation 835
    suspension 810
    timeout 811
    timeout periods 834
  escalation
    description 829
    OMEGAMON reports 871
  hierarchy
    description 817
  LOB locks 860
  LOB table space, LOCKSIZE clause 864
  maximum number 837
  mode 821
  modes for various processes 831
  object
    DB2 catalog 824
    DBD 825
    description 823
    indexes 824
    LOCKMAX clause 839
    LOCKSIZE clause 838
    SKCT (skeleton cursor table) 825
    SKPT (skeleton package table) 825
  options affecting
    bind 843
    cursor stability 848
    IFI (instrumentation facility interface) 1183
    IRLM 832
    program 843
    read stability 851
    repeatable read 852
    uncommitted read 850
  page locks
    commit duration 732
    CS, RS, and RR compared 852
    description 817
    performance 871
  promotion 829
  recommendations for concurrency 812
  row locks
    compared to page 839
  size
    controlling 838, 839
    page 817
    partition 817
    table 817
    table space 817

lock *(continued)*
storage needed 833
suspension time 874
table of modes acquired 826
trace records 731
LOCK TABLE statement
effect on auxiliary tables 864
effect on locks 859
lock/latch suspension time 643
LOCKMAX clause
effect of options 839
LOCKPART clause of CREATE and ALTER TABLESPACE
effect on locking 818
LOCKS PER TABLE(SPACE) field of panel DSNTIPJ 840
LOCKS PER USER field of panel DSNTIPJ 837
LOCKSIZE clause
effect of options 838, 864
recommendations 813
log
buffer
creating log records 425
retrieving log records 425
size 694
capture exit routine 1111, 1134
changing BSDS inventory 438
checkpoint records 1115
contents 1111
deciding how long to keep 438
determining size of active logs 698
dual
active copy 426
archive logs 437
synchronization 427
to minimize restart effort 610
effects of data compression 1112
excessive loss 613
failure
recovery scenario 531, 535
symptoms 593
total loss 613
hierarchy 425
implementing logging 430
initialization phase
failure scenario 593
process 445
operation 298
performance
considerations 694
recommendations 695
reading without running RECOVER utility 489
record structure
control interval definition (LCID) 1118
database page set control records 1116
format 1119
header (LRH) 1111, 1117
logical 1116
physical 1116
type codes 1120
truncation 601
use
backward recovery 448
establishing 425
exit routine 1101
forward recovery 447
managing 423, 476
monitoring 695
record retrieval 425
log *(continued)*
use *(continued)*
recovery scenario 610
restarting 444, 449
write threshold 694, 695
log capture exit routine
contents of log 1111
reading log records 1134
writing 1101
log capture exit routine.
See also DATA CAPTURE clause
description 1101
log range directory 8
log record header (LRH) 1117
log record sequence number (LRSN) 1111
log write, forced at commit 695
logical page list (LPL)
See LPL (logical page list)
LOW2KEY column
SYSCOLSTATS catalog table 901
SYSCOLUMNS catalog table
access path selection 901
recommendation for updating 915
SYSCOLUMNS_HIST catalog table 909
LOWKEY column of SYSCOLSTATS catalog table 901
LPL
option of DISPLAY DATABASE command 369
status in DISPLAY DATABASE output 369
LPL (logical page list)
deferred restart 450
description 369
recovering pages
methods 370
running utilities on objects 370
LRH (log record header) 1117
LRSN statement of stand-alone log services OPEN request 1129

M

mandatory access checking
definition 193
dominance 193
mapping macro
DSNDAIDL 1054
DSNDDTXP 1085
DSNDEDIT 1078
DSNDEXPL 1106
DSNDFPPB 1092
DSNDIFCA 1175
DSNDQWIW 1181
DSNDROW 1108
DSNDRVAL 1081
DSNDSLRB 1126
DSNDSLRF 1132
DSNDWBUF 1155
DSNDWQAL 1159
mass delete
contends with UR process 853
validation routine 1081
materialization
outer join 957
views and nested table expressions 976
materialized query table 884
altering 886
changing 85
changing attributes 86

materialized query table *(continued)*
changing definition of 86
changing to a base table 86
creating 883
defining 883
definition 881
design guidelines 897
enabling for automatic query rewrite 889
examples in automatic query rewrite 892, 898
introduction 881
maintaining 886
populating 886
refresh age 890
refreshing 886
registering 85
registering an existing table as 85
statistics 888
system-maintained 884
use in automatic query rewrite 891
user-maintained 884
MAX BATCH CONNECT field of panel DSNTIPE 743
MAX REMOTE ACTIVE field of panel DSNTIPE 735
MAX REMOTE CONNECTED field of panel DSNTIPE 735
MAX TSO CONNECT field of panel DSNTIPE 743
MAXCSA option of START irlmproc command 832
MEMBER CLUSTER option of CREATE TABLESPACE 813
merge processing
views or nested table expressions 975
message
DSNJ106I 605
format
DB2 331
IMS 332
prefix for DB2 331
receiving subsystem 331
z/OS abend
IEC030I 536, 537
IEC031I 536
IEC032I 536
message by identifier
$HASP373 325
DFS058 388
DFS058I 396
DFS3602I 524
DFS3613I 389
DFS554I 525
DFS555A 524
DFS555I 525
DSN1150I 608
DSN1157I 601, 608
DSN1160I 601, 609
DSN1162I 601, 608
DSN1213I 615
DSN2001I 527
DSN2025I 530
DSN2034I 527
DSN2035I 527
DSN2036I 527
DSN3100I 324, 326, 530
DSN3104I 326, 530
DSN3201I 526
DSN9032I 401
DSNB204I 542
DSNB207I 542
DSNB232I 543
DSNBB440I 998
DSNC012I 387
message by identifier *(continued)*
DSNC016I 461
DSNC025I 387
DSNI006I 370
DSNI021I 370
DSNI103I 830
DSNJ001I 325, 427, 446, 592, 593
DSNJ002I 427
DSNJ003I 427, 539
DSNJ004I 427, 533
DSNJ005I 427
DSNJ007I 595, 598, 606
DSNJ008E 427
DSNJ012I 595, 596, 604
DSNJ072E 430
DSNJ099I 325
DSNJ100I 538, 592, 610
DSNJ103I 535, 595, 597, 604, 606
DSNJ104I 535, 595
DSNJ105I 533
DSNJ106I 533, 595, 596, 604
DSNJ107I 538, 592, 610
DSNJ108I 538
DSNJ110E 427, 532
DSNJ111E 427, 532
DSNJ113E 595, 597, 604, 605, 610
DSNJ114I 536
DSNJ115I 535
DSNJ1191 592
DSNJ119I 610
DSNJ120I 445, 539
DSNJ123E 538
DSNJ124I 534
DSNJ125I 437, 538
DSNJ126I 538
DSNJ127I 325
DSNJ128I 536
DSNJ130I 445
DSNJ139I 427
DSNJ301I 538
DSNJ302I 538
DSNJ303I 538
DSNJ304I 538
DSNJ305I 538
DSNJ306I 538
DSNJ307I 538
DSNJ311E 432
DSNJ312I 432
DSNJ317I 432
DSNJ318I 432
DSNJ319I 432
DSNL001I 401
DSNL002I 418
DSNL003I 401
DSNL004I 401
DSNL005I 418
DSNL006I 418
DSNL009I 411
DSNL010I 411
DSNL030I 557
DSNL080I 402, 403
DSNL200I 404
DSNL432I 418
DSNL433I 418
DSNL500I 556
DSNL501I 553, 556
DSNL502I 553, 556

