DB2 10 for z/OS

Administration Guide

SC19-2968-14

DB2 10 for z/OS

Administration Guide

SC19-2968-14

Note
Before using this information and the product it supports, be sure to read the general information under "Notices" at the end of this information.

Fifteenth edition (October 2016)

This edition applies to DB2 10 for z/OS (product number 5605-DB2), DB2 10 for z/OS Value Unit Edition (product number 5697-P31), 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.

Specific changes are indicated by a vertical bar to the left of a change. A vertical bar to the left of a figure caption indicates that the figure has changed. Editorial changes that have no technical significance are not noted.

© Copyright IBM Corporation 1982, 2016.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

About this information. xiii
Who should read this information xiii
DB2 Utilities Suite xiii
Terminology and citations xiv
Accessibility features for DB2 10 for z/OS xiv
How to send your comments xv
How to read syntax diagrams xv

Part 1. Designing a database 1

Chapter 1. Database objects and relationships 3
Logical database design with the entity-relationship model 3
Modeling your data 3
Recommendations for logical data modeling 5
Practical examples of data modeling 5
Entities for different types of relationships 6
Entity attributes 7
Entity normalization 9
Logical database design with Unified Modeling Language 14
Physical database design 15
Denormalization of tables 16
Views to customize what data users see 18
Indexes on table columns 19
| Hash access on tables 19

Chapter 2. Implementing your database design 21
Implementing DB2 databases 21
Creating DB2 databases 21
Dropping DB2 databases 22
Implementing DB2 storage groups 22
Advantages of storage groups 22
Creating DB2 storage groups 24
Enabling SMS to control DB2 storage groups 24
Deferring allocation of DB2-managed data sets 25
How DB2 extends data sets 26
DB2 space allocation 27
Managing DB2 data sets with DFSMShsm 31
Managing your own data sets 35
Defining index space storage 39
Creating EA-enabled table spaces and index spaces 40
Implementing DB2 table spaces 41
Implicitly defined table spaces 42
| Creating table spaces explicitly 42
Implementing DB2 tables 46
Creating base tables 46
Guidelines for table names 47
| Creating tables that use hash organization 47
Creating temporary tables 48
| Creating temporal tables 53
Creating materialized query tables 61
Creating tables that use table-controlled partitioning 61
Creating tables that use index-controlled partitioning 66
Creating a clone table 66
Implementing DB2 views 69

© Copyright IBM Corp. 1982, 2016

Creating DB2 views . . . 69
Guidelines for view names . . . 71
How DB2 inserts and updates data through views . . . 71
Dropping DB2 views . . . 72
Implementing DB2 indexes . . . 72
Creating DB2 indexes . . . 72
Guidelines for defining indexes . . . 73
How DB2 implicitly creates an index . . . 74
Index versions . . . 75
Implementing DB2 schemas . . . 76
Creating a schema by using the schema processor . . . 76
Processing schema definitions . . . 77
Loading data into DB2 tables . . . 78
Loading data with the LOAD utility . . . 79
Loading data by using the INSERT statement . . . 81
Loading data from DL/I . . . 83
Implementing DB2 stored procedures . . . 84
Creating stored procedures . . . 84
Dropping stored procedures . . . 85
Implementing DB2 user-defined functions . . . 86
Creating user-defined functions . . . 86
Deleting user-defined functions . . . 87
Estimating disk storage for user data . . . 87
General approach to estimating storage . . . 88
Calculating the space required for a table . . . 90
Calculating the space required for an index . . . 93

Chapter 3. Altering your database design . . . 101
Using the catalog in database design . . . 101
Retrieving catalog information about DB2 storage groups . . . 102
Retrieving catalog information about a table . . . 102
Retrieving catalog information about partition order . . . 103
Retrieving catalog information about aliases . . . 103
Retrieving catalog information about columns . . . 104
Retrieving catalog information about indexes . . . 105
Retrieving catalog information about views . . . 105
Retrieving catalog information about authorizations . . . 106
Retrieving catalog information about primary keys . . . 107
Retrieving catalog information about foreign keys . . . 107
Retrieving catalog information about check pending . . . 108
Retrieving catalog information about check constraints . . . 109
Retrieving catalog information about LOBs . . . 110
Retrieving catalog information about user-defined functions and stored procedures . . . 110
Retrieving catalog information about triggers . . . 111
Retrieving catalog information about sequences . . . 111
Adding and retrieving comments . . . 112
Verifying the accuracy of the database definition . . . 113
Altering DB2 databases . . . 113
ALTER DATABASE options . . . 113
Altering DB2 storage groups . . . 114
Letting SMS manage your DB2 storage groups . . . 114
Adding or removing volumes from a DB2 storage group . . . 115
| Migrating existing data sets to a solid-state drive . . . 116
Altering table spaces . . . 117
Changing the logging attribute . . . 118
Changing the space allocation for user-managed data sets . . . 119
Dropping, re-creating, or converting a table space . . . 120
Redistributing data in partitioned table spaces . . . 122
| Increasing the partition size of a partitioned table space . . . 123
Altering a page set to contain DB2-defined extents . . . 124
Altering DB2 tables . . . 125

Adding a column to a table . . . 126
Specifying a default value when altering a column . . . 128
Altering the data type of a column . . . 128
Altering a table for referential integrity . . . 134
Adding or dropping table check constraints . . . 139
Adding partitions . . . 140
Altering partitions . . . 142
Adding XML columns . . . 149
Altering tables to enable hash access . . . 150
Altering the size of your hash spaces . . . 152
Adding a system period and system-period data versioning to an existing table . . . 153
Adding an application period to a table . . . 154
Manipulating data in a system-period temporal table . . . 155
Altering materialized query tables . . . 155
Altering the assignment of a validation routine . . . 158
Altering a table to capture changed data . . . 159
Changing an edit procedure or a field procedure . . . 159
Altering the subtype of a string column . . . 160
Altering the attributes of an identity column . . . 160
Changing data types by dropping and re-creating the table . . . 161
Moving a table to a table space of a different page size . . . 165
Altering DB2 views . . . 165
Altering views by using the INSTEAD OF trigger . . . 166
Altering DB2 indexes . . . 166
Alternative method for altering an index . . . 167
Adding columns to an index . . . 168
Altering how varying-length index columns are stored . . . 170
Altering the clustering of an index . . . 170
Dropping and redefining a DB2 index . . . 171
Reorganizing indexes . . . 172
Recycling index version numbers . . . 172
Pending data definition changes . . . 173
Materializing pending definition changes . . . 176
Restrictions for changes to objects that have pending data definition changes . . . 180
Altering stored procedures . . . 181
Altering user-defined functions . . . 183
Altering implicitly created XML objects . . . 184
Changing the high-level qualifier for DB2 data sets . . . 185
Defining a new integrated catalog alias . . . 185
Changing the qualifier for system data sets . . . 185
Changing qualifiers for other databases and user data sets . . . 189
Tools for moving DB2 data . . . 193
Moving DB2 data . . . 195
Moving a DB2 data set . . . 196
Scenario: Moving from index-controlled to table-controlled partitioning . . . 198

Part 2. Operation and recovery . . . 201

Chapter 4. Controlling DB2 operations by using commands . . . 203
Issuing commands from the z/OS console . . . 205
Issuing commands from TSO terminals . . . 205
Issuing commands from CICS terminals . . . 207
Issuing commands from IMS terminals . . . 207
Issuing commands from application programs . . . 208
Destinations for command output messages . . . 209
Unsolicited DB2 messages . . . 210

Chapter 5. Starting and stopping DB2 . . . 211
Starting DB2 . . . 211
Messages at start . . . 211

Subsystem parameters at start . . . 212
Application defaults module name at start . . . 213
Restricting access to data . . . 213
Ending the wait state at startup . . . 213
Restart options after an abend . . . 214
Stopping DB2 . . . 214

Chapter 6. Submitting work to DB2 . . . 217
Submitting work by using DB2I . . . 217
Running TSO application programs . . . 217
Sources that DB2 checks to find authorization access for an application program . . . 218
Running IMS application programs . . . 218
Running CICS application programs. . . . 219
Running batch application programs . . . 220
Running application programs using CAF . . . 221
Running application programs using RRSAF . . . 221

Chapter 7. Scheduling administrative tasks . . . 223
Interacting with the administrative task scheduler . . . 223
Adding a task . . . 223
Listing scheduled tasks . . . 229
Listing the status of scheduled tasks. . . . 229
Updating the schedule for a task . . . 232
Stopping the execution of a task . . . 232
Removing a scheduled task. . . . 232
Manually starting the administrative task scheduler . . . 233
Manually stopping the administrative task scheduler . . . 233
Synchronization between administrative task schedulers in a data sharing environment . . . 234
Troubleshooting the administrative task scheduler . . . 235
Architecture of the administrative task scheduler . . . 238
The lifecycle of the administrative task scheduler . . . 239
Task lists of the administrative task scheduler . . . 241
Architecture of the administrative task scheduler in a data sharing environment . . . 241
Accounting information for stored procedure tasks . . . 242
Security guidelines for the administrative task scheduler . . . 243
User roles in the administrative task scheduler . . . 244
Protection of the interface of the administrative task scheduler . . . 245
Protection of the resources of the administrative task scheduler . . . 245
Secure execution of tasks in the administrative task scheduler . . . 246
Execution of scheduled tasks in the administrative task scheduler . . . 247
Multi-threading in the administrative task scheduler . . . 247
Scheduling execution of a stored procedure . . . 249
How the administrative task scheduler works with Unicode . . . 250
Scheduled execution of a JCL job . . . 250
Execution of scheduled tasks in a data sharing environment . . . 251
Time zone considerations for the administrative task scheduler . . . 252

Chapter 8. Monitoring and controlling DB2 and its connections . . . 253
Controlling DB2 databases and buffer pools . . . 253
Starting databases . . . 254
Monitoring databases. . . . 256
Obtaining information about application programs . . . 259
Obtaining information about and handling pages in error . . . 261
Making objects unavailable . . . 264
Altering buffer pools . . . 266
Monitoring buffer pools . . . 266
Controlling user-defined functions . . . 268
Starting user-defined functions . . . 269
Monitoring user-defined functions . . . 269
Stopping user-defined functions . . . 270

Controlling DB2 utilities . . . 271
Starting online utilities . . . 271
Monitoring and changing online utilities . . . 272
Controlling DB2 stand-alone utilities . . . 272
Controlling the IRLM . . . 274
z/OS commands that operate on IRLM . . . 275
Starting the IRLM . . . 276
Stopping the IRLM . . . 276
Monitoring threads . . . 277
Types of threads . . . 278
Output of the DISPLAY THREAD command . . . 279
Displaying information about threads . . . 279
Monitoring all DBMSs in a transaction . . . 284
Controlling connections . . . 287
Controlling TSO connections . . . 287
Controlling CICS connections . . . 290
Controlling IMS connections . . . 296
Controlling RRS connections . . . 307
Controlling connections to remote systems . . . 312
Controlling traces . . . 333
Diagnostic traces for attachment facilities . . . 334
Controlling the DB2 trace . . . 334
Diagnostic trace for the IRLM . . . 335
Setting the priority of stored procedures . . . 336

Chapter 9. Managing the log and the bootstrap data set . . . 337
How database changes are made . . . 337
Units of recovery and points of consistency . . . 337
How DB2 rolls back work . . . 338
How the initial DB2 logging environment is established . . . 339
How DB2 creates log records . . . 339
How DB2 writes the active log . . . 339
How DB2 writes (offloads) the archive log . . . 340
How DB2 retrieves log records . . . 346
Managing the log . . . 346
Quiescing activity before offloading . . . 346
Archiving the log . . . 348
Adding an active log data set to the active log inventory with the SET LOG command . . . 349
Dynamically changing the checkpoint frequency . . . 350
Setting limits for archive log tape units . . . 351
Monitoring the system checkpoint . . . 351
Displaying log information . . . 351
Resetting the log RBA . . . 352
Log RBA range . . . 353
Resetting the log RBA value in a data sharing environment . . . 353
Resetting the log RBA value in a non-data sharing environment . . . 354
Canceling and restarting an offload . . . 356
Displaying the status of an offload . . . 356
Discarding archive log records . . . 357
Locating archive log data sets . . . 357
Management of the bootstrap data set . . . 360
Restoring dual-BSDS mode . . . 361
BSDS copies with archive log data sets . . . 361
Recommendations for changing the BSDS log inventory . . . 362

Chapter 10. Restarting DB2 after termination . . . 363
Methods of restarting . . . 363
Types of termination . . . 363
Normal restart and recovery . . . 364
Automatic restart . . . 369

Restart in a data sharing environment . . . 370
Restart implications for table spaces that are not logged . . . 370
Conditional restart . . . 371
Terminating DB2 normally . . . 371
Restarting automatically . . . 372
Deferring restart processing . . . 372
Deferral of restart . . . 373
Performing conditional restart . . . 374
Conditional restart with system-level backups . . . 374
Options for recovery operations after conditional restart . . . 375
Conditional restart records . . . 375
Resolving postponed units of recovery . . . 375
RECOVER POSTPONED command . . . 376
Recovering from an error during RECOVER POSTPONED processing . . . 377

Chapter 11. Maintaining consistency across multiple systems . . . 379
Multiple system consistency . . . 379
Two-phase commit process . . . 379
Commit coordinator and multiple participants . . . 381
Illustration of multi-site update . . . 382
Termination for multiple systems . . . 383
Consistency after termination or failure . . . 384
Normal restart and recovery for multiple systems . . . 385
Multiple-system restart with conditions . . . 386
Heuristic decisions about whether to commit or abort an indoubt thread . . . 386
Resolving indoubt units of recovery . . . 386
Resolution of IMS indoubt units of recovery . . . 387
Resolution of CICS indoubt units of recovery . . . 388
Resolution of RRS indoubt units of recovery . . . 388
Resolving WebSphere Application Server indoubt units of recovery . . . 389
Resolving remote DBMS indoubt units of recovery . . . 392
Determining the coordinator's commit or abort decision . . . 392
Recovering indoubt threads . . . 393
Resetting the status of an indoubt thread . . . 393
Resolving an indoubt unit of recovery during DB2 restart . . . 394

Chapter 12. Backing up and recovering your data. . . . 395
Plans for recovery of distributed data . . . 396
Plans for extended recovery facility toleration . . . 396
Plans for recovery of indexes . . . 397
Actions to take when you back up data . . . 397
Actions to avoid when you back up data . . . 398
Preparation for recovery: a scenario . . . 399
Events that occur during recovery . . . 400
Complete recovery cycles . . . 402
A recovery cycle example when using image copies . . . 403
How DFSMShsm affects your recovery environment . . . 404
Tips for maximizing data availability during backup and recovery . . . 405
Where to find recovery information . . . 408
How to report recovery information . . . 409
Discarding SYSCOPY and SYSLGRNX records . . . 410
Preparations for disaster recovery . . . 411
System-wide points of consistency . . . 413
Recommendations for more effective recovery from inconsistency . . . 413
Actions to take to aid in successful recovery of inconsistent data . . . 413
Actions to avoid in recovery of inconsistent data . . . 415
How to recover multiple objects in parallel . . . 416
Recovery of page sets and data sets . . . 417
Recovery of the work file database . . . 418
Page set and data set copies . . . 418

System-level backups for object-level recoveries . . . 422
Recovery of data to a prior point in time . . . 424
Plans for point-in-time recovery . . . 424
Point-in-time recovery with system-level backups . . . 425
Point-in-time recovery using the RECOVER utility . . . 427
Implications of moving data sets after a system-level backup . . . 435
Recovery of table spaces . . . 436
Recovery of indexes . . . 439
| Recovery of FlashCopy image copies . . . 440
Preparing to recover to a prior point of consistency . . . 441
Preparing to recover an entire DB2 subsystem to a prior point in time using image copies or object-level backups 443
Creating essential disaster recovery elements . . . 444
Resolving problems with a user-defined work file data set . . . 445
Resolving problems with DB2-managed work file data sets . . . 446
Recovering error ranges for a work file table space . . . 446
Recovery of error ranges for a work file table space . . . 447
Recovering after a conditional restart of DB2 . . . 447
Recovery of the catalog and directory . . . 447
Regenerating missing identity column values . . . 448
Recovery of tables that contain identity columns . . . 449
Recovering a table space and all of its indexes . . . 450
Recovery implications for objects that are not logged . . . 450
Removing various pending states from LOB and XML table spaces . . . 454
Restoring data by using DSN1COPY . . . 454
Backing up and restoring data with non-DB2 dump and restore . . . 455
Recovering accidentally dropped objects . . . 455
How to avoid accidentally dropping objects . . . 455
Recovering an accidentally dropped table . . . 456
Recovering an accidentally dropped table space . . . 458
Recovering a DB2 system to a given point in time using the RESTORE SYSTEM utility . . . 462
Recovering by using DB2 restart recovery . . . 463
Recovering by using FlashCopy volume backups . . . 464
Making catalog definitions consistent with your data after recovery to a prior point in time . . . 464
Recovery of catalog and directory tables . . . 466
Performing remote site recovery from a disaster at a local site . . . 467
Recovering with the BACKUP SYSTEM and RESTORE SYSTEM utilities . . . 467
Recovering without using the BACKUP SYSTEM utility . . . 468
Backup and recovery involving clone tables . . . 468
| Recovery of temporal tables with system-period data versioning . . . 469
Data restore of an entire system . . . 469

Chapter 13. Recovering from different DB2 for z/OS problems . . . 471
Recovering from IRLM failure . . . 471
Recovering from z/OS or power failure . . . 471
Recovering from disk failure . . . 472
Recovering from application errors . . . 474
Backing out incorrect application changes (with a quiesce point) . . . 474
Backing out incorrect application changes (without a quiesce point) . . . 475
Recovering from IMS-related failures . . . 475
Recovering from IMS control region failure . . . 476
Recovering from IMS indoubt units of recovery . . . 476
Recovering from IMS application failure . . . 478
Recovering from a DB2 failure in an IMS environment . . . 479
Recovering from CICS-related failure . . . 479
Recovering from CICS application failures . . . 480
Recovering DB2 when CICS is not operational . . . 480
Recovering DB2 when the CICS attachment facility cannot connect to DB2 . . . 481
Recovering CICS indoubt units of recovery . . . 482
Recovering from CICS attachment facility failure . . . 485
Recovering from a QMF query failure . . . 485
Recovering from subsystem termination . . . 486

Recovering from temporary resource failure . . . 487
Recovering from active log failures . . . 488
Recovering from being out of space in active logs . . . 488
Recovering from a write I/O error on an active log data set . . . 489
Recovering from a loss of dual active logging . . . 490
Recovering from I/O errors while reading the active log . . . 490
Recovering from archive log failures. . . . 492
Recovering from allocation problems with the archive log . . . 492
Recovering from write I/O errors during archive log offload . . . 493
Recovering from read I/O errors on an archive data set during recovery . . . 493
Recovering from insufficient disk space for offload processing . . . 494
Recovering from BSDS failures . . . 495
Recovering from an I/O error on the BSDS . . . 495
Recovering from an error that occurs while opening the BSDS . . . 496
Recovering from unequal timestamps on BSDSs . . . 496
Recovering the BSDS from a backup copy . . . 497
Recovering from BSDS or log failures during restart . . . 500
Recovering from failure during log initialization or current status rebuild . . . 502
Recovering from a failure during forward log recovery . . . 514
Recovering from a failure during backward log recovery . . . 519
Recovering from a failure during a log RBA read request . . . 522
Recovering from unresolvable BSDS or log data set problem during restart . . . 523
Recovering from a failure resulting from total or excessive loss of log data . . . 525
Resolving inconsistencies resulting from a conditional restart . . . 529
Recovering from DB2 database failure . . . 535
Recovering a DB2 subsystem to a prior point in time . . . 536
Recovering from a down-level page set problem. . . . 537
Recovering from a problem with invalid LOBs . . . 539
Recovering from table space I/O errors. . . . 540
Recovering from DB2 catalog or directory I/O errors . . . 541
Recovering from integrated catalog facility failure . . . 542
Recovering VSAM volume data sets that are out of space or destroyed . . . 542
Recovering from out-of-disk-space or extent limit problems . . . 543
Recovering from referential constraint violation . . . 547
Recovering from distributed data facility failure . . . 548
Recovering from conversation failure . . . 548
Recovering from communications database failure . . . 549
Recovering from database access thread failure . . . 550
Recovering from VTAM failure . . . 551
Recovering from TCP/IP failure . . . 551
Recovering from remote logical unit failure . . . 551
Recovering from an indefinite wait condition . . . 552
Recovering database access threads after security failure . . . 553
Performing remote-site disaster recovery . . . 553
Recovering from a disaster by using system-level backups . . . 554
Restoring data from image copies and archive logs . . . 554
Recovering from disasters by using a tracker site . . . 569
Using data mirroring for disaster recovery . . . 578
Scenarios for resolving problems with indoubt threads . . . 584
Scenario: Recovering from communication failure . . . 586
Scenario: Making a heuristic decision about whether to commit or abort an indoubt thread . . . 588
Scenario: Recovering from an IMS outage that results in an IMS cold start . . . 590
Scenario: Recovering from a DB2 outage at a requester that results in a DB2 cold start . . . 591
Scenario: What happens when the wrong DB2 subsystem is cold started . . . 595
Scenario: Correcting damage from an incorrect heuristic decision about an indoubt thread . . . 597

Chapter 14. Reading log records . . . 599
Contents of the log . . . 599
Unit of recovery log records . . . 600
Checkpoint log records . . . 604
Database page set control records . . . 605

Other exception information . . . 605
The physical structure of the log . . . 605
Physical and logical log records . . . 606
The log record header . . . 607
The log control interval definition (LCID) . . . 608
Log record type codes . . . 610
Log record subtype codes . . . 611
Interpreting data change log records. . . . 612
Reading log records with IFI . . . 613
Gathering active log records into a buffer . . . 613
Reading specific log records (IFCID 0129) . . . 614
Reading complete log data (IFCID 0306) . . . 615
Reading log records with OPEN, GET, and CLOSE . . . 619
JCL DD statements for DB2 stand-alone log services . . . 620
Data sharing members that participate in a read. . . . 622
Registers and return codes . . . 623
Stand-alone log OPEN request. . . . 623
Stand-alone log GET request . . . 625
Stand-alone log CLOSE request . . . 627
Sample application that uses stand-alone log services . . . 627
Reading log records with the log capture exit routine . . . 629

Part 3. Appendixes . . . 631

Appendix A. Exit routines . . . 633
Edit procedures . . . 633
Specifying edit procedures . . . 634
When edit routines are taken . . . 634
Parameter list for edit procedures . . . 635
Incomplete rows and edit routines . . . 635
Expected output for edit routines. . . . 636
Validation routines . . . 637
Specifying validation routines . . . 638
When validation routines are taken . . . 638
Parameter list for validation routines . . . 639
Incomplete rows and validation routines . . . 640
Expected output for validation routines . . . 640
Date and time routines . . . 641
Specifying date and time routines . . . 641
When date and time routines are taken. . . . 642
Parameter list for date and time routines . . . 643
Expected output for date and time routines . . . 644
Conversion procedures . . . 645
Specifying conversion procedures . . . 645
When conversion procedures are taken . . . 646
Parameter list for conversion procedures . . . 646
Expected output for conversion procedures . . . 647
Field procedures . . . 648
Field-definition for field procedures . . . 649
Specifying field procedures. . . . 649
When field procedures are taken . . . 650
Control blocks for execution of field procedures . . . 651
Field-definition (function code 8) . . . 655
Field-encoding (function code 0) . . . 657
Field-decoding (function code 4) . . . 659
Log capture routines . . . 661
Specifying log capture routines . . . 662
When log capture routines are invoked. . . . 662
Parameter list for log capture routines . . . 663
Routines for dynamic plan selection in CICS . . . 664

General guidelines for writing exit routines . . . 664
Coding rules for exit routines . . . 665
Modifying exit routines . . . 666
Execution environment for exit routines . . . 666
Registers at invocation for exit routines. . . . 666
Parameter list for exit routines. . . . 667
Row formats for edit and validation routines . . . 668
Column boundaries for edit and validation procedures . . . 668
Null values for edit procedures, field procedures, and validation routines . . . 669
Fixed-length rows for edit and validation routines . . . 669
Varying-length rows for edit and validation routines . . . 669
Varying-length rows with nulls for edit and validation routines . . . 670
EDITPROCs and VALIDPROCs for handling basic and reordered row formats . . . 671
Converting basic row format table spaces with edit and validation routines to reordered row format . . . 671
Dates, times, and timestamps for edit and validation routines . . . 673
Parameter list for row format descriptions . . . 674
DB2 codes for numeric data in edit and validation routines . . . 675

Appendix B. Stored procedures for administration . . . 677
Common SQL API stored procedures . . . 677
Versioning of XML documents. . . . 678
XML input documents . . . 679
XML output documents . . . 680
XML message documents . . . 681
Troubleshooting DB2 stored procedure problems . . . 682

Information resources for DB2 for z/OS and related products . . . 685

Notices . . . 687
Programming interface information . . . 688
Trademarks . . . 689
Terms and conditions for product documentation . . . 690
Privacy policy considerations . . . 690

Glossary . . . 693

Index . . . 695

About this information

This information provides guidance information that you can use to perform a variety of administrative tasks with DB2® for z/OS® (DB2).