message by identifier *(continued)*
DSNL700I 554
DSNL701I 555
DSNL702I 555
DSNL703I 555
DSNL704I 555
DSNL705I 555
DSNM001I 389, 396
DSNM002I 396, 522, 530
DSNM003I 389, 396
DSNM004I 460, 522
DSNM005I 392, 460, 523
DSNP001I 548, 549
DSNP007I 548
DSNP012I 547
DSNR001I 325
DSNR002I 325, 592
DSNR003I 325, 440, 606, 607, 608
DSNR004I 325, 446, 448, 592, 593, 602
DSNR005I 325, 448, 593, 607
DSNR006I 325, 449, 593
DSNR007I 325, 446, 448
DSNR031I 448
DSNT360I 365, 368, 371
DSNT361I 365, 368, 371
DSNT362I 365, 368, 371
DSNT392I 371, 1113
DSNT397I 368, 371
DSNU086I 545, 546
DSNU234I 704
DSNU244I 704
DSNU561I 552
DSNU563I 552
DSNV086E 530
DSNV400I 432
DSNV401I 385, 390, 391, 432, 528
DSNV402I 321, 380, 394, 406, 432
DSNV404I 382, 394
DSNV406I 380, 385, 390, 391, 528
DSNV407I 380
DSNV408I 385, 391, 398, 453, 528
DSNV414I 385, 391, 398, 529
DSNV415I 385, 391, 399, 529
DSNV431I 385
DSNV435I 453
DSNX940I 413
DSNY001I 325
DSNY002I 326
DSNZ002I 325
DXR105E 378
DXR117I 377
DXR1211 379
DXR122E 517
DXR1651 379
EDC3009I 547
IEC161I 542
message processing program (MPP)
See MPP (message processing program)
MIGRATE command of DFSMShsm (Hierarchical Storage Manager) 107
misc ruddy/dh122 modify ** tonello 1/28/02 703
mixed data
altering subtype 88
mode of a lock 821
MODIFY irlmproc,ABEND command of z/OS
stopping IRLM 378
MODIFY utility
retaining image copies 487
monitor program
using IFI 1151
using OMEGAMON 1196
MONITOR1 privilege
description 136
MONITOR2 privilege
description 136
monitoring
application packages 1197
application plans 1197
CAF connections 381
CICS 1187
connections activity 394
databases 365
DB2 1187
DSNC commands for 386
IMS 1187
IRLM 377, 378
server-elapsed time for remote requests 1017
threads 386
tools
DB2 trace 1189
monitor trace 1192
performance 1185
TSO connections 381
user-defined functions 374
using IFI 1151
moving DB2 data 105
MPP (message processing program), connection control 394
multi-character command prefix
See command prefix, multi-character
multi-site update
illustration 468
process 467
multilevel security
advantages 192
constraints 206
DB2 resource classes 196
definition 192
discretionary access checking 193
edit procedures 206
field procedures 206
global temporary tables 205
hierarchies for objects 196
implementing 196
in a distributed environment 206
mandatory access checking 193
row-level granularity 197
security category 193
security label 193
security level 193
SNA support 207
SQL statements 198
TCP/IP support 206
triggers 206
using utilities with 204
validation procedures 206
views 205
multiple allegiance 710
multivolume archive log data sets 430
MxxACT DD statement for stand-alone log services OPEN request 1128
MxxARCHV DD statement for stand-alone log services OPEN request 1127

MxxBSDS DD statement for stand-alone log services OPEN request 1127

N

NACTIVE column
SYSTABSTATS catalog table 906
NACTIVEF column of SYSTABLESPACE catalog table
data collected by RUNSTATS utility 906
naming convention
implicitly created table spaces 45
VSAM data sets 38
NEARINDREF column
SYSTABLEPART catalog table 904
SYSTABLEPART_HIST catalog table 911
NEAROFFPOSF column
SYSINDEXPART catalog table
data collected by RUNSTATS utility 903
SYSINDEXPART_HIST catalog table 910
nested table expression
processing 975
NetView
monitoring errors in the network 416
network ID (NID)
See NID (network ID)
NID (network ID)
indoubt threads 523
thread identification 392
unique value assigned by IMS 392
use with CICS 528
NLEAF column
SYSINDEXES catalog table
data collected by RUNSTATS utility 902
SYSINDEXES_HIST catalog table 910
SYSINDEXSTATS catalog table 903
SYSINDEXSTATS_HIST catalog table 910
NLEVELS column
SYSINDEXES catalog table
data collected by RUNSTATS utility 902
SYSINDEXES_HIST catalog table 910
SYSINDEXSTATS catalog table 903
SYSINDEXSTATS_HIST catalog table 910
non-DB2 utilities
effect on real-time statistics 1251
noncorrelated subqueries 781
nonsegmented table space
dropping 699
locking 819
scan 947
normal read 666
NOT NULL clause
CREATE TABLE statement
requires presence of data 293
notices, legal 1301
NPAGES column
SYSTABLES catalog table 905
SYSTABSTATS catalog table 906
SYSTABSTATS_HIST catalog table 911
NPAGESF column
SYSTABLES catalog table
data collected by RUNSTATS utilit 905
SYSTABLES_HIST catalog table 911
null value
effect on storage space 1107
NUMBER OF LOGS field of panel DSNTIPL 698
NUMCOLUMNS column
SYSCOLDIST catalog table
access path selection 900
SYSCOLDIST_HIST catalog tableST_HIST catalog table 909
SYSCOLDISTSTATS catalog table 900
numeric
data
format in storage 1110

O

OASN (originating sequence number)
indoubt threads 523
part of the NID 392
object
controlling access to 133, 191
ownership 145, 148
object of a lock 823
object registration table (ORT)
See registration tables for DDL
objects
recovering dropped objects 505
offloading
active log 426
description 425
messages 427
trigger events 426
OMEGAMON
accounting report
concurrency scenario 873
overview 640
description 1185, 1196
scenario using reports 872
statistics report
buffer pools 677
DB2 log 695
EDM pool 681
thread queuing 743
online monitor program using IFI 1151
OPEN
statement
performance 974
operation
continuous 12
description 363
log 298
operator
CICS 16
commands 317
not required for IMS start 16
START command 18
optimistic concurrency control 848
optimization hints 801
Optimization tools
tables used by 1199, 1227
OPTIMIZE FOR *n* ROWS clause 790
interaction with FETCH FIRST clause 789
OPTIMIZE FOR n ROWS clause
effect on distributed performance 1010, 1011
interaction with FETCH FIRST clause 1011
ORDER BY clause
effect on OPTIMIZE clause 791
ORGRATIO column
SYSLOBSTATS catalog table 904
SYSLOBSTATS_HIST catalog table 911

originating sequence number (OASN)
See OASN (originating sequence number)
originating task 990
ORT (object registration table)
See registration tables for DDL
OS/390 environment 13
outer join
EXPLAIN report 957
materialization 957
output area used in IFI
command request 1155
description 1179
example 1156
WRITE request 1173
output, unsolicited
CICS 332
operational control 332
subsystem messages 332
overflow 1114
OWNER
qualifies names in plan or package 145
ownership
changing 148
ownership of objects
establishing 145, 146
privileges 146

P

PACKADM authority
description 140
package
accounting trace 1191
administrator 171, 176
authorization to execute SQL in 149
binding
EXPLAIN option for remote 937
PLAN_TABLE 929
controlling use of DDL 215, 227
inoperative, when privilege is revoked 186
invalidated
dropping a view 96
dropping an index 94
when privilege is revoked 186
when table is dropped 89
list
privilege needed to include package 164
privileges needed to bind 154
monitoring 1197
privileges
description 130
explicit 137
for copying 154
of ownership 146
remote bind 154
retrieving catalog information 190
RLFPKG column of RLST 721
routine 155
SKPT (skeleton package table) 679
page
16-KB 113
32-KB 113
8-KB 113
buffer pool 666
locks
description 817
in OMEGAMON reports 871
page *(continued)*
number of records
description 113
root 117
size of index 117
table space 44
page set
control records 1116
copying 489
page size
choosing 45
choosing for LOBs 46
PAGE_RANGE column of PLAN_TABLE 944
PAGESAVE column
SYSTABLEPART catalog table
data collected by RUNSTATS utility 904
updated by LOAD and REORG utilities for data compression 705
SYSTABLEPART_HIST catalog table 911
Parallel Access Volumes (PAV) 710
parallel processing
description 987
disabling using resource limit facility 727
enabling 993
monitoring 997
related PLAN_TABLE columns 945
tuning 1000
parallelism
See also parallel processing
modes 727
PARM option of START DB2 command 325
partial recovery.
See point-in-time recovery
participant
in multi-site update 467
in two-phase commit 455
partition
adding 80, 84
altering 82
boundary
changing 82
changing to previous 84
extending 84
compressing data 702
index-controlled, redefining, procedure 552
rotating 83
table-controlled, redefining, procedure 552
partition scan, limited 944
partitioned data set, managing 20
partitioned table space
locking 818
partner LU
trusting 244
verifying by VTAM 242
PassTicket
configuring to send 263
password
changing expired ones when using DRDA 239
encrypting, for inbound IDs 244
encrypting, from workstation 264
RACF, encrypted 263
requiring, for inbound IDs 244
sending, with attachment request 262
pattern character
examples 223
in DDL registration tables 219
PAV (Parallel Access Volumes) 710

PC option of START irlmproc command 832
PCLOSEN subsystem parameter 691
PCLOSET subsystem parameter 691
PCTFREE
  effect on DB2 performance 652
PCTPAGES column
  SYSTABLES catalog table 905
  SYSTABLES_HIST catalog table 911
  SYSTABSTATS catalog table 906
PCTROWCOMP column
  SYSTABLES catalog table 705
    data collected by RUNSTATS utility 905
  SYSTABLES_HIST catalog table 911
  SYSTABSTATS catalog table 705, 906
  updated by LOAD and REORG for data compression 705
PERCACTIVE column
  SYSTABLEPART catalog table
    data collected by RUNSTATS utility 904
  SYSTABLEPART_HIST catalog table 911
PERCDROP column
  SYSTABLEPART catalog table
    data collected by RUNSTATS utility 905
  SYSTABLEPART_HIST catalog table 911
performance
  affected by
    cache for authorization IDs 152
    CLOSE NO 651
    data set distribution 655
    EDM and buffer pools 651
    extents 659
    I/O activity 651
    lock size 820
    PCTFREE 652
    PRIQTY clause 659
    secondary authorization IDs 161
    storage group 29
  monitoring
    planning 633
    RUNSTATS 651
    tools 1185
    trace 1192
    using OMEGAMON 1196
    with EXPLAIN 927
performance considerations
  DPSI 786
  scrollable cursor 785
Performance Reporter for OS/390 1197
phases of execution
  restart 445
PIECESIZE clause
  ALTER INDEX statement
    recommendations 655
    relation to PRIQTY 659
  CREATE INDEX statement
    recommendations 655
    relation to PRIQTY 659
PLAN
  option of DSNC DISPLAY command 386
PLAN_TABLE table
  column descriptions 929
  report of outer join 957
plan, application
  See application plan
planning
  auditing 127
  security 127
POE
  See port of entry
point in time recovery
  catalog and directory 494
  description 500
point of consistency
  CICS 455
  description 423
  IMS 455
  recovering data 495
  single system 455
pointer, overflow 1114
pool, inactive connections 736
populating
  tables 61
port of entry 246, 252
  RACF APPCPORT class 273
  RACF SERVAUTH class 273
postponed abort unit of recovery 458
power failure recovery scenario, z/OS 518
PQTY column
  SYSINDEXPART catalog table
    data collected by RUNSTATS utility 903
  SYSINDEXPART_HIST catalog table 910
  SYSTABLEPART catalog table
    data collected by RUNSTATS utility 905
  SYSTABLEPART_HIST catalog table 911
predicate
  description 749
  evaluation rules 753
  filter factor 760
  generation 770
  impact on access paths 749
  indexable 751
  join 750
  local 750
  modification 770
  properties 749
  stage 1 (sargable) 751
  stage 2
    evaluated 751
    influencing creation 796
  subquery 750
PREFORMAT
  option of LOAD utility 658
  option of REORG TABLESPACE utility 658
preformatting space for data sets 658
primary authorization ID
  See authorization ID, primary
PRIMARY_ACCESSTYPE column of PLAN_TABLE 942
PRINT
  command of access method services 504
print log map utility
  before fall back 611
  control of data set access 282
  prints contents of BSDS 376, 441
prioritizing resources 715
privilege
  description 133
  executing an application plan 130
  exercised by type of ID 161
  exercised through a plan or package 148, 154
  explicitly granted 133, 143
  granting 130, 173, 181, 187
  implicitly held 145, 148
  needed for various roles 171
  ownership 146