Information about DB2 concepts, which was present in previous editions of this book, is now provided in Introduction to DB2 for z/OS. In addition, information about DB2 security and auditing is now provided in Managing DB2 Security.

This information assumes that your DB2 subsystem is running in DB2 10 new-function mode.

Availability of new function in DB2 10

Generally, new SQL capabilities, including changes to existing functions, statements, and limits, become available only in new-function mode, unless explicitly stated otherwise. Exceptions to this general statement include optimization and virtual storage enhancements, which are also available in conversion mode unless stated otherwise. In Version 8 andDB2 9, most utility functions were available in conversion mode. However, for DB2 10, most utility functions become available in new-function mode.

Who should read this information

This information 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.

DB2 Utilities Suite

Important: In this version of DB2 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.

The DB2 Utilities Suite can work with DB2 Sort and the DFSORT program. You are licensed to use DFSORT 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

© Copyright IBM Corp. 1982, 2016

These informational APARs are periodically updated.

Related information

DB2 utilities packaging (Utility Guide)

Terminology and citations

When referring to a DB2 product other than DB2 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.

Tivoli® OMEGAMON® XE
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.

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.

Accessibility features for DB2 10 for z/OS

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use information technology products successfully.

Accessibility features

The following list includes the major accessibility features in z/OS products, including DB2 10 for z/OS. These features support:
- Keyboard-only operation.
- Interfaces that are commonly used by screen readers and screen magnifiers.
- Customization of display attributes such as color, contrast, and font size

Tip: The Information Management Software for z/OS Solutions Information Center (which includes information for DB2 10 for z/OS) and its related publications are accessibility-enabled for the IBM Home Page Reader. You can operate all features using the keyboard instead of the mouse.

Keyboard navigation

For information about navigating the DB2 10 for z/OS ISPF panels using TSO/E or ISPF, refer to the *z/OS TSO/E Primer*, the *z/OS TSO/E User's Guide*, and the *z/OS ISPF User's Guide*. These guides describe how to navigate each interface, including

the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

Related accessibility information

IBM and accessibility

See the *IBM Accessibility Center* at http://www.ibm.com/able for more information about the commitment that IBM has to accessibility.

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 for z/OS documentation.

Send your comments by email 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).

How to read syntax diagrams

Certain conventions apply to the syntax diagrams that are used in IBM documentation.

Apply the following rules when reading the syntax diagrams that are used in DB2 for z/OS documentation:

- 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.

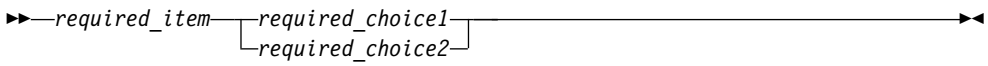
  The —►◄ symbol indicates the end of a statement.
- Required items appear on the horizontal line (the main path).

- Optional items appear below the main path.

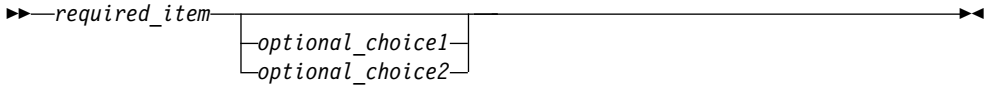
  If an optional item appears above the main path, that item has no effect on the execution of the statement and is used only for readability.

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

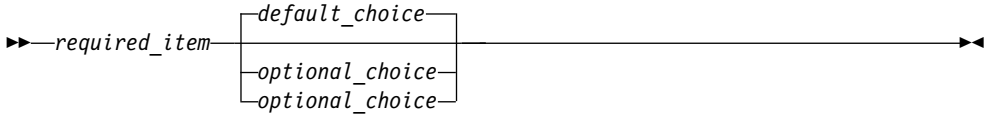
  If you *must* choose one of the items, one item of the stack appears on the main path.



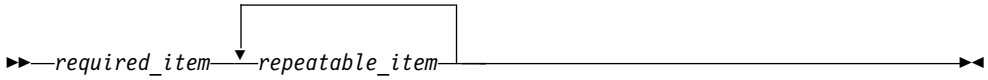
If choosing one of the items is optional, the entire stack appears below the main path.



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



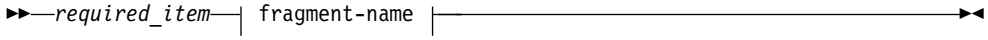
- An arrow returning to the left, above the main line, indicates an item that can be repeated.



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

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

- Sometimes a diagram must be split into fragments. The syntax fragment is shown separately from the main syntax diagram, but the contents of the fragment should be read as if they are on the main path of the diagram.



fragment-name:

- With the exception of XPath keywords, keywords appear in uppercase (for example, `FROM`). Keywords must be spelled exactly as shown. XPath keywords are defined as lowercase names, and 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.

Part 1. Designing a database

© Copyright IBM Corp. 1982, 2016

Chapter 1. Database objects and relationships

The general tasks that are necessary to design a database are logical data modeling and physical data modeling.

In logical data modeling, you design a model of the data without paying attention to specific functions and capabilities of the DBMS that will store the data. In fact, you could even build a logical data model without knowing which DBMS you will use.

Physical data modeling begins when you move closer to a physical implementation. The primary purpose of the physical design stage is to optimize performance while ensuring the integrity of the data.

Logical database design with the entity-relationship model

Before you implement a database, you should plan or design the database so that it satisfies all requirements.

Designing and implementing a successful database, one that satisfies the needs of an organization, requires a logical data model. Logical data modeling is the process of documenting the comprehensive business information requirements in an accurate and consistent format. Analysts who do data modeling define the data items and the business rules that affect those data items. The process of data modeling acknowledges that business data is a vital asset that the organization needs to understand and carefully manage. This section contains information that was adapted from Handbook of Relational Database Design.

Consider the following business facts that a manufacturing company needs to represent in its data model:

- Customers purchase products
- Products consist of parts
- Suppliers manufacture parts
- Warehouses store parts
- Transportation vehicles move the parts from suppliers to warehouses and then to manufacturers

These are all business facts that a manufacturing company's logical data model needs to include. Many people inside and outside the company rely on information that is based on these facts. Many reports include data about these facts.

Any business, not just manufacturing companies, can benefit from the task of data modeling. Database systems that supply information to decision makers, customers, suppliers, and others are more successful if their foundation is a sound data model.

Modeling your data

Data analysts can perform the task of data modeling in a variety of ways.

© Copyright IBM Corp. 1982, 2016

Procedure

To model data:

1. Build critical user views.
   a. Carefully examining a single business activity or function.
   b. Develop a user view, which is the model or representation of critical information that the business activity requires.

      This initial stage of the data modeling process is highly interactive. Because data analysts cannot fully understand all areas of the business that they are modeling, they work closely with the actual users. Working together, analysts and users define the major entities (significant objects of interest) and determine the general relationships between these entities.

      In a later stage, the analyst combines each individual user view with all the other user views into a consolidated logical data model.
2. Add keys to user views

   Key business rules affect insert, update, and delete operations on the data. For example, a business rule might require that each customer entity have at least one unique identifier. Any attempt to insert or update a customer identifier that matches another customer identifier is not valid. In a data model, a unique identifier is called a primary key.
3. Add detail to user views and validate them.
   a. Add other descriptive details that are less vital.
   b. Associate these descriptive details, called attributes, to the entities.

      For example, a customer entity probably has an associated phone number. The phone number is a non-key attribute of the customer entity.
   c. Validate all the user views

      To validate the views, analysts use the normalization process and process models. Process models document the details of how the business will use the data.
4. Determine additional business rules that affect attributes.
   a. Clarify the data-driven business rules.

      Data-driven business rules are constraints on particular data values. These constraints need to be true, regardless of any particular processing requirements.

      The advantage to defining data-driven business rules during the data design stage, rather than during application design is that programmers of many applications don't need to write code to enforce these business rules. For example, Assume that a business rule requires that a customer entity have a phone number, an address, or both. If this rule doesn't apply to the data itself, programmers must develop, test, and maintain applications that verify the existence of one of these attributes. Data-driven business requirements have a direct relationship with the data, thereby relieving programmers from extra work.
5. Integrate user views.
   a. Combine into a consolidated logical data model the newly created different user views.
   b. Integrate other data models that already exist in the organization with the new consolidated logical data model.

   At this stage, analysts also strive to make their data model flexible so that it can support the current business environment and possible future changes. For example, assume that a retail company operates in a single country and that

business plans include expansion to other countries. Armed with knowledge of these plans, analysts can build the model so that it is flexible enough to support expansion into other countries.

Recommendations for logical data modeling

To build sound data models, analysts follow a well-planned methodology.

Follow these recommendation for building quality data models:

- Work interactively with the users as much as possible.
- Use diagrams to represent as much of the logical data model as possible.
- Build a data dictionary to supplement the logical data model diagrams.

  A *data dictionary* is 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.

Practical examples of data modeling

To better understand the key activities that are necessary for creating valid data models, investigate one or more real-life data modeling scenarios.

You begin by defining your entities, the significant objects of interest. Entities are the things about which you want to store information. For example, you might want to define an entity, called EMPLOYEE, for employees because you need to store information about everyone who works for your organization. You might also define an entity, called DEPARTMENT, for departments.

Next, you define primary keys for your entities. A primary key is a unique identifier for an entity. In the case of the EMPLOYEE entity, you probably need to store a large amount of information. However, most of this information (such as gender, birth date, address, and hire date) would not be a good choice for the primary key. In this case, you could choose a unique employee ID or number (EMPLOYEE_NUMBER) as the primary key. In the case of the DEPARTMENT entity, you could use a unique department number (DEPARTMENT_NUMBER) as the primary key.

After you have decided on the entities and their primary keys, you can define the relationships that exist between the entities. The relationships are based on the primary keys. If you have an entity for EMPLOYEE and another entity for DEPARTMENT, the relationship that exists is that employees are assigned to departments. You can read more about this topic in the next section.

After defining the entities, their primary keys, and their relationships, you can define additional attributes for the entities. In the case of the EMPLOYEE entity, you might define the following additional attributes:

- Birth date
- Hire date
- Home address
- Office phone number
- Gender
- Resume

Lastly, you normalize the data.

Related concepts:

"Entity normalization" on page 9

Entities for different types of relationships

In a relational database, you can express several types of relationships.

Consider the possible relationships between employees and departments. If a given employee can work in only one department, this relationship is one-to-one for employees. One department usually has many employees; this relationship is one-to-many for departments. Relationships can be one-to-many, many-to-one, one-to-one, or many-to- many.

Subsections:

- "One-to-one relationships"
- "One-to-many and many-to-one relationships"
- "Many-to-many relationships" on page 7
- "Business rules for relationships" on page 7

The type of a given relationship can vary, depending on the specific environment. If employees of a company belong to several departments, the relationship between employees and departments is many-to-many.

You need to define separate entities for different types of relationships. When modeling relationships, you can use diagram conventions to depict relationships by using different styles of lines to connect the entities.

One-to-one relationships

When you are doing logical database design, one-to-one relationships are bidirectional relationships, which means that they are single-valued in both directions. For example, an employee has a single resume; each resume belongs to only one person. The previous figure illustrates that a one-to-one relationship exists between the two entities. In this case, the relationship reflects the rules that an employee can have only one resume and that a resume can belong to only one employee.

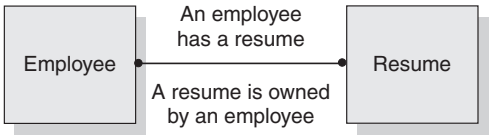


Figure 1. Assigning one-to-one facts to an entity

One-to-many and many-to-one relationships

A one-to-many relationship occurs when one entity has a multivalued relationship with another entity. In the following figure, you see that a one-to-many relationship exists between the two entities—employee and department. This figure reinforces the business rules that a department can have many employees, but that each individual employee can work for only one department.

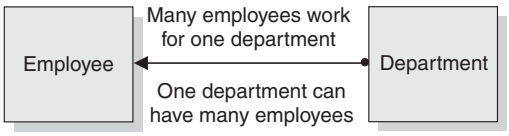


Figure 2. Assigning many-to-one facts to an entity

Many-to-many relationships

A many-to-many relationship is a relationship that is multivalued in both directions. The following figure illustrates this kind of relationship. An employee can work on more than one project, and a project can have more than one employee assigned.

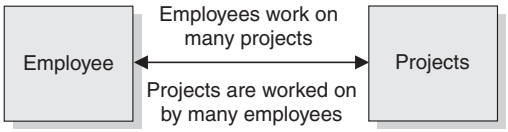


Figure 3. Assigning many-to-many facts to an entity

Business rules for relationships

Whether a given relationship is one-to-one, one-to-many, many-to-one, or many-to-many, your relationships need to make good business sense. Therefore, database designers and data analysts can be more effective when they have a good understanding of the business. If they understand the data, the applications, and the business rules, they can succeed in building a sound database design.

When you define relationships, you have a big influence on how smoothly your business runs. If you don't do a good job at this task, your database and associated applications are likely to have many problems, some of which may not manifest themselves for years.

Entity attributes

When you define attributes for the entities, you generally work with the data administrator to decide on names, data types, and appropriate values for the attributes.

Attribute names

Most organizations have naming guidelines. In addition to following these guidelines, data analysts also base attribute definitions on class words.

A *class word* is a single word that indicates the nature of the data that the attribute represents.

The class word NUMBER indicates an attribute that identifies the number of an entity. Therefore, attribute names that identify the numbers of entities should include the class word of NUMBER. Some examples are EMPLOYEE_NUMBER, PROJECT_NUMBER, and DEPARTMENT_NUMBER.

When an organization does not have well-defined guidelines for attribute names, data analysts try to determine how the database designers have historically named

attributes. Problems occur when multiple individuals are inventing their own naming guidelines without consulting one another.

Data types of attributes

You must specify a data type for each attribute of an entity. Most organizations have well-defined guidelines for using the different data types.

You might use the following data types for attributes of the EMPLOYEE entity:

- EMPLOYEE_NUMBER: CHAR(6)
- EMPLOYEE_LAST_NAME: VARCHAR(15)
- EMPLOYEE_HIRE_DATE: DATE
- EMPLOYEE_SALARY_AMOUNT: DECIMAL(9,2)

The data types that you choose are business definitions of the data type. During physical database design, you might need to change data type definitions or use a subset of these data types. The database or the host language might not support all of these definitions, or you might make a different choice for performance reasons.

For example, you might need to represent monetary amounts, but DB2 and many host languages do not have a data type MONEY. In the United States, a natural choice for the SQL data type in this situation is DECIMAL(10,2) to represent dollars. But you might also consider the INTEGER data type for fast, efficient performance.

Related reference:

CREATE TABLE (DB2 SQL)

SQL data type attributes (DB2 Programming for ODBC)

Appropriate values for attributes

When you design a database, you need to decide what values are acceptable for the various attributes of an entity.

For example, you would not want to allow numeric data in an attribute for a person's name. The data types that you choose limit the values that apply to a given attribute, but you can also use other mechanisms. These other mechanisms are domains, null values, and default values.

Subsections:

- "Domain"
- "Null values" on page 9
- "Default values" on page 9

Domain

A *domain* describes the conditions that an attribute value must meet to be a valid value. Sometimes the domain identifies a range of valid values. By defining the domain for a particular attribute, you apply business rules to ensure that the data will make sense.

Example 1: A domain might state that a phone number attribute must be a 10-digit value that contains only numbers. You would not want the phone number to be incomplete, nor would you want it to contain alphabetic or special characters and thereby be invalid. You could choose to use either a numeric data type or a

character data type. However, the domain states the business rule that the value must be a 10-digit value that consists of numbers.

Example 2: A domain might state that a month attribute must be a 2-digit value from 01 to 12. Again, you could choose to use datetime, character, or numeric data types for this value, but the domain demands that the value must be in the range of 01 through 12. In this case, incorporating the month into a datetime data type is probably the best choice. This decision should be reviewed again during physical database design.

Null values

When you are designing attributes for your entities, you will sometimes find that an attribute does not have a value for every instance of the entity. For example, you might want an attribute for a person's middle name, but you can't require a value because some people have no middle name. For these occasions, you can define the attribute so that it can contain null values.

A *null value* is a special indicator that represents the absence of a value. The value can be absent because it is unknown, not yet supplied, or nonexistent. The DBMS treats the null value as an actual value, not as a zero value, a blank, or an empty string.

Just as some attributes should be allowed to contain null values, other attributes should not contain null values.

Example: For the EMPLOYEE entity, you might not want to allow the attribute EMPLOYEE_LAST_NAME to contain a null value.

Default values

In some cases, you may not want a given attribute to contain a null value, but you don't want to require that the user or program always provide a value. In this case, a default value might be appropriate.

A *default value* is a value that applies to an attribute if no other valid value is available.

Example: Assume that you don't want the EMPLOYEE_HIRE_DATE attribute to contain null values and that you don't want to require users to provide this data. If data about new employees is generally added to the database on the employee's first day of employment, you could define a default value of the current date.

Entity normalization

After you define entities and decide on attributes for the entities, you normalize entities to avoid redundancy.

An entity is normalized if it meets a set of constraints for a particular normal form, which this section describes. Normalization helps you avoid redundancies and inconsistencies in your data. This section summarizes rules for first, second, third, and fourth normal forms of entities, and it describes reasons why you should or shouldn't follow these rules.

Subsections:

- "First normal form" on page 10

- "Second normal form"
- "Third normal form" on page 11
- "Fourth normal form" on page 13

The rules for normal form are cumulative. In other words, for an entity to satisfy the rules of second normal form, it also must satisfy the rules of first normal form. An entity that satisfies the rules of fourth normal form also satisfies the rules of first, second, and third normal form.

In this section, you will see many references to the word *instance*. In the context of logical data modeling, an instance is one particular occurrence. An instance of an entity is a set of data values for all of the attributes that correspond to that entity.

Example: The following figure shows one instance of the EMPLOYEE entity.

Employee

EMPLOYEE _NUMBER	EMPLOYEE _FIRST _NAME	EMPLOYEE _LAST _NAME	DEPARTMENT _NUMBER	EMPLOYEE _HIRE_DATE
000010	CHRISTINE	HAAS	A00	1975-01-01

Figure 4. The EMPLOYEE entity

First normal form

A relational entity satisfies the requirement of first normal form if every instance of an entity contains only one value, never multiple repeating attributes. Repeating attributes, often called a repeating group, are different attributes that are inherently the same. In an entity that satisfies the requirement of first normal form, each attribute is independent and unique in its meaning and its name.

Example: Assume that an entity contains the following attributes:

```
EMPLOYEE_NUMBER
JANUARY_SALARY_AMOUNT
FEBRUARY_SALARY_AMOUNT
MARCH_SALARY_AMOUNT
```

This situation violates the requirement of first normal form, because JANUARY_SALARY_AMOUNT, FEBRUARY_SALARY_AMOUNT, and MARCH_SALARY_AMOUNT are essentially the same attribute, EMPLOYEE_ MONTHLY_SALARY_AMOUNT.

Second normal form

An entity is in second normal form if each attribute that is not in the primary key provides a fact that depends on the entire key. A violation of the second normal form occurs when a nonprimary key attribute is a fact about a subset of a composite key.

Example: An inventory entity records quantities of specific parts that are stored at particular warehouses. The following figure shows the attributes of the inventory entity.

Figure 5. Entity in violation of the second normal form

Here, the primary key consists of the PART and the WAREHOUSE attributes together. Because the attribute WAREHOUSE_ADDRESS depends only on the value of WAREHOUSE, the entity violates the rule for second normal form. This design causes several problems:

- Each instance for a part that this warehouse stores repeats the address of the warehouse.
- If the address of the warehouse changes, every instance referring to a part that is stored in that warehouse must be updated.
- Because of the redundancy, the data might become inconsistent. Different instances could show different addresses for the same warehouse.
- If at any time the warehouse has no stored parts, the address of the warehouse might not exist in any instances in the entity.

To satisfy second normal form, the information in the previous figure would be in two entities, as the following figure shows.

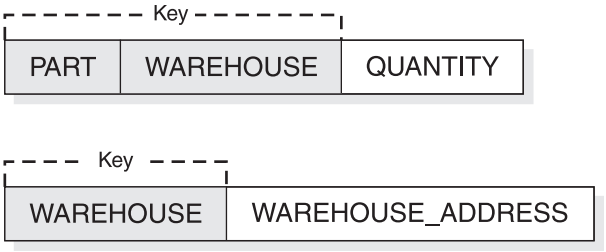


Figure 6. Entities that satisfy the second normal form

Third normal form

An entity is in third normal form if each nonprimary key attribute provides a fact that is independent of other non-key attributes and depends only on the key. A violation of the third normal form occurs when a nonprimary attribute is a fact about another non-key attribute.

Employee_Department table before updating

Key

EMPLOYEE _NUMBER	EMPLOYEE _FIRST _NAME	EMPLOYEE _LAST _NAME	DEPARTMENT _NUMBER	DEPARTMENT _NAME
000200	DAVID	BROWN	D11	MANUFACTURING
000320	RAMAL	MEHTA	E21	SOFTWARE
000220	JENIFER	LUTZ	D11	MANUFACTURING

Employee_Department table after updating

Key

EMPLOYEE _NUMBER	EMPLOYEE _FIRST _NAME	EMPLOYEE _LAST _NAME	DEPARTMENT _NUMBER	DEPARTMENT _NAME
000200	DAVID	BROWN	D11	INSTALLATION
000320	RAMAL	MEHTA	E21	SOFTWARE
000220	JENIFER	LUTZ	D11	MANUFACTURING

Figure 7. Results of an update in a table that violates the third normal form

Example: The first entity in the previous figure contains the attributes EMPLOYEE_NUMBER and DEPARTMENT_NUMBER. Suppose that a program or user adds an attribute, DEPARTMENT_NAME, to the entity. The new attribute depends on DEPARTMENT_NUMBER, whereas the primary key is on the EMPLOYEE_NUMBER attribute. The entity now violates third normal form.

Changing the DEPARTMENT_NAME value based on the update of a single employee, David Brown, does not change the DEPARTMENT_NAME value for other employees in that department. The updated version of the entity as shown in the previous figure illustrates the resulting inconsistency. Additionally, updating the DEPARTMENT_ NAME in this table does not update it in any other table that might contain a DEPARTMENT_NAME column.

You can normalize the entity by modifying the EMPLOYEE_DEPARTMENT entity and creating two new entities: EMPLOYEE and DEPARTMENT. The following figure shows the new entities. The DEPARTMENT entity contains attributes for DEPARTMENT_NUMBER and DEPARTMENT_NAME. Now, an update such as changing a department name is much easier. You need to make the update only to the DEPARTMENT entity.

Employee table

Key

EMPLOYEE _NUMBER	EMPLOYEE _FIRST _NAME	EMPLOYEE _LAST _NAME
000200	DAVID	BROWN
000320	RAMAL	MEHTA
000220	JENIFER	LUTZ

Department table

Key

DEPARTMENT _NUMBER	DEPARTMENT _NAME
D11	MANUFACTURING
E21	SOFTWARE

Employee_Department table

Key

DEPARTMENT _NUMBER	EMPLOYEE _NUMBER
D11	000200
D11	000220
E21	000329

Figure 8. Employee and department entities that satisfy the third normal form

Fourth normal form

An entity is in fourth normal form if no instance contains two or more independent, multivalued facts about an entity.

Key

EMPID	SKILL_CODE	LANGUAGE_CODE	SKILL_PROFICIENCY	LANGUAGE_PROFICIENCY

Figure 9. Entity in violation of the fourth normal form

Example: Consider the EMPLOYEE entity. Each instance of EMPLOYEE could have both SKILL_CODE and LANGUAGE_CODE. An employee can have several skills and know several languages. Two relationships exist, one between employees and skills, and one between employees and languages. An entity is not in fourth normal form if it represents both relationships, as the previous figure shows.

Instead, you can avoid this violation by creating two entities that represent both relationships, as the following figure shows.

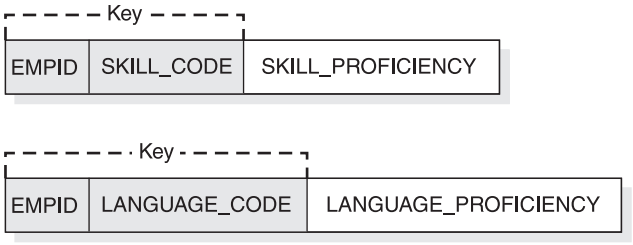


Figure 10. Entities that satisfy the fourth normal form

If, however, the facts are interdependent (that is, the employee applies certain languages only to certain skills), you should not split the entity.

You can put any data into fourth normal form. A good rule to follow when doing logical database design is to arrange all the data in entities that are in fourth normal form. Then decide whether the result gives you an acceptable level of performance. If the performance is not acceptable, denormalizing your design is a good approach to improving performance.

Related concepts:

"Practical examples of data modeling" on page 5

"Denormalization of tables" on page 16

Logical database design with Unified Modeling Language

You can use the Unified Modeling Language (UML) to create a model of your database design.

The Object Management Group is a consortium that created the UML standard. UML modeling is based on object-oriented programming principals. The basic difference between the entity-relationship model and the UML model is that, instead of designing entities, you model objects. UML defines a standard set of modeling diagrams for all stages of developing a software system. Conceptually, UML diagrams are like the blueprints for the design of a software development project.

Some examples of UML diagrams are as follows:

Class Identifies high-level entities, known as classes. A class describes a set of objects that have the same attributes. A class diagram shows the relationships between classes.

Use case
Presents a high-level view of a system from the user's perspective. A use case diagram defines the interactions between users and applications or between applications. These diagrams graphically depict system behavior. You can work with use-case diagrams to capture system requirements, learn how the system works, and specify system behavior.

Activity
Models the workflow of a business process, typically by defining rules for the sequence of activities in the process. For example, an accounting company can use activity diagrams to model financial transactions.

Interaction
Shows the required sequence of interactions between objects. Interaction diagrams can include sequence diagrams and collaboration diagrams.

- Sequence diagrams show object interactions in a time-based sequence that establishes the roles of objects and helps determine class responsibilities and interfaces.
- Collaboration diagrams show associations between objects that define the sequence of messages that implement an operation or a transaction.

Component
Shows the dependency relationships between components, such as main programs and subprograms.

Developers can graphically represent the architecture of a database and how it interacts with applications using one of many available UML modeling tools. Similarities exist between components of the entity-relationship model and UML diagrams. For example, the class structure corresponds closely to the entity structure.

The logical data model provides an overall view of the captured business requirements as they pertain to data entities. The data model diagram graphically represents the physical data model. The physical data model applies the logical data model's captured requirements to specific DBMS languages. Physical data models also capture the lower-level detail of a DBMS database.

Database designers can customize the data model diagram from other UML diagrams, which allows them to work with concepts and terminology, such as columns, tables, and relationships, with which they are already familiar. Developers can also transform a logical data model into a physical data model.

Because the data model diagram includes diagrams for modeling an entire system, it allows database designers, application developers, and other development team members to share and track business requirements throughout development. For example, database designers can capture information, such as constraints, triggers, and indexes, directly on the UML diagram. Developers can also transfer between object and data models and use basic transformation types such as many-to-many relationships.

Physical database design

After you complete the logical design of your database, you now move to the physical design. The purpose of building a physical design of your database is to optimize performance, while ensuring data integrity by avoiding unnecessary data redundancies.

During physical design, you transform the entities into tables, the instances into rows, and the attributes into columns. You and your colleagues must decide on many factors that affect the physical design, such as:

- How to translate entities into physical tables
- What attributes to use for columns of the physical tables
- Which columns of the tables to define as keys
- What indexes to define on the tables
- What views to define on the tables
- How to denormalize the tables
- How to resolve many-to-many relationships

Physical design is the time when you abbreviate the names that you chose during logical design. For example, you can abbreviate the column name that identifies employees, EMPLOYEE_NUMBER, to EMPNO. The column name size has a 30-byte maximum, and the table name size has a 128-byte maximum.

The task of building the physical design is a job that never ends. You need to continually monitor the performance and data integrity characteristics of a database as time passes. Many factors necessitate periodic refinements to the physical design.

DB2 lets you change many of the key attributes of your design with ALTER SQL statements. For example, assume that you design a partitioned table so that it will store 36 months of data. Later you discover that you need to extend that design to hold 84 months of data. You can add or rotate partitions for the current 36 months to accommodate the new design.

The remainder of this chapter includes some valuable information that can help you build and refine your database's physical design.

Denormalization of tables

During physical design, analysts transform the entities into tables and the attributes into columns.

Denormalization is a key step in the task of building a physical relational database design. It is the intentional duplication of columns in multiple tables, and the consequence is increased data redundancy.

The warehouse address column first appears as part of a table that contains information about parts and warehouses. To further normalize the design of the table, analysts remove the warehouse address column from that table. Analysts also define the column as part of a table that contains information only about warehouses.

Normalizing tables is generally the recommended approach. What if applications require information about both parts and warehouses, including the addresses of warehouses? The premise of the normalization rules is that SQL statements can retrieve the information by joining the two tables. The problem is that, in some cases, performance problems can occur as a result of normalization. For example, some user queries might view data that is in two or more related tables; the result is too many joins. As the number of tables increases, the access costs can increase, depending on the size of the tables, the available indexes, and so on. For example, if indexes are not available, the join of many large tables might take too much time. You might need to denormalize your tables. Denormalization is the intentional duplication of columns in multiple tables, and it increases data redundancy.

Example: Consider the design in which both tables have a column that contains the addresses of warehouses. If this design makes join operations unnecessary, it could be a worthwhile redundancy. Addresses of warehouses do not change often, and if one does change, you can use SQL to update all instances fairly easily.

Tip: Do not automatically assume that all joins take too much time. If you join normalized tables, you do not need to keep the same data values synchronized in

multiple tables. In many cases, joins are the most efficient access method, despite the overhead they require. For example, some applications achieve 44-way joins in subsecond response time.

When you are building your physical design, you and your colleagues need to decide whether to denormalize the data. Specifically, you need to decide whether to combine tables or parts of tables that are frequently accessed by joins that have high performance requirements. This is a complex decision about which this book cannot give specific advice. To make the decision, you need to assess the performance requirements, different methods of accessing the data, and the costs of denormalizing the data. You need to consider the trade-off: is duplication, in several tables, of often-requested columns less expensive than the time for performing joins?

Recommendations:

- Do not denormalize tables unless you have a good understanding of the data and the business transactions that access the data. Consult with application developers before denormalizing tables to improve the performance of users' queries.
- When you decide whether to denormalize a table, consider all programs that regularly access the table, both for reading and for updating. If programs frequently update a table, denormalizing the table affects performance of update programs because updates apply to multiple tables rather than to one table.

In the following figure, information about parts, warehouses, and warehouse addresses appears in two tables, both in normal form.

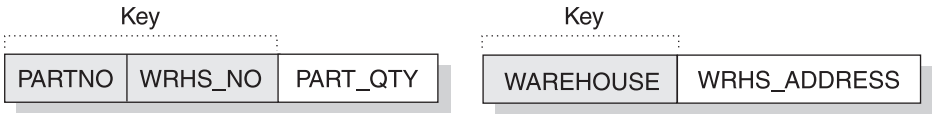


Figure 11. Two tables that satisfy second normal form

The following figure illustrates the denormalized table.

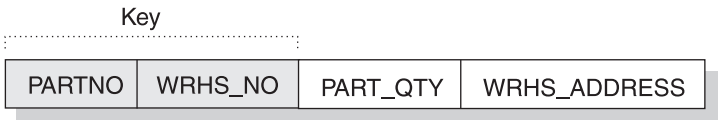


Figure 12. The denormalized table

Resolving many-to-many relationships is a particularly important activity because doing so helps maintain clarity and integrity in your physical database design. To resolve many-to-many relationships, you introduce associative tables, which are intermediate tables that you use to tie, or associate, two tables to each other.

Example: Employees work on many projects. Projects have many employees. In the logical database design, you show this relationship as a many-to-many relationship between project and employee. To resolve this relationship, you create a new associative table, EMPLOYEE_PROJECT. For each combination of employee and project, the EMPLOYEE_PROJECT table contains a corresponding row. The primary key for the table would consist of the employee number (EMPNO) and the project number (PROJNO).

Another decision that you must make relates to the use of repeating groups.

Example: Assume that a heavily used transaction requires the number of wires that are sold by month in a given year. Performance factors might justify changing a table so that it violates the rule of first normal form by storing repeating groups. In this case, the repeating group would be: MONTH, WIRE. The table would contain a row for the number of sold wires for each month (January wires, February wires, March wires, and so on).

Recommendation: If you decide to denormalize your data, document your denormalization thoroughly. Describe, in detail, the logic behind the denormalization and the steps that you took. Then, if your organization ever needs to normalize the data in the future, an accurate record is available for those who must do the work.

Related concepts:

"Entity normalization" on page 9

Database design with denormalization (Introduction to DB2 for z/OS)

Views to customize what data users see

A *view* offers an alternative way of describing data that exists in one or more tables.

Some users might find that no single table contains all the data they need; rather, the data might be scattered among several tables. Furthermore, one table might contain more data than users want to see, or more than you want to authorize them to see. For those situations, you can create views.

You might want to use views for a variety of reasons:

- To limit access to certain kinds of data

  You can create a view that contains only selected columns and rows from one or more tables. Users with the appropriate authorization on the view see only the information that you specify in the view definition.

  Example: You can define a view on the EMP table to show all columns except SALARY and COMM (commission). You can grant access to this view to people who are not managers because you probably don't want them to have access to salary and commission information.
- To combine data from multiple tables

  You can create a view that uses UNION or UNION ALL operators to logically combine smaller tables, and then query the view as if it were one large table.

  Example: Assume that three tables contain data for a period of one month. You can create a view that is the UNION ALL of three fullselects, one for each month of the first quarter of 2004. At the end of the third month, you can view comprehensive quarterly data.

You can create a view any time after the underlying tables exist. The owner of a set of tables implicitly has the authority to create a view on them. A user with administrative authority at the system or database level can create a view for any owner on any set of tables. If they have the necessary authority, other users can also create views on a table that they did not create.

Related concepts:

DB2 views (Introduction to DB2 for z/OS)

Indexes on table columns

If you are involved in the physical design of a database, you will be working with other designers to determine what columns you should index.

You will use process models that describe how different applications are going to be accessing the data. This information is important when you decide on indexing strategies to ensure adequate performance.

The main purposes of an index are:

- To optimize data access

  In many cases, access to data is faster with an index than without an index. If the DBMS uses an index to find a row in a table, the scan can be faster than when the DBMS scans an entire table.
- To ensure uniqueness

  A table with a unique index cannot have two rows with the same values in the column or columns that form the index key. For example, if payroll applications use employee numbers, no two employees can have the same employee number.

  Unique indexes can include additional columns that are not part of a unique constraint. Those columns are called INCLUDE columns. When you specify INCLUDE columns in a unique index, queries can use the unique index for index-only access. Including these columns can eliminate the need to maintain extra indexes that are used solely to enable index-only access.
- To enable clustering

  A clustering index keeps table rows in a specified sequence to minimize page access for a set of rows.

In general, users of the table are unaware that an index is in use. DB2 decides whether to use the index to access the table.

Related concepts:

Creation of indexes (Introduction to DB2 for z/OS)

Index access (ACCESSTYPE is 'I', 'IN', 'I1', 'N', 'MX', or 'DX') (DB2 Performance)

Related tasks:

Designing indexes for performance (DB2 Performance)

Related information:

"Implementing DB2 indexes" on page 72

Hash access on tables

You can use hash access to optimize data access for certain kinds of tables.

Introductory concepts:

DB2 hash spaces (Introduction to DB2 for z/OS)

If you are involved in the physical design of a database, you work with other designers to determine when to enable hash access on tables.

The main purposes of hash access is to optimize data access. If your programs regularly access a single row in a table and the table has a unique identifier for

each row, you can use hash access to directly retrieve the data from individual rows. Hash access requires that tables have at least one column with values that are unique to each row.

Related concepts:

Hash access (ACCESSTYPE='H', 'HN', or 'MH') (DB2 Performance)

Related tasks:

Organizing tables by hash for fast access to individual rows (DB2 Performance)

"Creating tables that use hash organization" on page 47
"Altering tables to enable hash access" on page 150

Monitoring hash access (DB2 Performance)

Chapter 2. Implementing your database design

Implementing your database design involves implementing DB2 objects, loading and managing data, and altering your design as necessary.

Tip:

GUPI

You can simplify your database implementation by letting DB2 implicitly create certain objects for you. On a CREATE TABLE statement, if you do not specify a database name, DB2 will use an existing implicitly created database. If an implicitly created database does not exist, DB2 creates one using the naming convention of DSNxxxxx. The DSNxxxxx values can range from DSN00001 to DSNnnnnn, where *nnnnn* is the maximum value of the sequence SYSIBM.DSNSEQ_IMPLICITDB, with a default of 10000. If the table space is implicitly created, DB2 will create all of the required system objects for you, including:

- The primary key enforcing index and the unique key index
- The ROWID index (if the ROWID column is defined as GENERATED BY DEFAULT)
- The LOB table spaces, the auxiliary tables, and the auxiliary indexes

GUPI

Related concepts:

Chapter 3, “Altering your database design,” on page 101

Related tasks:

Designing databases for performance (DB2 Performance)

Compressing your data (DB2 Performance)

Related reference:

CREATE TABLE (DB2 SQL)

Implementing DB2 databases

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

Use DB2 databases to collect and control data.

Related concepts:

DB2 databases (Introduction to DB2 for z/OS)

Creating DB2 databases

You can create a DB2 database by defining a database at the current server.

© Copyright IBM Corp. 1982, 2016

Procedure

To create a database:

Issue the CREATE DATABASE statement.

Related concepts:

DB2 databases (Introduction to DB2 for z/OS)

Related reference:

CREATE DATABASE (DB2 SQL)

Dropping DB2 databases

You can drop a DB2 database by removing the database at the current server. When you drop a database, all of its table spaces, tables, index spaces, and indexes are dropped, too.

Procedure

To drop a database:

Issue the DROP DATABASE statement.

Related concepts:

DB2 databases (Introduction to DB2 for z/OS)

Related reference:

DROP (DB2 SQL)

Implementing DB2 storage groups

A *storage group* is a set of storage objects on which DB2 for z/OS data can be stored. DB2 uses storage groups to allocate storage for table spaces and indexes, and to define, extend, alter, and delete VSAM data sets.

You have the following options for creating storage groups and managing DB2 data sets:

- You can let DB2 manage the data sets. This option means less work for DB2 database administrators.
- You can 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. For more information, see "Enabling SMS to control DB2 storage groups" on page 24.
- You can 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.

Related tasks:

"Altering DB2 storage groups" on page 114

Advantages of storage groups

Allowing DB2 to manage your data sets by using DB2 storage groups offers several advantages.

The following list describes 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 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.
- 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.
- 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 with the SHRLEVEL NONE option, 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).

  Restriction: 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.

Related concepts:

"Managing your own data sets" on page 35

Related information:

"Managing DB2 data sets with DFSMShsm" on page 31

Control interval sizing

A *control interval* is an area on disk where VSAM stores records and creates distributed free space. A control interval is a unit of information that VSAM transfers between virtual and auxiliary storage.

DB2 page sets are defined as VSAM linear data sets. DB2 can define data sets with variable VSAM control intervals. One of the biggest benefits of variable VSAM control intervals 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.

The following table shows the default and compatible control interval sizes for each table space page size. For example, a table space with pages that are 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 1. 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

You can create DB2 storage groups by using the CREATE STOGROUP statement. DB2 storage groups are a set of volumes on disks that hold the data sets in which tables and indexes are stored.

Procedure

GUPI To create a DB2 storage group:

1. Issue the SQL statement CREATE STOGROUP.
2. Specify the storage group name.

   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. GUPI

Results

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.

Enabling SMS to control DB2 storage groups

Managing data sets with the Storage Management Subsystem (SMS) family of products can reduce workload for database administrators and storage administrators.

Procedure

To enable SMS to control DB2 storage groups:

1. Issue a CREATE STOGROUP SQL statement to define a DB2 storage group. You can specify SMS classes when you create a storage group.
2. Indicate how you want SMS to control the allocation of volumes in one of the following ways:
   - Specify an asterisk (*) for the VOLUMES attribute.
   - Specify the DATACLAS, MGMTCLAS, or STORCLAS keywords.

What to do next

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.

Related tasks:

"Migrating to DFSMShsm" on page 31

Related reference:

CREATE STOGROUP (DB2 SQL)

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. However, you might want to define a table space without immediately allocating the associated data sets.

About this task

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 that you will not be using.

The deferral of allocating data sets is recommended when:
- Performance of the CREATE TABLESPACE statement is important
- Disk resource is constrained

Procedure

GUPI To defer the physical allocation of DB2-managed data sets:

Issue a CREATE TABLESPACE statement with the DEFINE NO clause.
The DEFINE NO clause is allowed on some DB2 objects, such as explicitly created LOB table spaces, auxiliary indexes, and XML indexes. Additionally, the IMPDSDEF subsystem parameter specifies whether DB2 defines the underlying data set for implicitly created table spaces and index spaces. When you specify this subsystem parameter as NO, the data set is not defined when the table space or index space is implicitly created.

Restriction: The DEFINE NO clause is not allowed for table spaces in a work file 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).
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.

Results

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.SYSTABLEPART contains a record of the created table space and an indication that the data sets are not yet allocated. GUPI

How DB2 extends data sets

When a data set is created, DB2 allocates a primary allocation space on a volume that has available space 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.