privilege *(continued)*
remote bind 154
remote users 174
retrieving catalog information 187, 191
revoking 181
routine plans, packages 155
types 134, 138
used in different jobs 171
privilege selection, sample security plan 302
problem determination
using OMEGAMON 1196
PROCEDURE privilege 136
process
description 128
processing
attachment requests 245, 258
connection requests 233, 235
sign-on requests 236, 238
processing speed
processor resources consumed
accounting trace 644, 1193
buffer pool 673
fixed-length records 661
thread creation 733
thread reuse 660
traces 660
transaction manager 1189
RMF reports 1188
time needed to perform I/O operations 654
PROCLIM option of IMS TRANSACTION macro 743
production binder
description 171
privileges 178
Profile tables 1227
DSN_VIRTUAL_INDEXES 1227
project activity sample table 1039
project sample table 1038
protocols
SNA 242
TCP/IP 249
PSB name, IMS 329
PSEUDO_DELETED_ENTRIES column
SYSINDEXPART catalog table 903
SYSINDEXPART_HIST catalog table 910
PSRCP (page set recovery pending) status
description 63
PSTOP transaction type 394
PUBLIC AT ALL LOCATIONS clause
GRANT statement 174
PUBLIC clause
GRANT statement 173
PUBLIC identifier 173
PUBLIC* identifier 174

Q

QMF (Query Management Facility)
database for each user 44
options 744
performance 744
QSAM (queued sequential access method) 428
qualification area used in IFI
description 1124
description of fields 1159
READS request 1158
restricted IFCIDs 1158
restrictions 1166
qualified objects
ownership 146
QUALIFIER
qualifies names in plan or package 145
Query Management Facility (QMF)
See QMF (Query Management Facility)
query parallelism 987
QUERYNO clause
reasons to use 805
queued sequential access method (QSAM)
See QSAM (queued sequential access method)
QUIESCE option
STOP DB2 command 396, 444

R

RACF (Resource Access Control Facility)
authorizing
access to data sets 132, 281, 283
access to protected resources 267
access to server resource class 275
group access 271
IMS access profile 271
SYSADM and SYSOPR authorities 271
checking
connection processing 233, 235
inbound remote IDs 243
sign-on processing 236, 238
defining
access profiles 266
DB2 resources 265, 278
protection for DB2 264, 278
remote user IDs 271
started procedure table 268
user ID for DB2 started tasks 267
description 131
PassTickets 263
passwords, encrypted 263
typical external security system 231
when supplying secondary authorization ID 235, 237
RACF access control module 1110
RBA (relative byte address)
description 1111
range shown in messages 427
RCT (resource control table)
DCT entry 384
ERRDEST option 332, 384
re-creating
tables 90
read asynchronously (READA) 1171
read synchronously (READS) 1156
READA (read asynchronously) 1171
reading
normal read 666
sequential prefetch 666
READS (read synchronously) 1156, 1158
real storage 708
REAL TIME STATS
field of panel DSNTIPO 1231
real-time statistics
accuracy 1254
for DEFINE NO objects 1251
for read-only objects 1251
for TEMP table spaces 1251
for work file table spaces 1251
improving concurrency 1253
in data sharing 1253

real-time statistics *(continued)*
when DB2 externalizes 1243
real-time statistics tables
altering 1230
contents 1231
creating 1230
description 1229
effect of dropping objects 1251
effect of mass delete operations 1253
effect of SQL operations 1252
effect of updating partitioning keys 1252
establishing base values 1231
INDEXSPACESTATS 1230
recovering 1253
setting up 1229
setting update interval 1231
starting 1231
TABLESPACESTATS 1230
reason code
X'00C90088' 812
X'00C9008E' 811
REBIND PACKAGE subcommand of DSN
options
ISOLATION 846
OWNER 148
RELEASE 843
REBIND PLAN subcommand of DSN
options
ACQUIRE 843
ISOLATION 846
OWNER 148
RELEASE 843
rebinding
after creating an index 94
after dropping a view 96
automatically
EXPLAIN processing 937
REBUILD INDEX utility
effect on real-time statistics 1248
REBUILD-pending status
description for indexes 473
record
performance considerations 113
size 113
RECORDING MAX field of panel DSNTIPA
preventing frequent BSDS wrapping 609
RECOVER BSDS command
copying good BSDS 436
RECOVER INDOUBT command
free locked resources 528
recover indoubt thread 465
RECOVER privilege
description 136
RECOVER TABLESPACE utility
DFSMSdss concurrent copy 491
recovers data modified after shutdown 612
RECOVER utility
cannot use with work file table space 493
catalog and directory tables 494
data inconsistency problems 486
deferred objects during restart 451
functions 491
kinds of objects 491
messages issued 491
options
TOCOPY 500
TOLOGPOINT 500
RECOVER utility *(continued)*
options *(continued)*
TOLOGPOINT in application program error 520
TORBA 501
problem on DSNDB07 493
recovers pages in error 370
running in parallel 488
use of fast log apply during processing 488
RECOVER utility, DFSMSdss RESTORE command 36
RECOVERDB privilege
description 135
recovery
BSDS 539
catalog and directory 494
data set
using DFSMS 491
using DFSMShsm 477
using non-DB2 dump and restore 504
database
active log 1111
using a backup copy 473
using RECOVER TOCOPY 500
using RECOVER TOLOGPOINT 500
using RECOVER TORBA 501
down-level page sets 543
dropped objects 505
dropped table 505
dropped table space 507
IFI calls 1184
indexes 473
indoubt threads 582
indoubt units of recovery
CICS 385, 527
IMS 391
media 492
minimizing outages 477
multiple systems environment 458
operation 474
point in time 500
prior point of consistency 495
real-time statistics tables 1253
reducing time 476
reporting information 480
restart 482, 611
scenarios
See recovery scenarios
subsystem 1111
system procedures 471
table space
COPY 504
dropped 507
DSN1COPY 504
point in time 482
QUIESCE 482
RECOVER TOCOPY 500
RECOVER TOLOGPOINT 500
RECOVER TORBA 501
scenario 545
work file table space 493
recovery log
description 8
record formats 1119
RECOVERY option of REPORT utility 520
recovery scenarios
application program error 520
CICS-related failures
application failure 525

recovery scenarios *(continued)*
CICS-related failures *(continued)*
attachment facility failure 530
manually recovering indoubt units of recovery 527
not operational 525
DB2-related failures
active log failure 531
archive log failure 535
BSDS 538
catalog or directory I/O errors 546
database failures 542
subsystem termination 530
system resource failures 531
table space I/O errors 545
disk failure 518
failure during log initialization or current status rebuild 593
IMS-related failures 521
application failure 524
control region failure 522
fails during indoubt resolution 522
indoubt threads 582
integrated catalog facility catalog VVDS failure 547
invalid LOB 544
IRLM failure 517
out of space 548
restart 591
starting 324
z/OS failure 518
RECP (RECOVERY-pending) status
description 63
redefining an index-based partition 552
redefining an table-based partition 552
redo log records 1112
REFERENCES privilege
description 134
referential constraint
adding to existing table 77
data consistency 294
recovering from violating 552
referential structure, maintaining consistency for recovery 487
refresh age 890
REFRESH TABLE statement 887
registering a base table as 884
registration tables for DDL
adding columns 216, 228
CREATE statements 226
creating 216
escape character 219
examples 220, 225
function 215, 227
indexes 216
managing 216
pattern characters 219
preparing for recovery 473
updating 228
relative byte address (RBA)
See RBA (relative byte address)
relative byte address (RBA).
See RBA (relative byte address)
RELCURHL subsystem parameter
recommendation 841
RELEASE
option of BIND PLAN subcommand
combining with other options 843
RELEASE LOCKS field of panel DSNTIP4
effect on page and row locks 857
RELEASE LOCKS field of panel DSNTIP4 *(continued)*
recommendation 841
remote logical unit, failure 556
remote request 242, 252
reoptimizing access path 774
REORG privilege
description 135
REORG UNLOAD EXTERNAL 106
REORG utility
effect on real-time statistics 1246
examples 87
moving data 106
REPAIR privilege
description 135
REPAIR utility
resolving inconsistent data 618
REPORT utility
options
RECOVERY 520
TABLESPACESET 520
table space recovery 480
REPRO command of access method services 504, 539
RESET INDOUBT command
reset indoubt thread 466
residual recovery entry (RRE)
See RRE (residual recovery entry)
Resource Access Control Facility (RACF)
See RACF (Resource Access Control Facility)
resource allocation 731
performance factors 731
resource control table (RCT)
See RCT (resource control table)
resource limit facility (governor)
calculating service units 726
database 10
description 716
distributed environment 716
governing by plan or package 723
preparing for recovery 473
specification table (RLST)
See RLST (resource limit specification table)
stopping and starting 718
resource manager
resolution of indoubt units of recovery 461
Resource Measurement Facility (RMF) 1185, 1188
resource objectives 714
Resource Recovery Services (RRS), controlling connections 397
Resource Recovery Services attachment facility (RRSAF)
RACF profile 274
stored procedures and RACF authorization 274
RESOURCE TIMEOUT field of panel DSNTIPI 833
resource translation table (RTT)
See RTT (resource translation table)
resources
defining to RACF 265
limiting 715
response time 661
restart
See also conditional restart
See also restarting
automatic 449
backward log recovery
failure during 607
phase 448
cold start situations 613

restart *(continued)*
conditional
control record governs 451
excessive loss of active log data 614
total loss of log 613
current status rebuild
failure during 593
phase 446
data object availability 450
DB2 443
deferring processing
objects 450
effect of lost connections 459
forward log recovery
failure during 602
phase 447
log initialization
failure during 593
phase 445
multiple systems environment 458
normal 444
overriding automatic 450
preparing for recovery 482
recovery operations for 453
resolving inconsistencies after 616
unresolvable
BSDS problems during 610
log data set problems during 610
RESTART ALL field of panel DSNTIPS 450
RESTORE phase of RECOVER utility 492
RESTORE SYSTEM utility 37
restoring data to a prior level 495
RETAINED LOCK TIMEOUT field of installation panel DSNTIPI 834
RETLWAIT subsystem parameter 834
REVOKE statement
cascading effect 180
delete a view 185
examples 180, 187
format 180
invalidates a plan or package 186
privileges required 164
revoking SYSADM authority 186
RID (record identifier) pool
size 683
storage
allocation 683
estimation 683
use in list prefetch 970
RLFASUERR column of RLST 721
RLFASUWARN column of RLST 721
RLST (resource limit specification table)
columns 719
creating 717
distributed processing 725
precedence of entries 722
RMF (Resource Measurement Facility) 1185, 1188
RO SWITCH CHKPTS field of installation panel DSNTIPL 691
RO SWITCH TIME field of installation panel DSNTIPL 691
rollback
effect on performance 697
maintaining consistency 457
unit of recovery 424
root page
description 117
index 117
route codes for messages 323
routine
example, authorization 157
plans, packages 155
retrieving information about authorization IDs 190
routine privileges 136
row
formats for exit routines 1106
validating 1080
row-level security
security label column 198
using SQL statements 198
ROWID
index-only access 942
ROWID column
inserting 65
loading data into 62
RR (repeatable read)
claim class 865
drain lock 866
effect on locking 848
how locks are held (figure) 852
page and row locking 852
RRDF (Remote Recovery Data Facility)
altering a table for 87
RRE (residual recovery entry)
detect 392
logged at IMS checkpoint 460
not resolved 460
purge 392
RRSAF (Recoverable Resource Manager Services attachment facility)
application program
authorization 151
transactions
using global transactions 817
RRSAF (Resource Recovery Services attachment facility)
application program
running 331
RS (read stability)
claim class 865
effect on locking 847
page and row locking (figure) 851
RTT (resource translation table)
transaction type 394
RUN
subcommand of DSN
example 327
RUNSTATS utility
aggregate statistics 912
effect on real-time statistics 1249
timestamp 915
use
tuning DB2 651
tuning queries 912
RVA (RAMAC Virtual Array)
backup 491

S

sample application
databases, for 1048
structure of 1047
sample exit routine
connection
location 1052
processing 1057

sample exit routine *(continued)*
connection *(continued)*
supplies secondary IDs 234
edit 1077
sign-on
location 1052
processing 1057
supplies secondary IDs 237
sample library
See SDSNSAMP library
sample security plan
new application 174, 180
sample table 1031
DSN8810.ACT (activity) 1031
DSN8810.DEMO_UNICODE (Unicode sample) 1041
DSN8810.DEPT (department) 1032
DSN8810.EMP (employee) 1034
DSN8810.EMP_PHOTO_RESUME (employee photo and resume) 1037
DSN8810.EMPPROJACT (employee-to-project activity) 1040
DSN8810.PROJ (project) 1038
PROJACT (project activity) 1039
views on 1042
SBCS data
altering subtype 88
scheduled tasks
checking status 347
defining 337
listing 347
removing 335, 348
schema
privileges 136
schema definition
authorization to process 59
description 58
example of processor input 59
processing 59
processor 58
scope of a lock 817
SCOPE option
START irlmproc command 832
scrollable cursor
block fetching 1007
optimistic concurrency control 848
performance considerations 785
SCT02 table space
description 8
placement of data sets 692
SDSNLOAD library
loading 393
SDSNSAMP library
processing schema definitions 59
SECACPT option of APPL statement 243
secondary authorization ID
See authorization ID, secondary
SECQTY1 column
SYSINDEXPART_HIST catalog table 910
SECQTYI column
SYSINDEXPART catalog table 903
SYSTABLEPART catalog table 905
SYSTABLEPART_HIST catalog table 911
SecureWay Security Server for OS/390 19
security
acceptance options 243
access to
data 127
security *(continued)*
access to *(continued)*
DB2 data sets 281
administrator privileges 171
authorizations for stored procedures 156
CICS 279
closed application 215, 227
DDL control registration tables 215
description 127
IMS 279
measures in application program 153
measures in force 290
mechanisms 238
planning 127
sample security plan 301
system, external 231
security administrator 171
security label
definition 193
security label column 198
segment of log record 1116
segmented table space
locking 818
scan 948
SEGSIZE clause of CREATE TABLESPACE
recommendations 948
SELECT privilege
description 134
SELECT statement
example
SYSIBM.SYSPLANDEP 90
SYSIBM.SYSTABLEPART 68
SYSIBM.SYSVIEWDEP 90
sequence
privileges of ownership 146
sequences
improving concurrency 816
sequential detection 972, 973
sequential prefetch
bind time 970
description 969
sequential prefetch threshold (SPTH) 668
SET ARCHIVE command
description 320
SET CURRENT DEGREE statement 993
SET CURRENT SQLID statement 134
SET ENCRYPTION PASSWORD 208
SHARE
INTENT EXCLUSIVE lock mode 822, 862
lock mode
LOB 862
page 821
row 821
table, partition, and table space 821
SHDDEST option of DSNCRCT macro 332
sign-on
exit point 1053
exit routine.
See sign-on exit routine
processing
See sign-on processing
requests 1053
sign-on exit routine
debugging 1059
default 237
description 1051
performance considerations 1058