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: When a table space requires a new piece, the primary allocation quantity of the new piece is determined as follows:

- If the value of subsystem parameter MGEXTSZ is NO, the primary quantity is the PRIQTY value for the table space. If PRIQTY is not specified, the default for PRIQTY is used.
- If the value of MGEXTSZ is YES, the primary quantity is the maximum of the following values:
  - The quantity that is calculated through sliding scale methodology
  - The primary quantity from rule 1
  - The specified SECQTY value

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.

Extension failures

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.

DB2 might not be able to extend a data set if the data set is in an SMS data class that constrains the number of extents to less than the number that is required to reach full size. To prevent extension failures, make sure that the SMS data class setting for the number of allowed extents is large enough to accommodate 128 GB and 256 GB data sets.

Related concepts:

"Primary space allocation" on page 28

"Secondary space allocation" on page 28

Related tasks:

Avoiding excessively small extents (DB2 Performance)

DB2 space allocation

Primary and secondary space allocation sizes are the main factors that affect the amount of disk space that DB2 uses.

In general, the primary space allocation must be large enough to handle the storage needs that you anticipate. The secondary space allocation must be large enough for your applications to continue operating until the data set is reorganized.

If the secondary space allocation is too small, the data set might have to be extended more times to satisfy those activities that need a large space.

Primary space allocation

DB2 uses default values for primary space allocation of DB2-managed data sets.

The default values are:

- 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 the following table.

Table 2. 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, 64 GB, 128 GB, and 256 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, 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 (DFSMS Using the New Functions).

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

Table 3. 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
128	559	414
256	559	740

GUPI 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. GUPI

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.

Related concepts:

"How DB2 extends data sets" on page 26

Example of primary and secondary space allocation

Primary and secondary space allocation 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 4. 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

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.

You can also 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.

Related concepts:

"Managing your own data sets" on page 35

"Advantages of storage groups" on page 22

Migrating to DFSMShsm

If you decide to use DFSMShsm for your DB2 data sets, you should develop a migration plan with your system administrator.

About this task

With user-managed data sets, you can specify DFSMS classes on the Access Method Services DEFINE command. With DB2 storage groups, you can specify SMS classes in the CREATE STOGROUP statement, develop automatic class selection routines, or both.

Restriction: If you use the BACKUP SYSTEM utility to create system-level backups, do not use DFSMShsm to migrate DB2 table spaces and indexes. You can use DFSMShsm to migrate or recall archive log data sets.

Procedure

To enable DFSMS to manage your DB2 storage groups:

1. Issue either a CREATE STOGROUP or ALTER STOGROUP SQL statement.
2. Specify one or more asterisks as *volume-ID* in the VOLUMES option, and optionally, specify the SMS class options.

   The following example causes all database data set allocations and definitions to use nonspecific selection through DFSMS filtering services.

   GUPI

```
CREATE STOGROUP G202
VOLUMES ('*')
VCAT vcat name
DATACLAS dataclass name
MGMTCLAS management class name
STORCLAS storage class name;
```

GUPI

3. Define the SMS classes for your table space data sets and index data sets.
4. 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.
5. Use the system parameters SMSDCFL and SMSDCIX to assign table spaces and indexes to different DFSMS data classes.
   - SMSDCFL specifies a DFSMS 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 DFSMS 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.

   Important: If you specified the DATACLAS keyword in the CREATE STOGROUP statement, the DATACLAS value overrides the SMSDCFL and SMSDCIX values.

   Note: The system parameters SMSDCFL and SMSDCIX are deprecated in DB2 10.

Example

GUPI The following example shows how to create a storage group in a SMS managed subsystem:

```
CREATE STOGROUP SGOS0101
    VCAT REGSMS
    DATACLAS REGSMSDC
    MGMTCLAS REGSMSMC
    STORCLAS REGSMSSC;
```

GUPI

Related tasks:

"Letting SMS manage your DB2 storage groups" on page 114

"Enabling SMS to control DB2 storage groups" on page 24

How archive logs are recalled by DFSMShsm

DFSMShsm can automatically migrate and recall archive log data sets 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.

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.

The RECOVER utility and the DFSMSdss RESTORE command

The RECOVER utility can run the DFSMSdss RESTORE command, which generally uses extensions that are larger than the primary and secondary space allocation values of a data set.

The RECOVER utility runs 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.

When the RECOVER utility chooses a system-level backup for object-level recovery, DFSMShsm is used to restore the data sets from the system-level backup.

The DFSMSdss RESTORE command extends a data set differently than DB2, so after this command runs, you must alter the page set to contain extents that are defined by DB2.

Related tasks:

"Altering a page set to contain DB2-defined extents" on page 124

Related reference:

RECOVER (DB2 Utilities)

Considerations for using the BACKUP SYSTEM utility and DFSMShsm

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

Restriction: If you use the BACKUP SYSTEM utility to create system-level backups, do not use DFSMShsm to migrate DB2 table spaces and indexes.

The BACKUP SYSTEM utility uses copy pools. 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. Use the following DB2 naming convention for a copy pool:

```
DSN$locn-name$cp-type
```

The variables that are used in this naming convention are described in the following table.

Table 5. Naming convention variables for a copy pool

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.

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.

Tip: The BACKUP SYSTEM utility can dump the copy pools to tape automatically if you specify the options that enable that function.

Related tasks:

Managing DFSMShsm default settings when using the BACKUP SYSTEM, RESTORE SYSTEM, and RECOVER utilities

Related reference:

BACKUP SYSTEM (DB2 Utilities)

RESTORE SYSTEM (DB2 Utilities)

Incremental system-level backups

You can use the BACKUP SYSTEM utility to take incremental FlashCopy® backups of the data 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 DFSMSsms.

An incremental FlashCopy relationship is established for each source volume in the copy pool with corresponding target volumes. Each source volume can have only one incremental relationship. Therefore, multiple incremental FlashCopy backup versions are not supported.

The incremental FlashCopy backup feature is supported by the z/OS Version 1 Release 8, or later operating system. To support this feature the following keywords were added to the syntax of the BACKUP SYSTEM utility:

- ESTABLISH FCINCREMENTAL: Specifies that a persistent incremental FlashCopy relationship is to be established, if none exists for source copy volumes in the database copy pool. Use this keyword once to establish the persistent incremental FlashCopy relationships. Subsequent invocations of BACKUP SYSTEM (without this keyword) will automatically process the persistent incremental FlashCopy relationship.
- END FCINCREMENTAL: Specifies that a last incremental FlashCopy backup be taken and for the persistent incremental FlashCopy relationship to be withdrawn

for all of the volumes in the database copy pool. Use this keyword only if you do not need additional incremental FlashCopy backups of the database copy pool.

The first time that you use the ESTABLISH FCINCREMENTAL keyword in an invocation of the BACKUP SYSTEM utility the persistent incremental FlashCopy relationship is established. The incremental FlashCopy relationship exists until you withdraw it by specifying the END FCINCREMENTAL keyword in the utility control statement.

For the first invocation of BACKUP SYSTEM that specifies the ESTABLISH FCINCREMENTAL keyword, all of the tracks of each source volume are copied to their corresponding target volumes. For subsequent BACKUP SYSTEM requests, only the changed tracks are copied to the target volumes.

If you keep more than one DASD FlashCopy version of the database copy pool, you need to create full-copy backups for versions other than the incremental version.

For example, you decide to keep two DASD FlashCopy versions of your database copy pool. You invoke the BACKUP SYSTEM utility with the ESTABLISH FCINCREMENTAL keyword. A full-copy of each volume is created, because the incremental FlashCopy relationship is established for the first time. You invoke the BACKUP SYSTEM utility the next day. This request creates the second version of the backup. This version is a full-copy backup, because the incremental FlashCopy relationship is established with the target volumes in the first version. The following day you run the BACKUP SYSTEM utility again, but without the ESTABLISH FCINCREMENTAL keyword. The incremental version is the oldest version, so the incremental version is used for the FlashCopy backup. This time only the tracks that have changed are copied. The result is a complete copy of the source volume.

DFSMShsm allows multiple versions of FlashCopy backups for a copy pool. However, only one incremental FlashCopy backup is supported. If the database copy pool has two versions of FlashCopy backups, every other copy is an incremental copy since the oldest copy is replaced.

Managing your own data sets

You might choose to manage your own VSAM data sets for several reasons.

For example, consider the following reasons:

- You have a large nonpartitioned 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.

Tip: 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.

Related concepts:

"Advantages of storage groups" on page 22

Related information:

"Managing DB2 data sets with DFSMShsm" on page 31

Defining data sets

DB2 checks whether you have defined your data sets correctly.

About this task

You must define a data set for each of the following items:

- A simple or segmented table space
- A partition of a partitioned table space
- A partition of a partitioned index

You must define the data sets before you can issue the CREATE TABLESPACE, CREATE INDEX, or ALTER TABLE ADD PARTITION SQL statements.

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.

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.

Procedure

To define and manage VSAM data sets yourself:

1. Issue a DEFINE CLUSTER statement to create the data set.
2. Give each data set a name that complies with the following format:
   catname.DSNDB*x*.*dbname*.*psname*.*y*0001.*znnn*
3. In the DEFINE CLUSTER statement, specify 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. Specify that the data sets be LINEAR. Do not use RECORDSIZE; this attribute is invalid. Use the CONTROLINTERVALSIZE attribute if you are using variable-sized control intervals.
5. Specify the REUSE option. You must define the data set as REUSE before running the DSN1COPY utility.
6. Use SHAREOPTIONS(3,3).

Example

The following example code 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)
```

For user-managed data sets, you must pre-allocate shadow data sets prior to running the following against the table space:

- REORG with SHRLEVEL CHANGE
- REORG with SHRLEVEL REFERENCE
- CHECK INDEX with SHRLEVEL CHANGE
- CHECK DATA with SHRLEVEL CHANGE
- CHECK LOB with SHRLEVEL CHANGE

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 the following example code.

```
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')) -
```

In the previous example, 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.

What to do next

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, 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.

Related concepts:

Advantages of storage groups

Related reference:

Data set naming conventions

Data set naming conventions:

When you define a data set, you must give each data set a name that is in the correct format.

The correct format for the name of a data set is as follows:

catname.DSNDB*x*.*dbname*.*psname*.*y*0001.*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 you *do not* plan to run the following items:

- REORG with SHRLEVEL CHANGE or SHRLEVEL REFERENCE
- CHECK DATA with SHRLEVEL REFERENCE
- CHECK INDEX with SHRLEVEL REFERENCE
- CHECK LOB with SHRLEVEL REFERENCE

Define two data sets if you plan to run the following items:

- REORG with SHRLEVEL CHANGE or SHRLEVEL REFERENCE
- CHECK DATA with SHRLEVEL CHANGE
- CHECK INDEX with SHRLEVEL CHANGE
- CHECK LOB with SHRLEVEL CHANGE

Define one data set with a value of I for *y*, and one with a value of J for *y*.

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.

Extending user-managed data sets

A user-managed data set is allocated by using only volumes that are defined for that data set in the ICF catalog. Before the current volume runs out of space, you must extend the data set.

Procedure

To extend a user-managed data set:

Issue the Access Method Services commands ALTER ADDVOLUMES or ALTER REMOVEVOLUMES for candidate volumes.

Deleting user-managed data sets

If you manage the data sets of a storage structure yourself, at some point you might need to delete data sets.

Procedure

To delete a user-managed data set:

Issue the DELETE CLUSTER command 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, even if you defer building the index.

Exceptions:

- If you specify the USING VCAT clause for indexes that are not created on the DB2 catalog, you create and manage the data sets yourself.
- If you specify the DEFINE NO clause on a CREATE INDEX statement with the USING STOGROUP clause, DB2 defers the allocation of the data sets for the index space.

Procedure

GUPI To define your index space storage:

Issue a CREATE INDEX statement.
Optionally, for indexes that are not on DB2 catalog tables, include the USING clause to specify whether you want DB2-managed or user-managed data sets. For DB2-managed data sets, you can also specify the primary and secondary space allocation parameters for the index or partition in the USING clause. If you do not specify USING, DB2 assigns the index data sets to the default storage groups with the default space attributes.
For indexes on DB2 catalog tables, DB2 defines and manages the index data sets. The data sets are defined in the same SMS environment that is used for the catalog data sets with default space attributes. If you specify the USING clause for indexes on the catalog, DB2 ignores that clause.

GUPI

Results

Information about space allocation for the index is stored in the DB2 catalog table SYSIBM.SYSINDEXPART. Other information about the index is in the SYSIBM.SYSINDEXES table.

Related tasks:

Define the SMS environment for the DB2 catalog and directory data sets (DSNTIJSS) (DB2 Installation and Migration)

Related reference:

CREATE INDEX (DB2 SQL)

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*.

About this task

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.

Procedure

To create EA-enabled page sets:

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.

   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.

Creating partitioned table spaces that are enabled for EA

The following CREATE TABLESPACE statement creates an EA-enabled table space, SALESHX. Assume that a large query application uses this table space to record historical sales data for marketing statistics. The first USING clause establishes the MYSTOGRP storage group and space allocations for all partitions:

```
CREATE TABLESPACE SALESHX
  IN MYDB
  USING STOGROUP MYSTOGRP
    PRIQTY 4000
    SECQTY 130
    ERASE NO
  DSSIZE 16G
  NUMPARTS 48
    (PARTITION 46
     COMPRESS YES,
    PARTITION 47
     COMPRESS YES,
    PARTITION 48
     COMPRESS YES)
  LOCKSIZE PAGE
  BUFFERPOOL BP1
  CLOSE NO;
```

Related tasks:

"Creating table spaces explicitly" on page 42

Implementing DB2 table spaces

DB2 table spaces are storage structures that store one or more data sets, which store one or more tables.

Introductory concepts:

DB2 table spaces (Introduction to DB2 for z/OS)

Creation of table spaces (Introduction to DB2 for z/OS)

You can let DB2 create table spaces for your tables, or you can create them explicitly before you create the tables. DB2 supports several different types of table spaces. For more information about the different types, see Types of DB2 table spaces (Introduction to DB2 for z/OS) It is best to create universal table spaces in most cases. Other table space types are deprecated. That is, they are supported in DB2 10, but support might be removed in the future.

Related reference:

CREATE TABLE (DB2 SQL)

CREATE TABLESPACE (DB2 SQL)

Related information:

Migrate Classic Partitioned Table Spaces to Partitioned By Range Table Spaces (DB2 Utilities Development Blog)

Implicitly defined table spaces

DB2 implicitly creates a partition-by-growth or range-partitioned universal table space when you issue a CREATE TABLE statement without specifying an existing table space name.

When DB2 defines a table space implicitly, it completes the following actions:

- Generates a table space for you.
- Derives a table space name from the name of your table, according to the following rules:
  - The table space name is the same as the table name if the following 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 another 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 guarantee uniqueness.
- Uses default values for space allocation and other table space attributes.
- Creates the required LOB objects and XML objects.
- Enforces the UNIQUE constraint.
- Creates the primary key index.
- Creates the ROWID index, if the ROWID column is defined as GENERATED BY DEFAULT.

DB2 stores the names and attributes of all table spaces in the SYSIBM.SYSTABLESPACE catalog table, regardless of whether you define the table spaces explicitly or implicitly.

Related concepts:

Universal table spaces (Introduction to DB2 for z/OS)

Related reference:

CREATE TABLE (DB2 SQL)

SYSIBM.SYSTABLESPACE table (DB2 SQL)

Creating table spaces explicitly

DB2 can create table spaces for you. However, you might also create table spaces explicitly if you manage your own data sets, among other reasons.

Before you begin

For information about how DB2 can create table spaces for you, see "Implicitly defined table spaces."

About this task

GUPI

You can create different types of table spaces. Universal table spaces are best in most cases. Other types are deprecated. That is, they are supported in DB2 10, but support might be removed in the future. For more information about the different types, see Types of DB2 table spaces (Introduction to DB2 for z/OS).

Tip: You can alter table spaces after they are created, but the application of some statements, such as ALTER MAXPARTITIONS, prevent access to the database until alterations complete. Consider future growth when you define new table spaces.

Procedure

To explicitly create a table space:

1. Issue a CREATE TABLESPACE SQL statement.
2. Specify the attributes of the table space.

   The table space name is an identifier of up to 8 characters. You can qualify a table space name with a database name. Consider the following facts about naming guidelines for table spaces:

   - If you do not qualify an explicit table space with a database name, the default database name is DSNDB04.
   - If you do not explicitly specify a table space, DB2 implicitly creates the table space with a derived name. The name is derived based on the name of the table that is being created.
   - DB2 either implicitly creates a new database for the table space, or uses an existing implicitly created database.

   The following list introduces some of the clauses of the CREATE TABLESPACE statement that define the attributes of a table space:

   LOB
   Indicates that the table space is to be a large object (LOB) table space.

   DSSIZE
   Indicates the maximum size, in GB, for each partition or, for LOB table spaces, for each data set. The size of the table space depends on how many partitions are in the table space and on the DSSIZE. The maximum number of partitions for a partition-by-growth table space depends on the value that is specified for the MAXPARTITIONS option.

   FREEPAGE *integer*
   Specifies how often DB2 is to leave a page of free space when the table space or partition is loaded or reorganized. You specify that DB2 is to set aside one free page for every *integer* number of pages. Using free pages can improve performance for applications that perform high-volume inserts or that update variable-length columns. For details, see Reserving free space for table spaces (DB2 Performance).

   PCTFREE *integer*
   Indicates the percentage (*integer*) of each page that DB2 leaves as free space when the table is loaded or reorganized. Specifying PCTFREE can improve performance for applications that use high-volume inserts or that update variable-length columns. For details, see Reserving free space for table spaces (DB2 Performance).

   .

COMPRESS
Specifies that data is to be compressed. You can compress data in a table space to store more data on each data page. For details, see Compressing your data (DB2 Performance).

BUFFERPOOL *bpname*
Identifies the buffer pool that this table space is to use and determines the page size of the table space. The buffer pool is a portion of memory in which DB2 temporarily stores data for retrieval. For more information, see Tuning database buffer pools (DB2 Performance).

LOCKSIZE
Specifies the size of locks that DB2 is to use within the table space. DB2 uses locks to protect data integrity. Use of locks results in some processing costs, so choose the lock size carefully. For details, see Specifying the size of locks for a table space (DB2 Performance).

MAXPARTITIONS
Specifies the maximum number of partitions for a partition-by-growth table space. Within this clause, you can specify the NUMPARTS clause to specify the number of partitions that you want to create initially.

MEMBER CLUSTER
Specifies that data that is inserted by an INSERT operation is not clustered by the implicit clustering index (the first index), or the explicit clustering index. DB2 locates the data in the table space based on available space. You can use the MEMBER CLUSTER keyword on range-partitioned universal table spaces and partition-by-growth table spaces. For details, see Member affinity clustering (DB2 Data Sharing Planning and Administration).

NUMPARTS
Indicates that the table space is partitioned. If you also specify the MAXPARTITIONS clause, the table space is a partition-by-growth table space; otherwise, the table space is a range-partitioned universal table space.

MAXROWS
Specifies the maximum number of rows that DB2 places on each data page. The integer can range from 1 through 255. If you do not specify MAXROWS, the default number of rows is 255. Do not use MAXROWS for a LOB table space or a table space in a work file database.

Examples

The following examples illustrate how to use CREATE TABLE statements to create different types of table spaces.

Examples of creating partition-by-growth universal table spaces

The following examples show how to create a partition-by-growth universal table space.

Example 1: In the following SQL statement, the universal table space is implicitly created by a CREATE TABLE statement.

```
CREATE TABLE TEST02TB(
C1 SMALLINT,
C2 DECIMAL(9,2),
C3 CHAR(4))
PARTITIONING BY SIZE EVERY 4G
IN TEST02DB;
COMMIT;
```

Example 2: In the following SQL statement, the partition-by-growth universal table space has a maximum size of 2 GB for each partition, 4 pages per segment, with a maximum of 24 partitions for table space.

```
CREATE TABLESPACE TEST01TS IN TEST01DB USING STOGROUP SG1
DSSIZE 2G
MAXPARTITIONS 24
LOCKSIZE ANY
SEGSIZE 4;
COMMIT;
```

Creating range-partitioned universal table spaces

The following examples show how to create a range-partitioned universal table space (UTS).

Example 1: The following SQL statement defines a range-partitioned universal table space with 16 pages per segment and 55 partitions. This universal table space uses a storage group SG1 and has LOCKSIZE ANY.

```
CREATE TABLESPACE TS1 IN DB1 USING STOGROUP SG1
NUMPARTS 55 SEGSIZE 16
LOCKSIZE ANY;
```

Example 2: The following SQL statement defines a range-partitioned universal table space with 64 pages per segment and 7 defer-defined partitions. This universal table space uses a storage group SG1 and compresses every odd-numbered partition.