sign-on exit routine *(continued)*
sample 237
location 1052
provides secondary IDs 1057
secondary authorization ID 237
using 237
writing 1051
sign-on processing
choosing for remote requests 243
initial primary authorization ID 236
invoking RACF 236
requests 232
supplying secondary IDs 237
usage 232
using exit routine 237
SIGNON-ID option of IMS 329
simple table space
locking 818
single logging 9
SKCT (skeleton cursor table)
description 8
EDM pool 679
EDM pool efficiency 681
locks on 825
skeleton cursor table (SKCT)
See SKCT (skeleton cursor table)
skeleton package table (SKPT)
See SKPT (skeleton package table)
SKPT (skeleton package table)
description 8
EDM pool 679
locks on 825
SMF (System Management Facility)
buffers 1194
measured usage pricing 661
record types 1190, 1192
trace record
accounting 1192
auditing 289
format 1135
lost records 1194
recording 1194
statistics 1190
type 89 records 661
SMS (Storage Management Subsystem)
See DFSMS (Data Facility Storage Management Subsystem)
SMS archive log data sets 430
SNA
mechanisms 238
protocols 242
software protection 292
sort
description 684
pool 684
program
reducing unnecessary use 707
RIDs (record identifiers) 974
when performed 974
removing duplicates 974
shown in PLAN_TABLE 973
SORT POOL SIZE field of panel DSNTIPC 684
sorting sequence, altering by a field procedure 1089
space attributes 69
specifying 82
SPACE column
SYSTABLEPART catalog table 905
SPACE column of SYSTABLESPACE catalog table
data collected by RUNSTATS utility 906
space reservation options 652
SPACEF column
SYSINDEXES catalog table 902
SYSINDEXPART catalog table 903
SYSINDEXPART_HIST catalog table 910
SYSTABLEPART catalog table 905
SYSTABLEPART_HIST catalog table 911
SYSTABLES catalog table 905
SPACEF column of SYSTABLESPACE catalog table
data collected by RUNSTATS utility 906
SPACENAM option
DISPLAY DATABASE command 368
START DATABASE command 365
speed, tuning DB2 651
SPT01 table space 8
SPTH (sequential prefetch threshold) 668
SPUFI
disconnecting 383
resource limit facility (governor) 723
SQL (Structured Query Language)
performance trace 731
statement cost 732
statements
See SQL statements
performance factors 732
transaction unit of recovery 423
SQL authorization ID
See authorization ID, SQL
SQL statements
DECLARE CURSOR
to ensure block fetching 1007
EXPLAIN
monitor access paths 927
RELEASE 1005
SET CURRENT DEGREE 993
SQLCA (SQL communication area)
reason code for deadlock 812
reason code for timeout 811
SQLCODE
-30082 239
-510 856
-905 722
SQLSTATE
'08001' 239
'57014' 722
SQTY column
SYSINDEXPART catalog table 903
SYSTABLEPART catalog table 905
SSM (subsystem member)
error options 394
specified on EXEC parameter 393
thread reuse 742
SSR command of IMS
entering 320
prefix 333
stand-alone utilities
recommendation 376
standard, SQL (ANSI/ISO)
schemas 58
star join 963
dedicated virtual memory pool 967
star schema
defining indexes for 796
START DATABASE command
example 364

START DATABASE command *(continued)*
problem on DSNDB07 493
SPACENAM option 365
START DB2 command
description 325
entered from z/OS console 324
mode identified by reason code 396
PARM option 325
restart 451
START FUNCTION SPECIFIC command
starting user-defined functions 373
START REGION command of IMS 396
START SUBSYS command of IMS 388
START TRACE command
AUDIT option 286
controlling data 419
STARTDB privilege
description 135
started procedures table in RACF 270
started-task address space 267
starting
audit trace 286
databases 364
DB2
after an abend 326
process 324
IRLM
process 377
table space or index space having restrictions 365
user-defined functions 373
state
of a lock 821
statement table
column descriptions 982
static SQL
privileges required 164
statistics
aggregate 912
created temporary tables 908
distribution 915
filter factor 906
history catalog tables 909, 913
materialized query tables 888
partitioned table spaces 908
trace
class 4 1013
description 1190
STATS privilege
description 135
STATSTIME column
use by RUNSTATS 900
status
CHECK-pending
resetting 63
COPY-pending, resetting 63
STATUS
column of DISPLAY DATABASE report 366
STDDEV function
when evaluation occurs 946
STOGROUP privilege
description 138
STOP DATABASE command
example 372
problem on DSNDB07 493
SPACENAM option 365
timeout 811
STOP DDF command
description 417
STOP FUNCTION SPECIFIC command
stopping user-defined functions 374
STOP REGION command of IMS 396
STOP SUBSYS command of IMS 388, 396
STOP TRACE command
AUDIT option 286
description 419
STOP transaction type 394
STOPALL privilege
description 136
STOPDB privilege
description 135
stopping
audit trace 286
data definition control 228
databases 371
DB2 327
IRLM 378
user-defined functions 374
storage
auxiliary 29
calculating
locks 833
controller cache 709
external
See auxiliary storage
hierarchy 708
IFI requirements
READA 1171
READS 1158
real 708
space of dropped table, reclaiming 89
using DFSMShsm to manage 35
storage controller cache 709
storage group, DB2
adding volumes 67
altering 67
assigning objects to 29
changing to SMS-managed 67
changing to use a new high-level qualifier 104
creating 29
default group 29
definition 27
description 5, 29
managing
control interval sizing 28
deferring allocation 28
defining data sets 27
deleting data sets 28
extending data sets 28
moving data sets 28
reusing data sets 28
using SMS 30
moving data 108
order of use 29
privileges of ownership 146
SYSDEFLT 29
storage group, for sample application data 1048
storage management subsystem
See DFSMS (Data Facility Storage Management Subsystem)
stored procedure
address space 268
altering 96
authority to access non-DB2 resources 277
authorizations 155, 156

stored procedure *(continued)*
commands 413
DNSLEUSR 1287
DSNACCOR 1257
DSNACICS 1280
DSNAEXP 1295
example, authorization 157
IMS transsactions 1291
limiting resources 716
monitoring using accounting trace 1022
privileges
of ownership 146
RACF protection for 274
running concurrently 1019
WLM_REFRESH 1278
stored procedures
ADMIN_TASK_ADD 339
ADMIN_TASK_REMOVE 349
scheduled execution 360
STOSPACE privilege
description 136
string conversion exit routine.
See conversion procedure
subquery
correlated
tuning 780
join transformation 782
noncorrelated 781
tuning 780
tuning examples 784
subsystem
controlling access 132, 231
recovery 1111
termination scenario 530
subsystem command prefix 11
subsystem member (SSM)
See SSM (subsystem member)
subtypes 88
synchronous data from IFI 1167
synchronous write
analyzing accounting report 644
immediate 668, 679
synonym
privileges of ownership 146
syntax diagram
how to read xxvi
SYS1.LOGREC data set 531
SYS1.PARMLIB library
specifying IRLM in IEFSSNxx member 376
SYSADM authority
description 142
revoking 186
SYSCOPY
catalog table, retaining records in 510
SYSCTRL authority
description 141
SYSIBM.ADMIN_TASKS 356
SYSIBM.IPNAMES table of CDB
remote request processing 254
translating outbound IDs 254
SYSIBM.LUNAMES table of CDB
accepting inbound remote IDs 240, 253
dummy row 243
remote request processing 240, 253
sample entries 247
translating inbound IDs 247
translating outbound IDs 240, 253
SYSIBM.LUNAMES table of CDB *(continued)*
verifying attachment requests 243
SYSIBM.USERNAMES table of CDB
managing inbound remote IDs 243
remote request processing 240, 253
sample entries for inbound translation 247
sample entries for outbound translation 261
translating inbound and outbound IDs 240, 253
SYSLGRNX directory table
information from the REPORT utility 480
table space
description 8
retaining records 510
SYSOPR authority
description 140
usage 324
Sysplex query parallelism
disabling Sysplex query parallelism 1001
disabling using buffer pool threshold 669
processing across a data sharing group 991
splitting large queries across DB2 members 987
system
management functions, controlling 418
privileges 136
structures 7
system administrator
description 171
privileges 175
System Management Facility (SMF)
See SMF (System Management Facility)
System Management Facility (SMF).
See SMF (System Management Facility)
system monitoring
monitoring tools
DB2 trace 1189
system operator
See SYSOPR authority
system programmer 172
system-directed access
authorization at second server 150
SYSUTILX directory table space 8

T

table
altering
adding a column 71
auditing 288
creating
description 48
description 6
dropping
implications 89
estimating storage 112
expression, nested
processing 975
locks 817
ownership 146
populating
loading data into 61
privileges 134, 146
qualified name 146
re-creating 90
recovery of dropped 505
registration, for DDL 215, 227
retrieving
IDs allowed to access 189

table *(continued)*
plans and packages that can access 190
types 48
table check constraints
adding 80
dropping 80
table expressions, nested
materialization 976
table space
compressing data 702
copying 489
creating
default database 45
default name 45
default space allocation 45
default storage group 45
description 44
explicitly 44
implicitly 45
deferring allocation of data sets 30
description 5
dropping 69
EA-enabled 42
for sample application 1048
loading data into 61
locks
control structures 731
description 817
maximum addressable range 44
privileges of ownership 146
quiescing 482
re-creating 69
recovery
See recovery, table space
recovery of dropped 507
reorganizing 76
scans
access path 947
determined by EXPLAIN 928
versions 76
recycling version numbers 77
table-controlled partitioning
automatic conversion to 52
contrasted with index-controlled partitioning 52
implementing 51
using nullable partitioning columns 53
tables 1199
tables used in examples 1031
TABLESPACE privilege
description 138
TABLESPACESET option of REPORT utility 520
TABLESPACESTATS
contents 1231
real-time statistics table 1230
TCP/IP
authorizing DDF to connect 278
keep_alive interval 738
protocols 249
temporary table
monitoring 693
thread reuse 733
temporary work file
See work file
TERM UTILITY command
when not to use 487
terminal monitor program (TMP)
See TMP (terminal monitor program)
terminating
See also stopping
DB2
abend 444
concepts 443
normal 443
normal restart 444
scenario 530
thread
allied 400
attachment in IMS 389
CICS
access to DB2 386
creation
CICS 742
connections 743
description 730
IMS 742
performance factors 730
database access
description 400
displaying
CICS 386
IMS 394
distributed
active 737
inactive vs. active 736
maximum number 735
pooling of inactive threads 736
monitoring in 386
queuing 743
reuse
CICS 742
description 730
effect on processor resources 660
IMS 742
remote connections 739
TSO 733
when to use 733
steps in creation and termination 730
termination
CICS 384, 742
description 733
IMS 390, 396, 742
time out for idle distributed threads 737
type 2, storage usage 707
TIME FORMAT field of panel DSNTIPF 1084
time routine
description 1083
writing 1083
timeout
changing multiplier
IMS BMP and DL/I batch 834
utilities 836
description 811
idle thread 737
multiplier values 834
row vs. page locks 839
X'00C9008E' reason code in SQLCA 811
Tivoli Decision Support for OS/390 1197
TMP (terminal monitor program)
DSN command processor 381
sample job 330
TSO batch work 330
TOCOPY option of RECOVER utility 500
TOKENE option of DSNCRCT macro 650
TOLOGPOINT option of RECOVER utility 500

TORBA option of RECOVER utility 501
trace
accounting 1190
audit 1192
controlling
DB2 419
IMS 420
description 1185, 1189
diagnostic
CICS 420
IRLM 420
distributed data 1013
effect on processor resources 660
interpreting output 1135
monitor 1192
performance 1192
recommendation 1013
record descriptions 1135
record processing 1135
statistics
description 1190
TRACE privilege
description 136
TRACE SUBSYS command of IMS 388
tracker site 568
transaction
CICS
accessing 386
DSNC codes 321
entering 329
IMS
connecting to DB2 388
entering 328
thread attachment 389
thread termination 390
using global transactions 817
SQL unit of recovery 423
transaction lock
description 809
transaction manager
coordinating recovery of distributed transactions 461
TRANSACTION option
DSNC DISPLAY command 386
transaction types 394
translating
inbound authorization IDs 247
outbound authorization IDs 260
truncation
active log 427, 601
TSO
application programs
batch 17
conditions 327
foreground 17
running 327
background execution 330
commands issued from DSN session 328
connections
controlling 380
DB2 381
disconnecting from DB2 383
monitoring 381
tuning 743
DB2 considerations 17
DSNELI language interface module
IFI 1153
link editing 327
TSO *(continued)*
entering DB2 commands 321
environment 327
foreground 733
requirement 17
resource limit facility (governor) 715
running SQL 733
tuning
DB2
active log size 698
catalog location 693
catalog size 693
disk utilization 701
queries containing host variables 774
speed 651
virtual storage utilization 706
two-phase commit
illustration 455
process 455
TYPE column
SYSCOLDI 909
SYSCOLDIST catalog table
access path selection 900
SYSCOLDISTSTATS catalog table 900