```
CREATE TABLESPACE TS1 IN DB1 USING STOGROUP SG1
NUMPARTS 7
(
PARTITION 1 COMPRESS YES,
PARTITION 3 COMPRESS YES,
PARTITION 5 COMPRESS YES,
PARTITION 7 COMPRESS YES
)
SEGSIZE 64
DEFINE NO;
```

Example 2:

Creating segmented table spaces (deprecated)

The following CREATE TABLESPACE statement creates a segmented table space with 32 pages in each segment:

```
CREATE TABLESPACE MYTS
  IN MYDB
  USING STOGROUP MYSTOGRP
    PRIQTY 30720
    SECQTY 10240
  SEGSIZE 32
  LOCKSIZE TABLE
  BUFFERPOOL BP0
  CLOSE NO;
```

What to do next

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.

GUPI

Related concepts:

"Managing your own data sets" on page 35

Related tasks:

"Altering table spaces" on page 117

Choosing data page sizes (DB2 Performance)

Choosing data page sizes for LOB data (DB2 Performance)

"Creating EA-enabled table spaces and index spaces" on page 40

Related reference:

CREATE TABLESPACE (DB2 SQL)

SYSIBM.SYSTABLESPACE table (DB2 SQL)

Implementing DB2 tables

Use the columns and rows of DB2 tables as logical structures for storing data.

Related concepts:

Creation of tables (Introduction to DB2 for z/OS)

Related tasks:

Creating tables (DB2 Application programming and SQL)

Related reference:

CREATE TABLE (DB2 SQL)

Creating base tables

When you create a table, DB2 records a definition of the table in the DB2 catalog.

About this task

Creating a table does not store the application data. You can put data into the table by using several methods, such as the LOAD utility or the INSERT statement.

Procedure

To create a base table that you designed:

Issue the CREATE TABLE statement.

Example

GUPI

The following CREATE TABLE statement creates the EMP table, which is in a database named MYDB and in a table space named MYTS:

```
CREATE TABLE EMP
      (EMPNO     CHAR(6)         NOT NULL,
       FIRSTNME  VARCHAR(12)     NOT NULL,
       LASTNAME  VARCHAR(15)     NOT NULL,
       DEPT      CHAR(3)                 ,
       HIREDATE  DATE                    ,
```

```
        JOB       CHAR(8)               ,
        EDL       SMALLINT              ,
        SALARY    DECIMAL(9,2)          ,
        COMM      DECIMAL(9,2)          ,
        PRIMARY KEY (EMPNO))
IN MYDB.MYTS;
```

This CREATE TABLE statement shows the definition of multiple columns.

GUPI

Related reference:

CREATE TABLE (DB2 SQL)

Guidelines for table names

Most organizations have naming conventions to ensure that objects are named in a consistent manner. Consider these basic requirements for table names.

The table name is an identifier of up to 128 characters. You can qualify the table name with an SQL identifier, which is a schema. When you define a table that is based directly on an entity, these factors also apply to the table names.

Creating tables that use hash organization

When you create a table, you can organize the table by hash to improve the performance queries that access individual rows.

About this task

When you create new tables on universal table spaces, you can enable hash access to that table by adding the organization-clause to your CREATE TABLE statement.

Procedure

To create a new table that is organized for hash access:

1. Specify ORGANIZE BY HASH in the organization-clause of your CREATE TABLE statement.
2. Specify UNIQUE followed by the column names for one or more columns that contain unique values in each row. You can specify more than one column-name as long as no two rows in the table have the same values in those columns. You can only specify columns that are defined as NOT NULL. You can specify a maximum of 64 columns to be used as unique identifiers for hash access. The sum of the column length attributes must not exceed 255. DB2 maintains the uniqueness of the hash key columns, and an index is not needed for this purpose.
3. Specify HASH SPACE followed by an integer and a modifier that specifies the size of the hash space. You can specify the size of the hash space in kilobytes, megabytes, and gigabytes. Specify:
   - `K` for kilobytes
   - `M` for megabytes
   - `G` for gigabytes

   You can specify a size that is larger than your data to minimize the overhead of access to data that overflows the hash space. The value that you specify is most important if you do not reorganize the table and specify the AUTOESTSPACE(YES) option soon after you create the table.

4. Commit the CREATE TABLE statement.

Results

After you organize a table for hash access, DB2 is likely but not certain to select hash access for statements that access the table.

Example

GUPI Consider the following CREATE TABLE statement:

```
CREATE TABLE EMP
      (EMPNO     CHAR(6)         NOT NULL,
       FIRSTNME  VARCHAR(12)     NOT NULL,
       LASTNAME  VARCHAR(15)     NOT NULL,
       DEPT      CHAR(3)                 ,
       HIREDATE  DATE                    ,
       JOB       CHAR(8)                 ,
       EDL       SMALLINT                ,
       SALARY    DECIMAL(9,2)            ,
       COMM      DECIMAL(9,2)            ,
       PRIMARY KEY (EMPNO))
IN MYDB.MYTS
ORGANIZE BY HASH UNIQUE (EMPNO)
HASH SPACE 64 M;
```

In this example the user creates a table named EMP in an explicitly defined table space, sets the EMPNO column as the unique identifier for hash access, and specifies a HASH SPACE size of 64 with the modifier M for megabytes. GUPI

What to do next

You can monitor the real-time-statistics information about your table to verify whether the hash access path is used regularly and to verify that the use of disk space is optimized.

Related tasks:

Organizing tables by hash for fast access to individual rows (DB2 Performance)

Managing space and page size for hash-organized tables (DB2 Performance)

Monitoring hash access (DB2 Performance)

"Altering tables to enable hash access" on page 150

"Altering the size of your hash spaces" on page 152

Related reference:

CREATE TABLE (DB2 SQL)

Creating temporary tables

Temporary tables are useful when you need to sort or query intermediate result tables that contain large numbers of rows and identify a small subset of rows to store permanently. The two types of temporary tables are *created temporary tables* and *declared temporary tables*.

About this task

Use a created temporary table when you need a permanent, sharable description of a table, and you need to store data only for the life of an application process. Use a declared temporary table when you need to store data for the life of an application process, but you don't need a permanent, sharable description of the table.

Procedure

To create a temporary table:

1. Determine the type of temporary table that you want to create.
2. Issue the appropriate SQL statement for the type of temporary table that you want to create:
   - To define a created temporary table, issue the CREATE GLOBAL TEMPORARY TABLE statement.
   - To define a declared temporary table, issue the DECLARE GLOBAL TEMPORARY TABLE statement.

Creating created temporary tables

If you need a permanent, sharable description of a table but need to store data only for the life of an application process, you can define and use a created temporary table.

About this task

DB2 does not log operations that it performs on created temporary tables; therefore, SQL statements that use created temporary tables can execute more efficiently. Each application process has its own instance of the created temporary table.

Procedure

To create a created temporary table:

Issue the CREATE GLOBAL TEMPORARY TABLE statement.

Example

GUPI

The following statement defines a created temporary table that is named TEMPPROD.

```
CREATE GLOBAL TEMPORARY TABLE TEMPPROD
  (SERIALNO     CHAR(8)     NOT NULL,
   DESCRIPTION  VARCHAR(60) NOT NULL,
   MFGCOSTAMT   DECIMAL(8,2)        ,
   MFGDEPTNO    CHAR(3)             ,
   MARKUPPCT    SMALLINT            ,
   SALESDEPTNO  CHAR(3)             ,
   CURDATE      DATE        NOT NULL);
```

GUPI

Related tasks:

Setting default statistics for created temporary tables (DB2 Performance)

Related reference:

CREATE GLOBAL TEMPORARY TABLE (DB2 SQL)

Creating declared temporary tables

If you need to store data for the life of an application process, but you don't need a permanent, sharable description of the table, you can define and use a declared temporary table.

Procedure

To create a declared temporary table:

Issue the DECLARE GLOBAL TEMPORARY TABLE statement. Unlike other DB2 DECLARE statements, DECLARE GLOBAL TEMPORARY TABLE is an executable statement that you can embed in an application program or issue interactively. You can also dynamically prepare the statement.
When a program in an application process issues a DECLARE GLOBAL TEMPORARY TABLE statement, DB2 creates an empty instance of the table. You can populate the declared temporary table by using INSERT statements, modify the table by using searched or positioned UPDATE or DELETE statements, and query the table by using SELECT statements. You can also create indexes on the declared temporary table. The definition of the declared temporary table exists as long as the application process runs.
At the end of an application process that uses a declared temporary table, DB2 deletes the rows of the table and implicitly drops the description of the table.

Example

GUPI

The following statement defines a declared temporary table, TEMP_EMP. (This example assumes that you have already created the WORKFILE database and corresponding table space for the temporary table.)

```
DECLARE GLOBAL TEMPORARY TABLE  SESSION.TEMP_EMP
   (EMPNO  CHAR(6)        NOT NULL,
    SALARY DECIMAL(9, 2)          ,
    COMM   DECIMAL(9, 2));
```

If specified explicitly, the qualifier for the name of a declared temporary table, must be SESSION. If the qualifier is not specified, it is implicitly defined to be SESSION.

GUPI

Related reference:

DECLARE GLOBAL TEMPORARY TABLE (DB2 SQL)

Distinctions between DB2 base tables and temporary tables

DB2 base tables and the two types of temporary tables have several distinctions.

The following table summarizes important distinctions between base tables, created temporary tables, and declared temporary tables.

Table 6. 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 that is used by created temporary tables is reset by the following commands: **START DB2**, **START DATABASE**, and **START DATABASE**(*dbname*) SPACENAM(*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*) SPACENAM(*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 that is 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 6. 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 to the same instance at the current server. If the table name that is being referenced is not qualified, DB2 implicitly qualifies the name using the standard DB2 qualification rules that apply 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 that is being referenced is not qualified, DB2 implicitly qualifies the name using the standard DB2 qualification rules that apply 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 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 6. Important distinctions between DB2 base tables and DB2 temporary tables (continued)

Area of distinction	Base tables	Created temporary tables	Declared temporary tables
Table space requirements and table size limitations	The table can be stored in implicitly created table spaces and 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 that are 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 a table space in the work file database. 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 that are 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.

Related concepts:

Temporary tables (DB2 Application programming and SQL)

Related tasks:

Creating temporary tables

Setting default statistics for created temporary tables (DB2 Performance)

Related reference:

CREATE GLOBAL TEMPORARY TABLE (DB2 SQL)

DECLARE GLOBAL TEMPORARY TABLE (DB2 SQL)

Creating temporal tables

You can create a temporal table, which is a table that records the period of time when a row is valid.

Temporal tables

A *temporal table* is a table that records the period of time when a row is valid.

A *period* is an interval of time that is defined by two datetime columns in a temporal table. A period contains a begin column and an end column. The begin column indicates the beginning of the period, and the end column indicates the end of the period. The beginning value of a period is inclusive, but the ending value of a period is exclusive. For example, if the begin column has a value of '01/01/1995', that date belongs in the row. Whereas, if the end column has a value of '03/21/1995', that date is not part of the row.

DB2 supports two types of periods, which are the system period (SYSTEM_TIME) and the application period (BUSINESS_TIME).

The *system period* consists of a pair of columns with system-maintained values that indicate the period of time when a row is valid. The begin column contains the timestamp value for when a row is created. The end column contains the timestamp value for when a row is updated or deleted.

The system period is meaningful because of *system-period data versioning*. System-period data versioning specifies that old rows are archived into another table. The table that contains the current active rows of a table is called the *system-period temporal table*. The table that contains the archived rows is called the *history table*. You can delete the rows from the history table when those rows are no longer needed, if you have the correct authorization. When you define a base table to use system-period data versioning, or when you define system-period data versioning on an existing table, you must create a history table, specify a name for the history table, and create a table space to hold that table. You define versioning by issuing the ALTER TABLE ADD VERSIONING statement with the USE HISTORY TABLE clause.

The *application period* consists of a pair of columns with application-maintained values that indicate the period of time when a row is valid. The begin column contains the value from which a row is valid. The end column contains the value for when a row stops being valid. A table with only an application period is called an *application-period temporal table*.

When you use the application period, determine the need for DB2 to enforce uniqueness across time. You can create a UNIQUE index that is unique over a period of time.

A *bitemporal table* is a table that is both a system-period temporal table and an application-period temporal table. You can use a bitemporal table to keep application period information and system-based historical information. Therefore, you have a lot of flexibility in how you query data based on periods of time.

Related concepts:

"Guidelines for history tables" on page 57

"Recovery of temporal tables with system-period data versioning" on page 469

"Guidelines for system-period temporal tables" on page 56

Related tasks:

"Creating a system-period temporal table" on page 55

"Creating an application-period temporal table" on page 57

Restrictions for system-period data versioning:

When a table is enabled for system-period data versioning, certain restrictions apply.

- For point-in-time recovery, to keep the data in the system-period temporal table and the data in the history table synchronized, you must recover the table spaces for both tables as a set. You can recover the table spaces individually only if you specify the VERIFYSET NO option in the RECOVER utility statement.
- You cannot run a utility operation that deletes data from a system-period temporal table. These utilities include LOAD REPLACE, REORG DISCARD, and CHECK DATA DELETE YES.
- You cannot run the CHECK DATA utility with the options LOBERROR INVALIDATE, AUXERROR INVALIDATE, or XMLERROR INVALIDATE on a system-period temporal table. The CHECK DATA utility will fail with return code 8 and message DSNU076.
- You cannot alter the schema (data type, check constraint, referential constraint, etc.) of a system-period temporal table or history table; however, you can add a column to system-period temporal table.
- You cannot drop the history table or its table space.

- You cannot define a clone table on the system-period temporal table or the history table.
- You cannot create another table in table space for either the system-period temporal table or history table.
- On the history table, you cannot use the UPDATE, DELETE, or SELECT statement syntax that specifies the application period.
- You cannot rename a column or table name of a system-period temporal table or a history table.

Related reference:

CHECK DATA (DB2 Utilities)

Creating a system-period temporal table

You can create a temporal table that has a system period and define system-period data versioning on the table, so that the data is versioned after insert, update, and delete operations.

About this task

A system period is a system-maintained period in which DB2 maintains the beginning and ending timestamp values for a row. The begin column contains the timestamp value for when a row is created. The end column contains the timestamp value for when a row is updated or deleted.

Procedure

To create a temporal table with a system period and define system-period data versioning on the table:

1. Issue a CREATE TABLE statement with a SYSTEM_TIME clause. This table is a system-period temporal table.
2. Issue a CREATE TABLE statement to create a history table that receives the old rows from the system-period temporal table.
3. Issue the ALTER TABLE ADD VERSIONING statement with the USE HISTORY TABLE clause to define system-period data versioning on the table. By defining system-period data versioning, you establish a link between the system-period temporal table and the history table.

Example

The following examples show how you can create a temporal table with a system period, create a history table, and then define system-period data versioning on the table. Also, a final example shows how to insert data.

GUPI

This example shows a CREATE TABLE statement for creating a temporal table with a SYSTEM_TIME period:

```
CREATE TABLE policy_info
(policy_id CHAR(10) NOT NULL,
coverage INT NOT NULL,
sys_start TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW BEGIN,
sys_end TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW END,
create_id TIMESTAMP(12) GENERATED ALWAYS AS TRANSACTION START ID,
PERIOD SYSTEM_TIME(sys_start,sys_end));
```

This example shows a CREATE TABLE statement for creating a history table:

```
CREATE TABLE hist_policy_info
(policy_id CHAR(10) NOT NULL,
coverage INT NOT NULL,
sys_start TIMESTAMP(12) NOT NULL,
sys_end TIMESTAMP(12) NOT NULL,
create_id TIMESTAMP(12))
```

To define versioning, issue the ALTER TABLE ADD VERSIONING statement with the USE HISTORY TABLE clause, which establishes a link between the system-period temporal table and the history table:

```
ALTER TABLE policy_info
ADD VERSIONING USE HISTORY TABLE hist_policy_info;
```

The following example shows how to insert data in the POLICY_ID and COVERAGE columns of the POLICY_INFO table:

```
INSERT INTO POLICY_INFO (POLICY_ID, COVERAGE)
VALUES('A123', 12000);
```

GUPI

Related concepts:

"Temporal tables" on page 53

Related reference:

CREATE TABLE (DB2 SQL)

ALTER TABLE (DB2 SQL)

Guidelines for system-period temporal tables:

For a table to be a system-period temporal table and be defined with system-period data versioning, the table must meet certain criteria.

A system-period temporal table must have:

- A corresponding history table that stores deleted rows
- A column defined as TIMESTAMP(12) NOT NULL with the GENERATED ALWAYS attribute for a begin column of a SYSTEM_TIME period. You cannot update this column, and the data type cannot be a user-defined type.
- A column defined as TIMESTAMP(12) NOT NULL with the GENERATED ALWAYS attribute for an end column of a SYSTEM_TIME period. You cannot update this column, and the data type cannot be a user-defined type.
- A system period (SYSTEM_TIME) specified on the two timestamp columns in which the first column is the begin column and the second column is the end column
- Only one table defined in the table space
- A complete table definition

A system-period temporal table cannot be a materialized query table and cannot have:

- A clone table defined on it
- A column mask
- Row permission
- A security label column

Related tasks:

"Adding a system period and system-period data versioning to an existing table" on page 153

Guidelines for history tables:

To define system-period data versioning on a system-period temporal table, the corresponding history table must meet certain criteria.

A history table must have:

- The same number of columns as the system-period temporal table that it corresponds to
- Columns with the same names, data types, null attributes, CCSIDs, subtypes, hidden attributes, and field procedures as the corresponding system-period temporal table

  If the system-period temporal table has a column of ROWID GENERATED ALWAYS or ROWID GENERATED BY DEFAULT, the history table must have a corresponding ROWID GENERATED ALWAYS column. Otherwise, the history table should not have any GENERATED columns.
- Only one table defined in the table space
- A complete table definition

A history table cannot be a materialized query table and cannot have:

- Identity columns or row change timestamp columns
- ROW BEGIN, ROW END, or TRANSACTION START ID attributes
- A clone table defined on it
- A column mask
- Row permission
- A security label column
- A system period or an application period

Related tasks:

"Adding a system period and system-period data versioning to an existing table" on page 153

Creating an application-period temporal table

An *application-period temporal table* is a type of temporal table where you maintain the values that indicate when a row is valid. The other type of temporal table is a *system-period temporal table* where DB2 maintains the values that indicate when a row is valid.

About this task

When you create an application-period temporal table, you define begin and end columns to indicate the application period, or period of time when the row is valid. The begin column contains the time from which a row is valid. The end column contains the time when a row stops being valid.

Procedure

To create an application-period temporal table:

Issue a CREATE TABLE statement with the following items:

- Two columns to define the application period. These columns are the begin and end columns. They must be type TIMESTAMP(6) WITHOUT TIME ZONE NOT NULL or DATE NOT NULL, and they must be the same type. The data type cannot be a user-defined type.
- The BUSINESS_TIME clause.

Example

GUPI

Example of creating an application-period temporal table

The following example SQL statements create a table with an application period and a unique index:

```
CREATE TABLE policy_info
(policy_id CHAR(4) NOT NULL,
coverage INT NOT NULL,
bus_start DATE NOT NULL,
bus_end DATE NOT NULL,
PERIOD BUSINESS_TIME(bus_start, bus_end));

CREATE UNIQUE INDEX ix_policy
ON policy_info (policy_id, BUSINESS_TIME WITHOUT OVERLAPS);
```

GUPI

Related concepts:

"Temporal tables" on page 53

Related reference:

CREATE TABLE (DB2 SQL)

CREATE INDEX (DB2 SQL)

Creating bitemporal tables

You can create a bitemporal table that maintains both system-based historical information and application period information.

About this task

You maintain system-based historical information by adding a system period to a table, and you maintain application period information by adding an application period to the table.

Procedure

To create a bitemporal table and define system-period data versioning on the table:

1. Issue a CREATE TABLE statement with both the SYSTEM_TIME clause and the BUSINESS_TIME clause.
2. Issue a CREATE TABLE statement to create a history table that receives the old rows from the bitemporal table.
3. Issue the ALTER TABLE ADD VERSIONING statement with the USE HISTORY TABLE clause to define system-period data versioning and establish a link between the bitemporal table and the history table.

Example

The following examples show how you can create a bitemporal table, create a history table, and then define system-period data versioning.

GUPI

This example shows a CREATE TABLE statement with the SYSTEM_TIME and BUSINESS_TIME clauses for creating a bitemporal table:

```
CREATE TABLE policy_info
(policy_id CHAR(4) NOT NULL,
coverage INT NOT NULL,
bus_start DATE NOT NULL,
bus_end DATE NOT NULL,
sys_start TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW BEGIN,
sys_end TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW END,
create_id TIMESTAMP(12) GENERATED ALWAYS AS TRANSACTION START ID,
PERIOD BUSINESS_TIME(bus_start, bus_end),
PERIOD SYSTEM_TIME(sys_start, sys_end));
```

This example shows a CREATE TABLE statement for creating a history table:

```
CREATE TABLE hist_policy_info
(policy_id CHAR(4) NOT NULL,
coverage INT NOT NULL,
bus_start DATE NOT NULL,
bus_end DATE NOT NULL,
sys_start TIMESTAMP(12) NOT NULL,
sys_end TIMESTAMP(12) NOT NULL,
create_id TIMESTAMP(12));
```

This example shows the ALTER TABLE ADD VERSIONING statement with the USE HISTORY TABLE clause that establishes a link between the bitemporal table and the history table to enable system-period data versioning. Also, a unique index is added to the bitemporal table.

```
ALTER TABLE policy_info
ADD VERSIONING USE HISTORY TABLE hist_policy_info;
```

```
CREATE UNIQUE INDEX ix_policy
ON policy_info (policy_id, BUSINESS_TIME WITHOUT OVERLAPS);
```

GUPI

Related concepts:

"Guidelines for history tables" on page 57

"Guidelines for system-period temporal tables" on page 56

Related reference:

CREATE TABLE (DB2 SQL)

ALTER TABLE (DB2 SQL)

Finding the name of a history table

A history table is a base table that is associated with a system-period temporal table. A history table is used by DB2 to store the historical versions of the rows from the associated system-period temporal table.

About this task

If you know the name of the system-period temporal table, you can find the name of the corresponding history table.

Procedure

GUPI

To find the name of a history table:

Issue a SELECT statement, such as:

```
SELECT VERSIONING_SCHEMA, VERSIONING_TABLE FROM SYSIBM.SYSTABLES WHERE
NAME = 'table-name' AND CREATOR = 'creator-name'
```

GUPI

Querying temporal tables

You can query a temporal table to retrieve data based on the time criteria that you specify.

About this task

A temporal table that includes a system period (SYSTEM_TIME) and is defined with system-period data versioning is a system-period temporal table. A temporal table that includes an application period (BUSINESS_PERIOD) is an application-period temporal table.

Procedure

To query a temporal table:

Issue a SELECT statement, and in the *table-reference* of the FROM clause, specify a *period-specification*. A *period-specification* consists of the following clauses:

- FOR SYSTEM TIME or FOR BUSINESS TIME to indicate whether you want to query a system-period temporal table or an application-period temporal table
- AS OF, FROM, or BETWEEN to indicate the time criteria for which you want data

GUPI

The following example shows how you can request data based on time criteria from a system-period temporal table.

```
SELECT policy_id, coverage FROM policy_info
FOR SYSTEM_TIME AS OF '2009-01-08-00.00.00.000000000000';
```

Likewise, the following example shows how you can request data based on time criteria from an application-period temporal table.

```
SELECT policy_id, coverage FROM policy_info
FOR BUSINESS_TIME AS OF '2008-06-01';
```

GUPI

If you are requesting historical data from a system-period temporal table that is defined with system-period data versioning, DB2 rewrites the query to include data from the history table.

Related concepts:

"Temporal tables" on page 53

Related reference:

from-clause (DB2 SQL)

table-reference (DB2 SQL)

Creating materialized query tables

Materialized query tables improve the performance of complex queries that operate on very large amounts of data. Use the CREATE TABLE statement to create a materialized query table.

About this task

DB2 uses a materialized query table to precompute the results of data that is derived from one or more tables. When you submit a query, DB2 can use the results that are stored in a materialized query table rather than compute the results from the underlying source tables on which the materialized query table is defined.

Procedure

To create a new materialized query table:

Issue the CREATE TABLE statement.

Example

GUPI

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 ACCOUNTID, 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.

GUPI

Creating tables that use table-controlled partitioning

Table-controlled partitioning does not require an index for partitioning and is defined by PARTITION clauses on the CREATE TABLE statement.

Procedure

To create a table that uses table-controlled partitioning:

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.

Example

GUPI

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'));
```

GUPI

Related concepts:

"Scenario: Moving from index-controlled to table-controlled partitioning" on page 198

Related reference:

CREATE TABLE (DB2 SQL)

Differences between partitioning methods

When possible, use table-controlled partitioning instead of index-controlled partitioning.

DB2 supports two types of partitioning for partitioned table spaces (non-universal):

index-controlled partitioning
: Partitioning is controlled by an index called a *partitioning index*. When you define a partitioning index on a table in a partitioned table space, you specify the partitioning key and the limit key values in the PARTITION clause of the CREATE INDEX statement.

table-controlled partitioning
: Partitioning is not controlled by an index. You 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. When you use this type of partitioning, an index is not required for partitioning.

The following table lists the differences between the two partitioning methods.

Table 7. Differences between table-controlled and index-controlled partitioning

Table-controlled partitioning	Index-controlled partitioning
A partitioning index is not required; a clustering index is not required.	A partitioning index is required; a 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.

To prevent the creation of any new partitioned tables that use index-controlled partitioning, set the PREVENT_NEW_IXCTRL_PART subsystem parameter to YES.

Related concepts:

"Scenario: Moving from index-controlled to table-controlled partitioning" on page 198

Related reference:

CREATE INDEX (DB2 SQL)

Subsystem parameters that are not on installation panels (DB2 Installation and Migration)

Automatic conversion to table-controlled partitioning

Some index operations cause DB2 to automatically convert an index-controlled partitioned table space to a table-controlled partitioned table space.

GUPI

Consider the following operations:

- Use the CREATE INDEX statement with the PARTITIONED clause to create a partitioned index on an index-controlled partitioned table space.
- Use the CREATE INDEX statement 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 the ALTER INDEX statement with the NOT CLUSTER clause on a partitioning index that is on an index-controlled partitioned table space.
- Use the DROP INDEX statement to drop a partitioning index on an index-controlled partitioned table space.
- Use the ALTER TABLE statement to add a new partition, change a partition boundary, or rotate a partition 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.

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 or 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, unless the high-limit key value is already being enforced, or the last partition is empty.
- When altering a partition that converts a table space from index-controlled partitioning to table-controlled partitioning, DB2 changes the existing high-limit key to the highest value that is possible for the data types of the limit key columns. After the conversion to table-controlled partitioning, DB2 changes the high-limit key value back to the user-specified value and puts the last partition into a REORP state.

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

You can use the IX_TB_PART_CONV_EXCLUDE subsystem parameter to specify whether to exclude trailing columns from the table-controlled partitioning keys when table spaces are converted from index-controlled partitioning to table-controlled partitioning. The default value is YES, which means the DB2 subsystem excludes trailing columns from the original partitioning key in the definition of the new partitioning key. The trailing columns are the columns that do not affect the partitioning. Trailing columns have all X'FF' values in the LIMITKEY column value of the SYSIBM.SYSINDEXPART catalog table. A value of NO means the DB2 subsystem uses all columns of the original partitioning index to define the new partitioning key.

The IX_TB_PART_CONV_EXCLUDE subsystem parameter does not affect the automatic conversion to table-controlled partitioning in the following cases:

- When issuing the ALTER TABLE statement to add a new partition, change a partition boundary, or rotate a partition to last on an index-controlled partitioned table space, In these cases, the DB2 subsystem uses all columns of the original partitioning index to define the new partitioning key.
- When issuing the CREATE INDEX statement with a PART VALUES clause and without a CLUSTER clause. In this case, the DB2 subsystem uses the partitioning key that is specified by the CREATE INDEX statement.

GUPI

Related concepts:

“Scenario: Moving from index-controlled to table-controlled partitioning” on page 198

Related reference:

Subsystem parameters that are not on installation panels (DB2 Installation and Migration)

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, DB2 prevents the INSERT of a row with a null value for the key column, unless a partition is created that specifies MAXVALUE, which will hold the null values.
- If the partitioning key is descending, DB2 prevents the INSERT of a row with a null value for the key column, unless a partition is created that specifies MINVALUE, which will hold the null values.

Example 1:

GUPI

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, because no partition specifies MAXVALUE. If the CREATE TABLE statement had specified the key as descending and the first partition specified MINVALUE, 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 2: 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.

GUPI

Creating tables that use index-controlled partitioning

Tables that use index-controlled partitioning require a partitioning index. Because the index is created separately from the associated table, you cannot insert data into the table until the partitioning index is created.

About this task

Recommendation: Create partitioned tables with table-controlled partitioning instead of index-controlled partitioning.

Restriction: If the PREVENT_NEW_IXCTRL_PART subsystem parameter is set to YES, you cannot create tables that use index-controlled partitioning.

Procedure

To create tables that use index-controlled partitioning:

Define a partitioning index on a table in a partitioned table space. Specify the partitioning key and the limit key values in the PARTITION clause of the CREATE INDEX statement.

Related concepts:

"Differences between partitioning methods" on page 62

Related tasks:

"Creating tables that use table-controlled partitioning" on page 61

Related reference:

Subsystem parameters that are not on installation panels (DB2 Installation and Migration)

Creating a clone table

You can create a clone table on an existing base table at the current server by using the ALTER TABLE statement.

Before you begin

Although the ALTER TABLE syntax is used to create a clone table, the authorization that is granted as part of the clone creation process is the same as you would get during regular CREATE TABLE processing. The schema for the clone table will be the same as for the base table.

The base table must meet the following requirements:

- Be a universal table space
- Reside in a DB2-managed table space
- Be the only table in the table space
- Not already have a clone table defined on it
- Not be part of any referential constraints
- Not be defined with any AFTER triggers
- Not be a materialized query table
- Not be a created global temporary table or a declared global temporary table
- Not have any data sets that still need to be created. For example, you cannot create a clone table on a base table that resides in a table space that was created by using the DEFINE NO option and that has VSAM data sets that still need to be created.
- Not have any pending alterations or active versioning. The values in the OLDEST_VERSION and CURRENT_VERSION columns of the SYSIBM.SYSTABLESPACE table must be identical.
- Not have an incomplete definition

About this task

Restrictions: In addition, the following restrictions apply to clone tables:

- A clone table uses the statistics from the base table. RUNSTATS does not collect statistics on a clone table, and Access Path Selection (APS) does not use RUNSTATS statistics when accessing a clone table. This is in contrast to real-time statistics, which keeps statistics for both the base and clone objects. Also, autonomic statistics are not collected on a clone table.
- Catalog and directory tables cannot be cloned.
- Indexes cannot be created on a clone table. Indexes can be created on the base table but not on the clone table. Indexes that are created on the base table apply to both the base and clone tables.
- BEFORE triggers can be created on the base table but not on the clone table. BEFORE triggers that are created on the base table apply to both the base and clone tables.
- You cannot rename a base table that has a clone relationship.
- You cannot clone an RTS table.
- You cannot drop an AUX table or an AUX index on an object that is involved in cloning.
- You cannot alter any table, or column attributes of a base table or clone table when the objects are involved with cloning.
- The maximum number of partitions cannot be altered when a clone table resides in a partition-by-growth table space.

Procedure

GUPI To create a clone table:

Issue the ALTER TABLE statement with the ADD CLONE option.
Creating or dropping a clone table does not impact applications that are accessing base table data. No base object quiesce is necessary, and this process does not invalidate packages or the dynamic statement cache.

Example

The following example shows how to create a clone table by issuing the ALTER TABLE statement with the ADD CLONE option:

```
ALTER TABLE base-table-name ADD CLONE clone-table-name
```

GUPI

Related tasks:

"Exchanging data between a base table and clone table"

Related reference:

ALTER TABLE (DB2 SQL)

Exchanging data between a base table and clone table

You can exchange table data and index data between the base table and clone table by using the EXCHANGE statement.

Procedure

GUPI

To exchange data between the base table and clone table:

Issue an EXCHANGE statement with the DATA BETWEEN TABLE *table-name1* AND *table-name2* syntax.
Exchanging data between the base table and the clone table does not invalidate packages.

Example

```
EXCHANGE DATA BETWEEN TABLE table-name1 AND table-name2
```

GUPI

What to do next

After a data exchange, the base and clone table names remain the same as they were prior to the data exchange. No data movement actually takes place. The instance numbers in the underlying VSAM data sets for the objects (tables and indexes) do change, and this has the effect of changing the data that appears in the base and clone tables and their indexes. For example, a base table exists with the data set name *I0001.*. The table is cloned and the clone's data set is initially named *.I0002.*. After an exchange, the base objects are named *.I0002.* and the clones are named *I0001.*. Each time that an exchange happens, the instance numbers that represent the base and the clone objects change, which immediately changes the data contained in the base and clone tables and indexes.

Related tasks:

"Creating a clone table" on page 66

Related reference:

EXCHANGE (DB2 SQL)

Implementing DB2 views

When you design your database, you might need to give users access to only certain pieces of data. You can give users access by designing and using views.

You can use views to perform the following tasks:

- Control access to a table
- Make data easier to use
- Simplify authorization by granting access to a view without granting access to the table
- Show only portions of data in the table
- Show summary data for a given table
- Combine two or more tables in meaningful ways

Creating DB2 views

You can create a view on tables or on other views at the current server.

Before you begin

Before you create different column names for your view, remember the naming conventions that you established when designing the relational database.

Procedure

GUPI

To define a view:

Issue the CREATE VIEW SQL statement.
Unless you specifically list different column names after the view name, the column names of the view are the same as those of the underlying table.

GUPI

Example

Example of defining a view on a single table: Assume that you want to create a view on the DEPT table. Of the four columns in the table, the view needs only three: DEPTNO, DEPTNAME, and MGRNO. The order of the columns that you specify in the SELECT clause is the order in which they appear in the view:

GUPI

```
CREATE VIEW MYVIEW AS
 SELECT DEPTNO,DEPTNAME,MGRNO
 FROM DEPT;
```

GUPI

In this example, no column list follows the view name, MYVIEW. Therefore, the columns of the view have the same names as those of the DEPT table on which it is based. You can execute the following SELECT statement to see the view contents:

GUPI

```
SELECT * FROM MYVIEW;
```

GUPI

The result table looks like this:

```
DEPTNO DEPTNAME              MGRNO
====== ==================== ======
A00    CHAIRMANS OFFICE      000010
B01    PLANNING              000020
C01    INFORMATION CENTER    000030
D11    MANUFACTURING SYSTEMS 000060
E21    SOFTWARE SUPPORT      ------
```

Example of defining a view that combines information from several tables: You can create a view that contains a union of more than one table. DB2 provides two types of joins—an outer join and an inner join. An outer join includes rows in which the values in the join columns don't match, and rows in which the values match. An inner join includes only rows in which matching values in the join columns are returned.

The following example is an inner join of columns from the DEPT and EMP tables. The WHERE clause limits the view to just those columns in which the MGRNO in the DEPT table matches the EMPNO in the EMP table:

GUPI

```
CREATE VIEW MYVIEW AS
 SELECT DEPTNO, MGRNO, LASTNAME, ADMRDEPT
 FROM DEPT, EMP
 WHERE EMP.EMPNO = DEPT.MGRNO;
```

GUPI

The result of executing this CREATE VIEW statement is an inner join view of two tables, which is shown below:

```
DEPTNO MGRNO  LASTNAME ADMRDEPT
====== ====== ======== ========
A00    000010 HAAS     A00
B01    000020 THOMPSON A00
C01    000030 KWAN     A00
D11    000060 STERN    D11
```

Related tasks:

"Altering DB2 views" on page 165

"Dropping DB2 views" on page 72

Related reference:

CREATE VIEW (DB2 SQL)

Guidelines for view names

The name for a view is an identifier of up to 128 characters.

The following example shows a view name:

Object	Name
View	MYVIEW

Use the CREATE VIEW statement to define and name a view. Unless you specifically list different column names after the view name, the column names of the view are the same as those of the underlying table. When you create different column names for your view, remember the naming conventions that you established when designing the relational database.

How DB2 inserts and updates data through views

After you define a view, you can refer to the name of a view in an INSERT, UPDATE, or DELETE statement.

GUPI

To ensure that the insert or update conforms to the view definition, specify the WITH CHECK OPTION clause. The following example illustrates some undesirable results of omitting that check.

Example 1: Suppose that you define a view, V1, as follows:

```
CREATE VIEW V1 AS
 SELECT * FROM EMP
 WHERE DEPT LIKE 'D%';
```

A user with the SELECT privilege on view V1 can see the information from the EMP table for employees in departments whose IDs begin with D. The EMP table has only one department (D11) with an ID that satisfies the condition.

Assume that a user has the INSERT privilege on view V1. A user with both SELECT and INSERT privileges can insert a row for department E01, perhaps erroneously, but cannot select the row that was just inserted.

The following example shows an alternative way to define view V1.

Example 2: You can avoid the situation in which a value that does not match the view definition is inserted into the base table. To do this, instead define view V1 to include the WITH CHECK OPTION clause:

```
CREATE VIEW V1 AS SELECT * FROM EMP
 WHERE DEPT LIKE 'D%' WITH CHECK OPTION;
```

With the new definition, any insert or update to view V1 must satisfy the predicate that is contained in the WHERE clause: DEPT LIKE ‘D%'. The check can be valuable, but it also carries a processing cost; each potential insert or update must be checked against the view definition. Therefore, you must weigh the advantage of protecting data integrity against the disadvantage of performance degradation.

GUPI

Dropping DB2 views

You can drop a DB2 view by removing the view at the current server.

Procedure

To drop a view:

Issue the DROP VIEW statement.

Related tasks:

"Altering DB2 views" on page 165

"Creating DB2 views" on page 69

Related reference:

DROP (DB2 SQL)

Implementing DB2 indexes

DB2 uses indexes for a variety of reasons. DB2 indexes enforce uniqueness on column values, as in the case of parent keys. DB2 indexes are also used to cluster data, to partition tables, to provide access paths to data, and to order retrieved data without a sort.

Related concepts:

"Indexes on table columns" on page 19

Creation of indexes (Introduction to DB2 for z/OS)

Indexes that are padded or not padded (Introduction to DB2 for z/OS)

Related tasks:

Designing indexes for performance (DB2 Performance)

Compressing indexes (DB2 Performance)

Creating DB2 indexes

When you define an index, DB2 builds and maintains an ordered set of pointers to rows of a base table or an auxiliary table.

Before you begin

Before you define an index, you need to define the table.

Procedure

GUPI To create a partitioning index, or a secondary index, and an index space at the current server:

Issue the CREATE INDEX statement, and specify an index key.

Example

The following example creates a unique index on the EMPPROJACT table. A composite key is defined on two columns, PROJNO and STDATE.

```
 CREATE UNIQUE INDEX XPROJAC1
  ON EMPPROJACT
 (PROJNO ASC,
 STDATE ASC)
INCLUDE(EMPNO, FIRSTNAME)
```

The INCLUDE clause, which is applicable only on unique indexes, specifies additional columns that you want appended to the set of index key columns. Any columns that are included with this clause are not used to enforce uniqueness. These included columns might improve the performance of some queries through index only access. Using this option might eliminate the need to access data pages for more queries and might eliminate redundant indexes.

If you issue `SELECT PROJNO, STDATE, EMPNO, and FIRSTNAME` to the table on which this index resides, all of the required data can be retrieved from the index without reading data pages. This option might improve performance.

GUPI

Related tasks:

"Dropping and redefining a DB2 index" on page 171

Related reference:

CREATE INDEX (DB2 SQL)

Guidelines for defining indexes

By following certain guidelines, you can successfully work with indexes.

Index names

The name for an index is an identifier of up to 128 characters. You can qualify this name with an identifier, or schema, of up to 128 characters.

Example: The following example shows an index name:

Object	**Name**
Index	MYINDEX

The index space name is an eight-character name, which must be unique among names of all index spaces and table spaces in the database.

Sequence of index entries

The sequence of the index entries can be in ascending order or descending order. The ASC and DESC keywords of the CREATE INDEX statement indicate ascending and descending order. ASC is the default.

Indexes on tables with large objects

You can use indexes on tables with LOBs the same way that you use them on other tables, but consider the following facts:

- A LOB column cannot be a column in an index.
- An auxiliary table can have only one index. (An auxiliary table, which you create by using the SQL CREATE AUXILIARY TABLE statement, holds the data for a column that a base table defines.
- Indexes on auxiliary tables are different than indexes on base tables.

Creation of an index

If the table that you are indexing is empty, DB2 creates the index. However, DB2 does not actually create index entries until the table is loaded or rows are inserted. If the table is not empty, you can choose to have DB2 build the index when the CREATE INDEX statement is executed. Alternatively, you can defer the index build until later. Optimally, you should create the indexes on a table before loading the table. However, if your table already has data, choosing the DEFER option is preferred; you can build the index later by using the REBUILD INDEX utility.

Copies of an index

If your index is fairly large and needs the benefit of high availability, consider copying it for faster recovery. Specify the COPY YES clause on a CREATE INDEX or ALTER INDEX statement to allow the indexes to be copied. DB2 can then track the ranges of log records to apply during recovery, after the image copy of the index is restored. (The alternative to copying the index is to use the REBUILD INDEX utility, which might increase the amount of time that the index is unavailable to applications.)

Deferred allocation of index space data sets

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

Example: You might be installing a software program that requires creation of many indexes, but your company might not need some of those indexes. You might prefer not to allocate data sets for indexes that you do not plan to use.

To defer the physical allocation of DB2-managed data sets, use the DEFINE NO clause of the CREATE INDEX statement. When you specify the DEFINE NO clause, DB2 defines the index but defers the allocation of data sets. The DB2 catalog table contains a record of the created index and an indication that the data sets are not yet allocated. DB2 allocates the data sets for the index space as needed when rows are inserted into the table on which the index is defined.

Related concepts:

Naming conventions (DB2 SQL)

Related reference:

CREATE INDEX (DB2 SQL)

How DB2 implicitly creates an index

In certain circumstances, DB2 implicitly creates the unique indexes that are used to enforce the uniqueness of the primary keys or unique keys.

These circumstances include:

- When the PRIMARY KEY or UNIQUE clause is specified in the CREATE TABLE statement and the CREATE TABLE statement is processed by the schema processor
- When the table space that contains the table is implicitly created

When a ROWID column is defined as GENERATED BY DEFAULT in the CREATE TABLE statement, and the CREATE TABLE statement is processed by SET CURRENT RULES = 'STD', or the table space that contains the table is implicitly created, DB2 implicitly creates the unique indexes used to enforce the uniqueness of the ROWID column. The privilege set must include the USE privilege of the buffer pool. Each index is created as if the following CREATE INDEX statement were issued:

```
CREATE UNIQUE INDEX xxx ON table-name (column1,...)
```

Where:

- *xxx* is the name of the index that DB2 generates.
- *table-name* is the name of the table that is specified in the CREATE TABLE statement.
- *(column1,...)* is the list of column names that were specified in the UNIQUE or PRIMARY KEY clause of the CREATE TABLE statement, or the column is a ROWID column that is defined as GENERATED BY DEFAULT.

In addition, if the table space that contains the table is implicitly created, DB2 will check the DEFINE DATA SET subsystem parameter to determine whether to define the underlying data set for the index space of the implicitly created index on the base table.

If DEFINE DATA SET is NO, the index is created as if the following CREATE INDEX statement is issued:

```
CREATE UNIQUE INDEX xxx ON table-name (column1,...) DEFINE NO
```

When you create a table and specify the *organization-clause* of the CREATE TABLE statement, DB2 implicitly creates an index for hash overflow rows. This index contains index entries for overflow rows that do not fit in the fixed hash space. If the hash space that is specified in the *organization-clause* is adequate, the hash overflow index should have no entries, or very few entries. The hash overflow index for a table in a range-partitioned universal table space will be a partitioned index. The hash overflow index for a table in a partition-by-growth table space will be a non-partitioned index.

DB2 determines how much space to allocate for the hash overflow index. Because this index will be sparsely populated, the size is relatively small compared to a normal index.

Index versions

DB2 uses index versions to maximize data availability. Index versions enable DB2 to keep track of schema changes and provides users with access to data in altered columns that are contained in 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.

DB2 creates an index version each time you commit one of the following schema changes:

Table 8. Situations when DB2 creates an index version

When you commit this change to a schema	DB2 creates this type of corresponding index version
Use the ALTER TABLE statement to change the data type of a non-numeric column that is contained in one or more indexes.	A new index version for each index that is affected by this operation.
Use the ALTER TABLE statement to change the length of a VARCHAR column that is contained in one or more PADDED indexes.	A new index version for each index that is affected by this operation.
Use the ALTER TABLE statement to extend the length of a CHAR column in a table.	A new index version for each index that is affected by this operation.
Use the ALTER INDEX statement to add a column to an index.	One new index version; only one index is affected by this operation. The index is set to REBUILD-pending status if the column was not added to the table in the same commit operation.
Add a new column to both a table and an index in the same commit operation.	A new index version for each index that 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-length character (VARCHAR data type) or varying-length graphic (VARGRAPHIC data type) column that is contained in one or more indexes that are defined with the NOT PADDED option
- 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 that are 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.

Related tasks:

"Recycling index version numbers" on page 172

"Reorganizing indexes" on page 172

Implementing DB2 schemas

Use schemas to provide a logical classification of objects in the database.

Creating a schema by using the schema processor

Schemas provide a logical classification of objects in the database. You can use the schema processor to create a schema.

About this task

Creating a schema by using the CREATE SCHEMA statement is also supported for compliance testing.

GUPI

CREATE SCHEMA statements cannot be embedded in a host program or executed interactively. To process the CREATE SCHEMA statement, you must use the schema processor. 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.