U

undo log records 1112
Unicode
sample table 1041
UNION clause
effect on OPTIMIZE clause 791
removing duplicates with sort 974
unit of recovery
description 423
ID 1119
illustration 423
in-abort
backward log recovery 448
description 458
excluded in forward log recovery 447
in-commit
description 457
included in forward log recovery 447
indoubt
causes inconsistent state 444
definition 326
description 457
displaying 390, 528
included in forward log recovery 447
recovering CICS 385
recovering IMS 391
recovery in CICS 527
recovery scenario 522
resolving 460, 461, 466
inflight
backward log recovery 448
description 457
excluded in forward log recovery 447
log records 1112
postponed
displaying 391
postponed abort 458
rollback 424, 457
SQL transaction 423
unit of recovery ID (URID) 1119

UNLOAD utility
delimited files 62
unqualified objects, ownership 145
unsolicited output
CICS 323, 332
IMS 323
operational control 332
subsystem messages 332
UPDATE
lock mode
page 821
row 821
table, partition, and table space 821
update efficiency 678
UPDATE privilege
description 134
updating
registration tables for DDL 228
UR (uncommitted read)
claim class 865
concurrent access restrictions 853
effect on locking 847
effect on reading LOBs 861
page and row locking 850
recommendation 816
URID (unit of recovery ID).
See unit of recovery
USAGE privilege
distinct type 138
Java class 138
sequence 138
USE AND KEEP EXCLUSIVE LOCKS option of WITH clause 858
USE AND KEEP SHARE LOCKS option of WITH clause 858
USE AND KEEP UPDATE LOCKS option of WITH clause 858
USE OF privileges 138
user analyst 171
user-defined function
controlling 373
START FUNCTION SPECIFIC command 373
example, authorization 157
monitoring 374
privileges of ownership 146
providing access cost 1021
starting 373
stopping 374
user-defined functions
altering 96
controlling 373
user-defined table function
improving query performance 793
user-managed data sets
changing high-level qualifier 104
extending 41
name format 38
requirements 38
specifying data class 42
utilities
access status needed 375
compatibility 867
concurrency 809, 864
controlling 375
description 11
effect on real-time statistics 1244
executing
running on objects with pages in LPL 370
utilities *(continued)*
internal integrity reports 299
timeout multiplier 836
types
RUNSTATS 912
UTILITY TIMEOUT field of panel DSNTIPI 836
UTSERIAL lock 867

V

validating
connections from remote application 238
existing rows with a new VALIDPROC 87
rows of a table 1080
validation routine
altering assignment 86
checking existing table rows 87
description 294
ensuring data accuracy 294
row formats 1106
writing 1080
validation routine.
See also VALIDPROC clause
description 1080
VALIDPROC clause
ALTER TABLE statement 86
exit points 1081
value
descriptors in field procedures 1094
VARCHAR
data type
subtypes 88
VARIANCE function
when evaluation occurs 946
VARY NET command of VTAM
TERM option 412
varying-length records
effect on processor resources 661
VDWQT option of ALTER BUFFERPOOL command 669
verifying VTAM partner LU 242
vertical deferred write threshold (VDWQT) 669
view
altering 96
creating
on catalog tables 191
description 7
dropping
deleted by REVOKE 185
invalidates plan or package 96
EXPLAIN 978, 979
list of dependent objects 90
name
qualified name 146
privileges
effect of revoking table privileges 185
ownership 146
table privileges for 143
processing
view materialization description 976
view materialization in PLAN_TABLE 944
view merge 975
reasons for using 7
virtual buffer pool assisting parallel sequential threshold (VPXPSEQT) 669
virtual buffer pool parallel sequential threshold (VPPSEQT) 669
virtual buffer pool sequential steal threshold (VPSEQT) 669

virtual storage
  buffer pools 706
  improving utilization 706
  IRLM 706
virtual storage access method (VSAM)
  See VSAM (virtual storage access method)
Virtual Telecommunications Access Method (VTAM)
  See VTAM (Virtual Telecommunications Access Method)
Visual Explain 789, 926, 927
volatile table 792
volume serial number 437
VPPSEQT option of ALTER BUFFERPOOL command 669
VPSEQT option of ALTER BUFFERPOOL command 669
VPXPSEQT option of ALTER BUFFERPOOL command 669
VSAM (virtual storage access method)
  control interval
    block size 428
    log records 425
    processing 504
  volume data set (VVDS) recovery scenario 547
VTAM (Virtual Telecommunications Access Method)
  APPL statement
    See APPL statement
  commands
    VARY NET,TERM 412
  controlling connections 242, 267
  conversation-level security 243
  partner LU verification 242
  password
    choosing 242
VVDS recovery scenario 547

W

wait state at start 325
WBUFxxx field of buffer information area 1155
WebSphere
  description, attachment facility 15
WebSphere Application Server
  identify outstanding indoubt units of recovery 461
WITH clause
  specifies isolation level 858
WITH HOLD cursor
  effect on locks and claims 857
WLM_REFRESH stored procedure
  description 1278
  option descriptions 1279
  sample JCL 1280
  syntax diagram 1279
work file
  table space
    minimize I/O contention 657
  used by sort 685
work file database
  changing high-level qualifier 103
  description 10
  enlarging 552
  error range recovery 493
  minimizing I/O contention 657
  problems 493
  starting 364
  used by sort 706
Workload Manager 739
WQAxxx fields of qualification area 1124, 1159
write claim class 865
write drain lock 866
write efficiency 678
write error page range (WEPR) 369
WRITE function of IFI 1173
WRITE TO OPER field of panel DSNTIPA 427
write-down control 195

X

XLKUPDLT subsystem parameter 841
XRC (Extended Remote Copy) 581
XRF (extended recovery facility)
  CICS toleration 472
  IMS toleration 472

Z

z/OS
  command group authorization level (SYS) 320, 323
  commands
    MODIFY irlmproc 378
    STOP irlmproc 378
  DB2 considerations 13
  entering DB2 commands 320, 323
  environment 13
  IRLM commands control 318
  performance options 711
  power failure recovery scenario 518
  workload manager 739

Program Number: 5625-DB2

Printed in USA

SC18-7413-05