Procedure

To create a schema:

1. Write a CREATE SCHEMA statement.
2. Use the schema processor to execute the statement.

Example

The following example shows 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
```

GUPI

Processing schema definitions

You must use the schema processor to process CREATE SCHEMA statements.

Before you begin

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.

Procedure

To process schema definitions:

1. 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.
2. Optional: 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.

Loading data into DB2 tables

You can use several methods to load data into DB2 tables.

The most common method for loading data into most of your tables is to use 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.

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.

You can also use an SQL INSERT statement to copy all or selected rows of another table, in any of the following methods:

- Using the INSERT statement in an application program
- Interactively through SPUFI
- With the command line processor

To reformat data from IMS DL/I databases and VSAM and SAM loading for the LOAD utility, use DB2 DataPropagator.

Loading data with the LOAD utility

Use the LOAD utility to load one or more tables of a table space. If you are loading a large number of rows, use the LOAD utility rather than inserting the rows by using the INSERT statement.

Before you begin

Before using the LOAD utility, make sure that you complete all of the prerequisite activities for your situation.

Procedure

To load data:

Run the LOAD utility control statement with the options that you need.

What to do next

Reset the restricted status of the table space that contains the loaded data.

Related concepts:

Before running LOAD (DB2 Utilities)

"Row format conversion for table spaces" on page 672

Related tasks:

Collecting statistics by using DB2 utilities (DB2 Performance)

Related reference:

LOAD (DB2 Utilities)

How the LOAD utility loads DB2 tables

Use the LOAD utility to load one or more persistent tables of a table space, or one or more partitions of a table space. The LOAD utility 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, you can use the LOAD utility to do the following:

- 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

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.

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 cross-loader function.

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 for other tables in the table space that is not part of the table that is being loaded is unavailable to other application programs during the load operation with the exception of LOAD SHRLEVEL CHANGE. For partitioned table spaces, data that is in the table space that is being loaded is also 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.
- With XML columns, if there is an implicitly created DOCID column in the table, it is created with the GENERATED ALWAYS attribute.

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.

XML columns

You can load XML documents from input records if the total input record length is less than 32 KB. For input record length greater than 32 KB, you must load the data from a separate file. (You can also use a separate file if the input record length is less than 32 KB.)

When the XML data is to be loaded from the input record, specify XML as the input field type. The target column must be an XML column. The LOAD utility treats XML columns as varying-length data when loading XML directly from input records and expects a two-byte length field preceding the actual XML value.

The XML tables are loaded when the base table is loaded. You cannot specify the name of the auxiliary XML table to load.

XML documents must be well formed in order to be loaded.

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.

Replacement or addition of 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 table 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).

Loading data by using the INSERT statement

You can load data into tables is by using the INSERT statement.

Procedure

To load data into tables:

Issue an INSERT statement, and insert single or multiple rows.

What to do next

You can issue the statement interactively or embed it in an application program.

Related tasks:

"Inserting multiple rows" on page 82

"Inserting a single row"

"Registering an existing table as a materialized query table" on page 155

"Changing the logging attribute" on page 118

Related reference:

INSERT (DB2 SQL)

Inserting a single row

The simplest form of the INSERT statement 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.

Procedure

GUPI

To insert a single row:

1. Issue an INSERT INTO statement.
2. Specify the table name, the columns into which the data is to be inserted, and the data itself.

Example

For 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 DSN8D91A.DSN8S91D;
```

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

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

What to do next

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.

GUPI

Inserting multiple rows

You can use a form of INSERT that copies rows from another table.

Procedure

GUPI

To add multiple rows to a table:

1. Issue an INSERT INTO statement. For example, the following statement loads TEMPDEPT with data from the department table about all departments that report to department D01.

   ```
   INSERT INTO SMITH.TEMPDEPT
     SELECT DEPTNO,DEPTNAME,MGRNO,ADMRDEPT
     FROM DSN8910.DEPT
     WHERE ADMRDEPT='D01';
   ```

2. Optional: Embed the INSERT statement in an application program to insert multiple rows into a table from the values that are provided in host variable arrays.

   a. 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.

GUPI

Implications of using an INSERT statement to load tables

If you plan to use the INSERT statement to load tables, you should consider some of the implications.

- 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.
- Set the NOT LOGGED attribute for table spaces when large volumes of data are being inserted with parallel INSERT processes. If the data in the table space is lost or damaged, it can be reinserted from its original source.
- 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.
- 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).
  - 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)

Loading data from DL/I

You might want 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.

Procedure

To convert the data:

Use the DataRefresher™ licensed program.

Related concepts:

"Tools for moving DB2 data" on page 193

Implementing DB2 stored procedures

You might choose to use stored procedures for code that is used repeatedly. Other benefits of using stored procedures include reducing network traffic, returning result sets to an application, or allowing access to data without granting the privileges to the applications.

About this task

Introductory concepts:

Procedures (Introduction to DB2 for z/OS)

A *stored procedure* is a compiled program that can execute SQL statements and is stored at a local or remote DB2 server. You can invoke a stored procedure from an application program or from the command line processor. A single call to a stored procedure from a client application can access the database at the server several times.

A typical stored procedure contains two or more SQL statements and some manipulative or logical processing in a host language or SQL procedure statements. You can call stored procedures from other applications or from the command line. DB2 provides some stored procedures, but you can also create your own.

Related tasks:

Creating an external stored procedure (DB2 Application programming and SQL)

Creating an external SQL procedure (DB2 Application programming and SQL)

Creating a native SQL procedure (DB2 Application programming and SQL)

Related reference:

Procedures that are supplied with DB2 (DB2 SQL)

Creating stored procedures

The process that you follow to create a stored procedure depends on the type of stored procedure that you want to create.

About this task

You can create one of the following types of stored procedures:

External stored procedures
A procedure that is written in a host language.

External SQL procedures
A procedure whose body is written entirely in SQL, but is created, implemented, and executed like other external stored procedures.

Native SQL procedures
A procedure with a procedural body that is written entirely in SQL and is created by issuing a single SQL statement, CREATE PROCEDURE. Native SQL procedures do not have an associated external application program.

Procedure

GUPI

To create a stored procedure:

1. Set up the stored procedure environment. This step is required for creating external SQL procedures and external stored procedures. For native SQL procedures this step is not required, unless the native SQL procedure calls an external stored procedure or a user-defined function. For more information about setting up the stored procedure environment, see Installation step 19: Configure DB2 for running stored procedures and user-defined functions or Migration step 25: Configure DB2 for running stored procedures and user-defined functions in DB2 Installation Guide.
2. Create the stored procedure by following the process for the type of stored procedure that you want to create. When you create a stored procedure, you use the CREATE PROCEDURE statement to register a stored procedure with a database server.

GUPI

Related tasks:

Creating an external stored procedure (DB2 Application programming and SQL)

Creating an external SQL procedure (DB2 Application programming and SQL)

Creating a native SQL procedure (DB2 Application programming and SQL)

Related reference:

CREATE PROCEDURE (DB2 SQL)

Related information:

IBM Data Studio (DB2 for z/OS Stored Procedures: Through the CALL and Beyond)

Dropping stored procedures

Use the DROP statement to drop all versions of a stored procedure and its associated packages at the current server.

About this task

You might want to drop a stored procedure for a number of reasons. You might not use a particular stored procedure any longer, or you might want to drop a stored procedure and re-create it. For example, you might want to migrate an external SQL procedure to a native SQL procedure, because native SQL procedures typically perform better and have more functionality than external SQL procedures.

Procedure

GUPI

To drop a stored procedure:

Issue the DROP PROCEDURE statement, and specify the name of the stored procedure that you want to drop.

Example

For example, to drop the stored procedure MYPROC in schema SYSPROC, issue the following statement:

```
DROP PROCEDURE SYSPROC.MYPROC;
```

GUPI

Related tasks:

Migrating an external SQL procedure to a native SQL procedure (DB2 Application programming and SQL)

Related reference:

DROP (DB2 SQL)

Implementing DB2 user-defined functions

In contrast to built-in DB2 functions, you can create and implement your own external, sourced, or SQL functions.

Creating user-defined functions

The CREATE FUNCTION statement registers a user-defined function with a database server.

Procedure

To create a user-defined function:

Issue the CREATE FUNCTION statement, and specify the type of function that you want to create. You can specify the following types of functions:

- External scalar
- External table
- Sourced
- SQL scalar

Related concepts:

User-defined functions (DB2 SQL)

Related tasks:

"Altering user-defined functions" on page 183

Defining a user-defined function (DB2 Application programming and SQL)

Related reference:

CREATE FUNCTION (DB2 SQL)

Deleting user-defined functions

Use the DROP statement to delete a user-defined function at the current server.

Procedure

GUPI

To delete a user-defined function:

1. Issue the DROP statement.
2. Specify FUNCTION or SPECIFIC FUNCTION.

Example

For example, drop the user-defined function ATOMIC_WEIGHT from the schema CHEM:

```
DROP FUNCTION CHEM.ATOMIC_WEIGHT;
```

GUPI

Related concepts:

User-defined functions (DB2 SQL)

Related reference:

DROP (DB2 SQL)

Estimating disk storage for user data

To properly estimate the amount of disk storage that is necessary to store your data, you need to consider several factors.

About this task

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.

Procedure

To estimate disk storage for user data:

Ensure that the statistics history is current by using the MODIFY STATISTICS utility to delete outdated statistical data from the catalog history tables.

Related concepts:

"General approach to estimating storage" on page 88

Related tasks:

Collecting history statistics (DB2 Performance)

Collecting statistics history (DB2 Utilities)

Improving disk storage (DB2 Performance)

Related reference:

MODIFY STATISTICS (DB2 Utilities)

General approach to estimating storage

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 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 particular 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.

Whether you use extended address volumes (EAV) is also a factor in estimating storage. Although, the EAV factor is not a multiplier, you need to add 10 cylinders for each object in the cylinder-managed space of an EAV. DB2 data sets might take more space or grow faster on EAV compared to non-extended address volumes. The reason is that the allocation unit in the extended addressing space (EAS) of EAV is a multiple of 21 cylinders, and every allocation is rounded up to this multiple. If you use EAV, the data set space estimation for an installation must take this factor into account. The effect is more pronounced for smaller data sets.

For more accuracy, you can calculate M as the product of the following factors:

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. 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. 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. The following table shows the track size, number of pages per track, and the value of the unusable-space factor for several different device types.

Table 9. 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

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.

The following table 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 10. 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.

Also, you need to add the EAV factor.

Related tasks:

"Estimating disk storage for user data" on page 87

Choosing data page sizes for LOB data (DB2 Performance)

Related information:

"Calculating the space required for a table" on page 90

"Calculating the space required for an index" on page 93

Calculating the space required for a table

The following information provides details about how to calculate the space that is required for a table. Space allocation parameters are specified in kilobytes (KB).

- "Calculations for record lengths and pages"
- "Estimating storage for LOBs" on page 91
- "Estimating storage when using the LOAD utility" on page 92

Related tasks:

Compressing your data (DB2 Performance)

Calculations for record lengths and pages

In DB2, a record is the storage representation of a row. An important factor in estimating the required amount of space for a table is the size of the records.

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 and XML data, which is not stored in the base row or included in the total length of the row), each record has:
- A 6-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
- Six bytes of descriptive information in the base table for each XML column. Or, if the column can contain multiple versions of an XML document, then 14 bytes of descriptive information for each XML column.

GUPI

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.

GUPI

Every data page has:
- A 22-byte header
- A 2-byte directory entry for each record that is stored in the page

The maximum space available to store records in a 4 KB page is 4056 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 1927 bytes (approximately $0.95 \times 4056 \times 0.5$).

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 eight-byte overhead for each record, the sum of column lengths cannot be greater than 4048 bytes (4056 minus the eight-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.

As shown in the following table, the maximum record size for each page size depends on the size of the table space, whether the table is enabled for hash access, and whether you specified the EDITPROC clause.

Table 11. Maximum record size (in bytes)

Table type	4 KB page	8 KB page	16 KB page	32 KB page
Non-hash table	4056	8138	16330	32714
Non-hash table with EDITPROC	4046	8128	16320	32704
Hash table (hash home page)	3817	7899	16091	32475
Hash table with EDITPROC (hash home page)	3807	7889	16081	32465

GUPI

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.

GUPI

Related concepts:

XML versions (DB2 Programming for XML)

"General approach to estimating storage" on page 88

Estimating storage for LOBs

Before calculating the storage that is required for LOB table spaces, you must understand the size restrictions for large object (LOBs) data types.

About this task

Tables with 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.

Procedure

To estimate the storage required for LOB table spaces, complete the following steps:

1. Begin with your estimates from other table spaces
2. Round the figure up to the next page size
3. Multiply the figure by 1.1

What to do next

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.

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.

When a LOB table space grows to its maximum size, no more data can be inserted into the table space or its associated base table.

Estimating storage when using the LOAD utility

You must complete several calculations to estimate the storage that is required for a table to be loaded by the LOAD utility.

About this task

For a table to be loaded by the LOAD utility, assume the following values:

- 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
- 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

Procedure

To calculate the storage required when using the LOAD utility, complete the following steps:

1. Calculate the usable page size.

   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. Calculate the records per page.

   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. Calculate the pages used.

   Pages used is 2+CEILING(*number of records / records per page*).

4. Calculate the total pages used.

   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*.

   If you are using data compression, you need additional pages to store the dictionary.

5. Estimate the 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*).

Example

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.

Related tasks:

Calculating the space that is required for a dictionary (DB2 Performance)

Related reference:

LOAD (DB2 Utilities)

Calculating the space required for an index

The following information provides details about how to calculate the space that is required for an index.

- "Levels of index pages" on page 94
- "Estimating storage from the number of index pages" on page 94

Space allocation parameters are specified in kilobytes (KB).

Levels of index pages

Indexes can have more than one level of pages. An index that is built by the LOAD utility requires a certain amount of storage, which depends on the number of index pages at all levels. The number of index pages at all levels depends on whether the index is unique.

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. The following figure shows, in schematic form, a typical index.

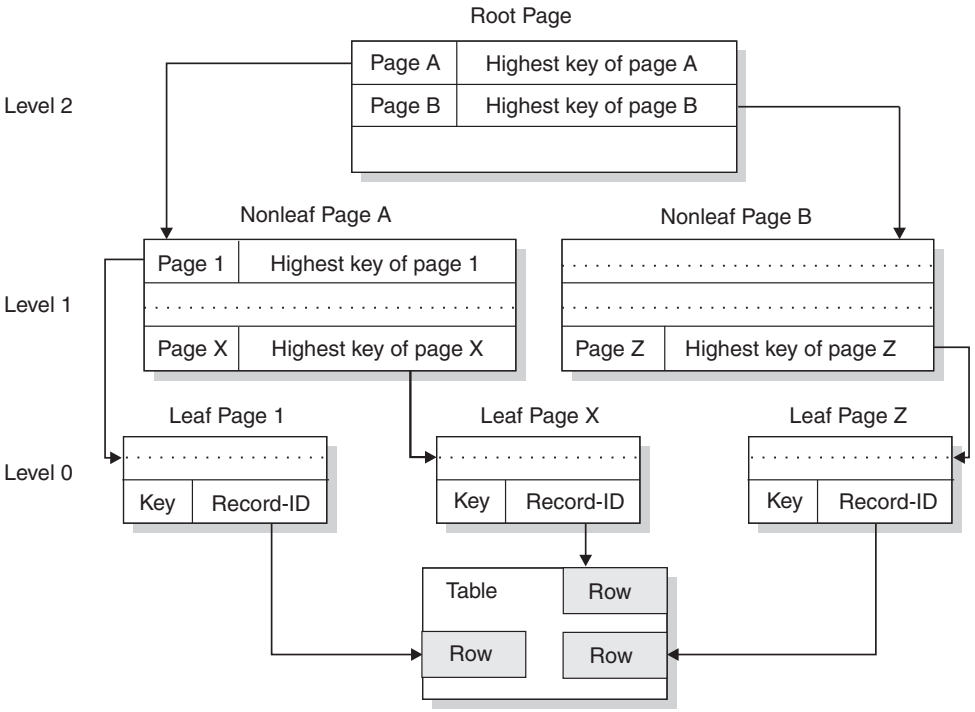


Figure 13. Sample index structure and pointers (three-level index)

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 non-null index entries cannot take advantage of the *highest-value* split.

Estimating storage from the number of index pages

Before you run a LOAD utility job to load an index, estimate the future storage requirements of the index.

About this task

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 be able to fit on an index page.

Important: Space allocation parameters are specified in kilobytes.

In the following calculations, assume the following:

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 The record identifier (RID) length. Let *r* = 4 for indexes on non-large table spaces and *r* = 5 for indexes on large table spaces (defined with DSSIZE greater than or equal to 4 GB) and on auxiliary tables.

S The value of the page size minus the length of the page header and page tail.

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.

Procedure

To estimate index storage size, complete the following calculations:

1. Calculate the pages for a unique index.
   a. Calculate the *total leaf pages*
      1) Calculate the *space per key*

         space per key is approximately $k + r + 3$
      2) Calculate the *usable space per page*

usable space per page is approximately FLOOR((100 - *f*)× S / 100)

3) Calculate the *entries per page*

   entries per page is approximately FLOOR(*usable space per page / space per key*)

4) Calculate the ***total leaf pages***

   total leaf pages is approximately CEILING(*number of table rows / entries per page*)

b. Calculate the *total nonleaf pages*

1) Calculate the *space per key*

   space per key is approximately *k* + 7

2) Calculate the *usable space per page*

   usable space per page is approximately FLOOR(MAX(90, (100 - *f*))× S /100)

3) Calculate the *entries per page*

   entries per page is approximately FLOOR(*usable space per page / space per key*)

4) Calculate the *minimum child pages*

   minimum child pages is approximately MAX(2, (entries per page + 1))

5) Calculate the *level 2 pages*

   level 2 pages is approximately CEILING(*total leaf pages / minimum child pages*)

6) Calculate the *level 3 pages*

   level 3 pages is approximately CEILING(*level 2 pages / minimum child pages*)

7) Calculate the *level x pages*

   level x pages is approximately CEILING(*previous level pages / minimum child pages*)

8) Calculate the ***total nonleaf pages***

   total nonleaf pages is approximately (*level 2 pages + level 3 pages* + ...+ *level x pages* until the number of *level x pages* = 1)

2. Calculate the pages for a nonunique index.

a. Calculate the *total leaf pages*

1) Calculate the *space per key*

   space per key is approximately 4 + *k* + (n × *(r+1)*)

2) Calculate the *usable space per page*

   usable space per page is approximately FLOOR((100 - *f*)× S / 100)

3) Calculate the *key entries per page*

   key entries per page is approximately *n*× (*usable space per page / space per key*)

4) Calculate the *remaining space per page*

   remaining space per page is approximately *usable space per page* - (*key entries per page* / n)×*space per key*

5) Calculate the *data records per partial entry*

   data records per partial entry is approximately FLOOR((*remaining space per page* - (*k* + 4)) / 5)

6) Calculate the *partial entries per page*

   partial entries per page is approximately (*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) Calculate the *entries per page*

entries per page is approximately MAX(1, (*key entries per page + partial entries per page*))

8) Calculate the ***total leaf pages***

total leaf pages is approximately CEILING(*number of table rows / entries per page*)

b. Calculate the *total nonleaf pages*

1) Calculate the *space per key*

space per key is approximately $k + r + 7$

2) Calculate the *usable space per page*

usable space per page is approximately FLOOR (MAX(90, (100- *f*))× S / 100)

3) Calculate the *entries per page*

entries per page is approximately FLOOR((*usable space per page / space per key*)

4) Calculate the *minimum child pages*

minimum child pages is approximately MAX(2, (entries per page + 1))

5) Calculate the *level 2 pages*

level 2 pages is approximately CEILING(*total leaf pages / minimum child pages*)

6) Calculate the *level 3 pages*

level 3 pages is approximately CEILING(*level 2 pages / minimum child pages*)

7) Calculate the *level x pages*

level x pages is approximately CEILING(*previous level pages / minimum child pages*)

8) Calculate the ***total nonleaf pages***

total nonleaf pages is approximately (*level 2 pages + level 3 pages + ...+ level x pages* until $x = 1$)

3. Calculate the pages for an index that is not compressed.

a. Calculate the *usable space per leaf page*:

usable space per leaf page is approximately FLOOR((100 - *f*) × S / 100)

The page size can be 4096 bytes (4 KB), 8192 bytes (8 KB), 16384 bytes (16 KB), or 32768 bytes (32 KB). The length of the page header is 62 bytes, and the length of the page tail is 2 bytes.

b. Calculate the *usable space per nonleaf page*:

usable space per nonleaf page is approximately FLOOR (MAX (90, (100 - *f*)) × S / 100)

The page size can be 4096 bytes (4 KB), 8192 bytes (8 KB), 16384 bytes (16 KB), or 32768 bytes (32 KB). The length of the page header is 48 bytes, and the length of the page tail is 2 bytes.

c. Calculate the *usable space per space map*:

usable space per space map is approximately CEILING ((*tree pages + free pages*) / S), where S equals (*page size − header length − tail length*) * 2 − 1.

The page size can be 4096 bytes (4 KB), 8192 bytes (8 KB), 16384 bytes (16 KB), or 32768 bytes (32 KB). The length of the page header is 28 bytes, and the length of the page tail is 2 bytes.

4. Calculate the pages for a compressed index.

a. Calculate the *usable space per leaf page*:

usable space per leaf page is approximately FLOOR((100 - *f*) × S / 100)

The page size can be 4096 bytes (4 KB), 8192 bytes (8 KB), 16384 bytes (16 KB), or 32768 bytes (32 KB). The length of the page header is 66 bytes, and the length of the page tail is 2 bytes.

b. Calculate the *usable space per nonleaf page*:

usable space per nonleaf page is approximately FLOOR (MAX (90, (100 - *f*)) × S / 100)

The page size is 4096 bytes for 4 KB, 8 KB, 16 KB, and 32 KB page sizes. The length of the page header is 48 bytes, and the length of the page tail is 2 bytes.

c. Calculate the *usable space per space map*:

usable space per space map is approximately CEILING ((tree pages + free pages) / S), where S equals (page size – header length – tail length) * 2 – 1.

The page size is 4096 bytes for 4 KB, 8 KB, 16 KB, and 32 KB page sizes. The length of the page header is 28 bytes, and the length of the page tail is 2 bytes.

5. Calculate the total space requirement by estimating the number of kilobytes required for an index built by the LOAD utility.

a. Calculate the *free pages*

free pages is approximately FLOOR(*total leaf pages* / *p*), or 0 if *p* = 0

b. Calculate the *space map pages*

space map pages is approximately CEILING((*tree pages* + *free pages*) / S)

c. Calculate the *tree pages*

tree pages is approximately MAX(2, (*total leaf pages* + *total nonleaf pages*))

d. Calculate the *total index pages*

total index pages is approximately MAX(4, (1 + *tree pages* + *free pages* + *space map pages*))

e. Calculate the ***total space requirement***

total space requirement is approximately 4× (*total index pages* + 2)

Example

In the following example of the entire calculation, assume that an index is defined with these characteristics:

- The index 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 page size is 4 KB.

Table 12. Sample of the total space requirement for a unique index

Quantity	Calculation	Result
Length of key	*k*	10
Average number of duplicate keys	*n*	1
PCTFREE	*f*	5
FREEPAGE	*p*	4

Table 12. Sample of the total space requirement for a unique index (continued)

Quantity	Calculation	Result
Calculate total leaf pages		
Space per key	*k* + 7	17
Usable space per page	FLOOR((100 - *f*) × 4032/100)	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
Calculate total nonleaf pages		
Space per key	*k* + 7	17
Usable space per page	FLOOR(MAX(90, (100 - *f*)) × 4046/100)	3843
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

Chapter 3. Altering your database design

After using a relational database for a while, you might want to change some aspects of its design.

To alter the database design you need to change the definitions of DB2 objects.

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.

Attention: 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 packages that reference deleted objects are marked invalid by DB2.

Related concepts:

Chapter 2, "Implementing your database design," on page 21

Related reference:

Statements (DB2 SQL)

DROP (DB2 SQL)

COMMIT (DB2 SQL)

Using the catalog in database design

Retrieving information from the catalog by using SQL statements, can be helpful in designing your relational database.

GUPI

DB2 SQL Reference lists all the DB2 catalog tables and the information stored in them.

The information in the catalog is vital to normal DB2 operation. You can *retrieve* catalog information, but *changing* it can have serious consequences. Therefore you cannot execute insert or delete operations that affect the catalog, and only a limited number of columns exist that you can update. Exceptions to these restrictions are the SYSIBM.SYSSTRINGS, SYSIBM.SYSCOLDIST, and SYSIBM.SYSCOLDISTSTATS catalog tables, into which you can insert rows and proceed to update and delete rows.

To retrieve information from the catalog, you need at least the SELECT privilege on the appropriate catalog tables.

Note: Some catalog queries can result in long table space scans.

© Copyright IBM Corp. 1982, 2016

GUPI

Retrieving catalog information about DB2 storage groups

The SYSIBM.SYSSTOGROUP and SYSIBM.SYSVOLUMES tables contain information about DB2 storage groups and the volumes in those storage groups.

Procedure

GUPI

To obtain information about DB2 storage groups and the volumes in those storage groups:

Query the SYSIBM.SYSSTOGROUP and SYSIBM.SYSVOLUMES tables. The following query shows what volumes are in a DB2 storage group, how much space is used, and when that space was last calculated.

```
SELECT SGNAME,VOLID,SPACE,SPCDATE
  FROM SYSIBM.SYSVOLUMES,SYSIBM.SYSSTOGROUP
  WHERE SGNAME=NAME
  ORDER BY SGNAME;
```

GUPI

Related reference:

SYSIBM.SYSSTOGROUP table (DB2 SQL)

SYSIBM.SYSVOLUMES table (DB2 SQL)

Retrieving catalog information about a table

The SYSIBM.SYSTABLES table contains information about every table, view, and alias in your DB2 system.

About this task

GUPI

The SYSIBM.SYSTABLES table contains a row for every table, view, and alias in your DB2 system. Each row tells you whether the object is a table, a view, or an alias, its name, who created it, what database it belongs to, what table space it belongs to, and other information. The SYSTABLES table also has a REMARKS column in which you can store your own information about the table in question.

Procedure

To retrieve catalog information about a table:

Query the SYSIBM.SYSTABLES table. The following example query displays all the information for the project activity sample table:

```
SELECT *
  FROM SYSIBM.SYSTABLES
  WHERE NAME = 'PROJACT'
  AND CREATOR = 'DSN8A10';
```

GUPI

Related concepts:

Adding and retrieving comments

Related reference:

SYSIBM.SYSTABLES table (DB2 SQL)

Retrieving catalog information about partition order

The LOGICAL_PART column in the SYSIBM.SYSTABLEPART table contains information for key order or logical partition order.

Procedure

GUPI

To retrieve catalog information about partition order:

Query the SYSIBM.SYSTABLEPART table. The following statement displays information on partition order in ascending limit value order:

```
SELECT LIMITKEY, PARTITION
  FROM SYSIBM.SYSTABLEPART
  ORDER BY LOGICAL_PART;
```

GUPI

Related reference:

SYSIBM.SYSTABLEPART table (DB2 SQL)

Retrieving catalog information about aliases

Query SYSIBM.SYSTABLES to obtain information about aliases.

About this task

GUPI

You can use the SYSIBM.SYSTABLES table to find information about aliases by referencing the following three columns:

- LOCATION contains your subsystem's location name for the remote system, if the object on which the alias is defined resides at a remote subsystem.
- TBCREATOR contains the schema table or view.
- TBNAME contains the name of the table or the view.

You can also find information about aliases by using the following user-defined functions:

- TABLE_NAME returns the name of a table, view, or undefined object found after resolving aliases for a user-specified object.
- TABLE_SCHEMA returns the schema name of a table, view, or undefined object found after resolving aliases for a user-specified object.

- TABLE_LOCATION returns the location name of a table, view, or undefined object found after resolving aliases for a user-specified object.

The NAME and CREATOR columns of the SYSTABLES table contain the name and schema of the alias, and three other columns contain the following information for aliases:
- TYPE is A.
- DBNAME is DSNDB06.
- TSNAME is SYSTSTAB.

If similar tables at different locations have names with the same second and third parts, you can retrieve the aliases for them with a query like this one:

```
SELECT LOCATION, CREATOR, NAME
  FROM SYSIBM.SYSTABLES
    WHERE TBCREATOR='DSN8A10' AND TBNAME='EMP'
      AND TYPE='A';
```

GUPI

Related reference:

SYSIBM.SYSTABLES table (DB2 SQL)

TABLE_NAME (DB2 SQL)

TABLE_SCHEMA (DB2 SQL)

TABLE_LOCATION (DB2 SQL)

Retrieving catalog information about columns

The SYSIBM.SYSCOLUMNS table has one row for each column of every table and view.

Procedure

GUPI

To obtain information about the columns of a table or view:

Query the SYSIBM.SYSCOLUMNS table.
The following statement retrieves information about columns in the sample department table:

```
SELECT NAME, TBNAME, COLTYPE, LENGTH, NULLS, DEFAULT
  FROM SYSIBM.SYSCOLUMNS
  WHERE TBNAME='DEPT'
  AND TBCREATOR = 'DSN8A10';
```

The result is shown below; for each column, the following information about each column is given:
- The column name
- The name of the table that contains it
- Its data type
- Its length attribute. For LOB columns, the LENGTH column shows the length of the pointer to the LOB.
- Whether it allows nulls
- Whether it allows default values

```
NAME      TBNAME  COLTYPE         LENGTH  NULLS  DEFAULT
DEPTNO    DEPT    CHAR               3     N       N
DEPTNAME  DEPT    VARCHAR           36     N       N
MGRNO     DEPT    CHAR               6     Y       N
ADMRDEPT  DEPT    CHAR               3     N       N
```

GUPI

Related tasks:

Retrieving catalog information about LOBs

Related reference:

SYSIBM.SYSCOLUMNS table (DB2 SQL)

Retrieving catalog information about indexes

The SYSIBM.SYSINDEXES table contains a row for every index, including indexes on catalog tables.

Procedure

GUPI

To obtain information about indexes:

Query the SYSIBM.SYSINDEXES table. For example, to retrieve a row about an index named XEMPL2:

```
SELECT *
  FROM SYSIBM.SYSINDEXES
  WHERE NAME = 'XEMPL2'
  AND CREATOR = 'DSN8A10';
```

A table can have more than one index. To display information about all the indexes of a table:

```
SELECT *
  FROM SYSIBM.SYSINDEXES
  WHERE TBNAME = 'EMP'
  AND TBCREATOR = 'DSN8A10';
```

GUPI

Related reference:

SYSIBM.SYSINDEXES table (DB2 SQL)

Retrieving catalog information about views

For every view you create, DB2 stores descriptive information in several catalog tables. Query these catalog tables to obtain information about views in your database.

About this task

GUPI

The following actions occur in the catalog after the execution of CREATE VIEW:

- A row is inserted into the SYSIBM.SYSTABLES table.
- A row is inserted into the SYSIBM.SYSTABAUTH table to record the owner's privileges on the view.
- For each column of the view, a row is inserted into the SYSIBM.SYSCOLUMNS table.
- One or more rows are inserted into the SYSIBM.SYSVIEWS table to record the text of the CREATE VIEW statement.
- For each table or view on which the view is dependent, a row is inserted into the SYSIBM.SYSVIEWDEP table to record the dependency.

GUPI

Procedure

To obtain information about views:

Query one or more catalog tables.

Related reference:

CREATE VIEW (DB2 SQL)

SYSIBM.SYSTABLES table (DB2 SQL)

SYSIBM.SYSTABAUTH table (DB2 SQL)

SYSIBM.SYSCOLUMNS table (DB2 SQL)

SYSIBM.SYSVIEWS table (DB2 SQL)

SYSIBM.SYSVIEWDEP table (DB2 SQL)

Retrieving catalog information about authorizations

The SYSIBM.SYSTABAUTH table contains information about who can access your data.

Procedure

To obtain information about who can access your data:

GUPI

Query the SYSIBM.SYSTABAUTH table. The following query retrieves the names of all users who have been granted access to the DSN8A10.DEPT table.

```
SELECT GRANTEE
  FROM SYSIBM.SYSTABAUTH
  WHERE TTNAME = 'DEPT'
    AND GRANTEETYPE <> 'P'
    AND TCREATOR = 'DSN8A10';
```

GRANTEE is the name of the column that contains authorization IDs for users of tables. The TTNAME and TCREATOR columns specify the DSN8A10.DEPT table. The clause GRANTEETYPE <> 'P' ensures that you retrieve the names only of users (not application plans or packages) that have authority to access the table.

GUPI

Related reference:

SYSIBM.SYSTABAUTH table (DB2 SQL)

Retrieving catalog information about primary keys

The SYSIBM.SYSCOLUMNS table identifies columns of a primary key in column KEYSEQ; a nonzero value indicates the place of a column in the primary key.

Procedure

GUPI

To obtain catalog information about primary keys:

Query the SYSIBM.SYSCOLUMNS table. To retrieve the creator, database, and names of the columns in the primary key of the sample project activity table using SQL statements, execute:

```
SELECT TBCREATOR, TBNAME, NAME, KEYSEQ
  FROM SYSIBM.SYSCOLUMNS
  WHERE TBCREATOR = 'DSN8A10'
  AND TBNAME = 'PROJACT'
  AND KEYSEQ > 0
    ORDER BY KEYSEQ;
```

The SYSIBM.SYSINDEXES table identifies the primary index of a table by the value P in column UNIQUERULE. To find the name, creator, database, and index space of the primary index on the project activity table, execute:

```
SELECT TBCREATOR, TBNAME, NAME, CREATOR, DBNAME, INDEXSPACE
  FROM SYSIBM.SYSINDEXES
  WHERE TBCREATOR = 'DSN8A10'
  AND TBNAME = 'PROJACT'
  AND UNIQUERULE = 'P';
```

GUPI

Related reference:

SYSIBM.SYSCOLUMNS table (DB2 SQL)

SYSIBM.SYSINDEXES table (DB2 SQL)

Retrieving catalog information about foreign keys

The SYSIBM.SYSRELS and SYSIBM.SYSFOREIGNKEYS tables contain information about referential constraints and the columns of the foreign key that defines the constraint.

About this task

GUPI

The SYSIBM.SYSRELS table contains information about referential constraints, and each constraint is uniquely identified by the schema and name of the dependent table and the constraint name (RELNAME). The SYSIBM.SYSFOREIGNKEYS table contains information about the columns of the foreign key that defines the constraint.

Procedure

To obtain information about referential constraints and the columns of the foreign key that defines the constraint:

Query the SYSIBM.SYSRELS table or the SYSIBM.SYSFOREIGNKEYS table. To retrieve the constraint name, column names, and parent table names for every relationship in which the project table is a dependent, execute:

```
SELECT A.CREATOR, A.TBNAME, A.RELNAME, B.COLNAME, B.COLSEQ,
       A.REFTBCREATOR, A.REFTBNAME
  FROM SYSIBM.SYSRELS A, SYSIBM.SYSFOREIGNKEYS B
  WHERE A.CREATOR = 'DSN8A10'
  AND B.CREATOR = 'DSN8A10'
  AND A.TBNAME = 'PROJ'
  AND B.TBNAME = 'PROJ'
  AND A.RELNAME = B.RELNAME
    ORDER BY A.RELNAME, B.COLSEQ;
```

To find information about the foreign keys of tables to which the project table is a parent:

```
SELECT A.RELNAME, A.CREATOR, A.TBNAME, B.COLNAME, B.COLNO
  FROM SYSIBM.SYSRELS A, SYSIBM.SYSFOREIGNKEYS B
  WHERE A.REFTBCREATOR = 'DSN8A10'
  AND A.REFTBNAME = 'PROJ'
  AND A.RELNAME = B.RELNAME
    ORDER BY A.RELNAME, B.COLNO;
```

GUPI

Related reference:

SYSIBM.SYSRELS table (DB2 SQL)

SYSIBM.SYSFOREIGNKEYS table (DB2 SQL)

Retrieving catalog information about check pending

The SYSIBM.SYSTABLESPACE table contains information about table spaces that are in check-pending status.

About this task

GUPI

The SYSIBM.SYSTABLESPACE table indicates that a table space is in check-pending status by a value in column STATUS: P if the entire table space has that status, S if the status has a scope of less than the entire space.

Procedure

To obtain information about table spaces that are in check-pending status:

Query the SYSIBM.SYSTABLESPACE table. To list all table spaces whose use is restricted for *any* reason, issue this command:

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

To retrieve the names of table spaces in check-pending status only, with the names of the tables they contain, execute:

```
SELECT A.DBNAME, A.NAME, B.CREATOR, B.NAME
  FROM SYSIBM.SYSTABLESPACE A, SYSIBM.SYSTABLES B
  WHERE A.DBNAME = B.DBNAME
  AND A.NAME = B.TSNAME
  AND (A.STATUS = 'P' OR A.STATUS = 'S')
    ORDER BY 1, 2, 3, 4;
```

GUPI

Related reference:

SYSIBM.SYSTABLESPACE table (DB2 SQL)

Retrieving catalog information about check constraints

The SYSIBM.SYSCHECKS and SYSIBM.SYSCHECKDEP tables contain information about check constraints.

About this task

GUPI

Information about check constraints is stored in the DB2 catalog in:

- SYSIBM.SYSCHECKS, which contains one row for each check constraint defined on a table
- SYSIBM.SYSCHECKDEP, which contains one row for each reference to a column in a check constraint

Procedure

To retrieve catalog information about check constraints:

Query the SYSIBM.SYSCHECKS and SYSIBM.SYSCHECKDEP tables. The following query shows all check constraints on all tables named SIMPDEPT and SIMPEMPL in order by column name within table schema. It shows the name, authorization ID of the creator, and text for each constraint. A constraint that uses more than one column name appears more than once in the result.

```
CREATE TABLE SIMPDEPT
  (DEPTNO   CHAR(3) NOT NULL,
   DEPTNAME VARCHAR(12) CONSTRAINT CC1 CHECK (DEPTNAME IS NOT NULL),
   MGRNO    CHAR(6),
   MGRNAME  CHAR(6));
```

```
SELECT A.TBOWNER, A.TBNAME, B.COLNAME,
A.CHECKNAME, A.CREATOR, A.CHECKCONDITION
FROM SYSIBM.SYSCHECKS A, SYSIBM.SYSCHECKDEP B
WHERE A.TBOWNER = B.TBOWNER
  AND A.TBNAME = B.TBNAME
  AND B.TBNAME = 'SIMPDEPT'
  AND A.CHECKNAME = B.CHECKNAME
  ORDER BY TBOWNER, TBNAME, COLNAME;
```

GUPI

Related reference:

SYSIBM.SYSCHECKS table (DB2 SQL)

SYSIBM.SYSCHECKDEP table (DB2 SQL)

Retrieving catalog information about LOBs

The SYSIBM.SYSAUXRELS table contains information about the relationship between a base table and an auxiliary table.

Procedure

GUPI

To retrieve catalog information about LOBs:

Query the SYSIBM.SYSAUXRELS table. For example, this query returns information about the name of the LOB columns for the employee table and its associated auxiliary table schema and name:

```
SELECT COLNAME, PARTITION, AUXTBOWNER, AUXTBNAME
   FROM SYSIBM.SYSAUXRELS
   WHERE TBNAME = 'EMP' AND TBOWNER = 'DSN8A10';
```

Information about the length of a LOB is in the LENGTH2 column of the SYSCOLUMNS table. You can query information about the length of the column as it is returned to an application with the following query:

```
SELECT NAME, TBNAME, COLTYPE, LENGTH2, NULLS, DEFAULT
  FROM SYSIBM.SYSCOLUMNS
  WHERE TBNAME='DEPT'
  AND TBCREATOR = 'DSN8A10';
```

GUPI

Related reference:

SYSIBM.SYSAUXRELS table (DB2 SQL)

SYSIBM.SYSCOLUMNS table (DB2 SQL)

Retrieving catalog information about user-defined functions and stored procedures

The SYSIBM.SYSROUTINES table contains information about routines.

Procedure

To retrieve information about user-defined functions and stored procedures:

GUPI

Query the SYSIBM.SYSROUTINES table to obtain information about user-defined functions and stored procedures. You can use this example to find packages with stored procedures that were created prior to Version 6 and then migrated to the SYSIBM.SYSROUTINES table:

```
SELECT SCHEMA, NAME FROM SYSIBM.SYSROUTINES
   WHERE ROUTINETYPE = 'P';
```

You can use this query to retrieve information about user-defined functions:

```
SELECT SCHEME, NAME, FUNCTION_TYPE, PARM_COUNT FROM SYSIBM.SYSROUTINES
   WHERE ROUTINETYPE='F';
```

GUPI

Related tasks:

Preparing a client program that calls a remote stored procedure (DB2 Application programming and SQL)

Related reference:

SYSIBM.SYSROUTINES table (DB2 SQL)

Retrieving catalog information about triggers

The SYSIBM.SYSTRIGGERS table contains information about triggers.

Procedure

To retrieve catalog information about triggers:

GUPI

Query the SYSIBM.SYSTRIGGERS table to obtain information about the triggers defined in your databases. You can issue this query to find all the triggers defined on a particular table, their characteristics, and to determine the order they are activated in:

```
SELECT DISTINCT SCHEMA, NAME, TRIGTIME, TRIGEVENT, GRANULARITY, CREADEDTS
   FROM SYSIBM.SYSTRIGGERS
   WHERE TBNAME = 'EMP' AND TBOWNER = 'DSN8A10';
```

Issue this query to retrieve the text of a particular trigger:

```
SELECT STATEMENT, CREATEDTS
   FROM SYSIBM.SYSTRIGGERS
   WHERE SCHEMA = schema_name
      AND NAME = trigger_name
   ORDER BY CREATEDTS;
```

Issue this query to determine triggers that must be rebound because they are invalidated after objects are dropped or altered:

```
SELECT COLLID, NAME
   FROM SYSIBM.SYSPACKAGE
   WHERE TYPE = 'T'
      AND (VALID = 'N' OR OPERATIVE = 'N');
```

GUPI

Related reference:

SYSIBM.SYSTRIGGERS table (DB2 SQL)

Retrieving catalog information about sequences

The SYSIBM.SYSSEQUENCES and SYSIBM.SYSSEQUENCEAUTH tables contain information about sequences.

Procedure

GUPI

To obtain information about sequences:

Query the SYSIBM.SYSSEQUENCES or SYSIBM.SYSSEQUENCEAUTH table. To retrieve the attributes of a sequence, issue this query:

```
SELECT *
  FROM SYSIBM.SYSSEQUENCES
  WHERE NAME = 'MYSEQ' AND SCHEMA = 'USER1B';
```

Issue this query to determine the privileges that user USER1B has on sequences:

```
SELECT GRANTOR, NAME, DATEGRANTED, ALTERAUTH, USEAUTH
  FROM SYSIBM.SEQUENCEAUTH
  WHERE GRANTEE = 'USER1B';
```

GUPI

Related reference:

SYSIBM.SYSSEQUENCES table (DB2 SQL)

SYSIBM.SYSSEQUENCEAUTH table (DB2 SQL)

Adding and retrieving comments

After you create an object, you can provide explanatory information about it for future reference. For example, you can provide information about the purpose of the object, who uses it, and anything unusual about it.

GUPI

You can create comments about tables, views, indexes, aliases, packages, plans, distinct types, triggers, stored procedures, and user-defined functions. You can store a comment about the table or the view as a whole, and you can also include a comment for *each column*. A comment must not exceed 762 bytes.

A comment is especially useful if your names do not clearly indicate the contents of columns or tables. In that case, use a comment to describe the specific contents of the column or table.

Below are two examples of COMMENT:

```
COMMENT ON TABLE DSN8A10.EMP IS
  'Employee table. Each row in this table represents one
   employee of the company.';
COMMENT ON COLUMN DSN8A10.PROJ.PRSTDATE IS
  'Estimated project start date. The format is DATE.';
```

After you execute a COMMENT statement, your comments are stored in the REMARKS column of SYSIBM.SYSTABLES or SYSIBM.SYSCOLUMNS. (Any comment that is already present in the row is replaced by the new one.) The next two examples retrieve the comments that are added by the previous COMMENT statements.

```
SELECT REMARKS
  FROM SYSIBM.SYSTABLES
  WHERE NAME = 'EMP'
  AND CREATOR = 'DSN8A10';
SELECT REMARKS
  FROM SYSIBM.SYSCOLUMNS
  WHERE NAME = 'PRSTDATE' AND TBNAME = 'PROJ'
  AND TBCREATOR = 'DSN8A10';
```

GUPI

Verifying the accuracy of the database definition

You can use the catalog to verify the accuracy of your database definition.

Procedure

GUPI

To verify that you have created the objects in your database and check that no errors are in your CREATE statements:

Query the catalog tables to verify that your tables are in the correct table space, your table spaces are in the correct storage group, and so on.

GUPI

Related reference:

DB2 catalog tables (DB2 SQL)

Altering DB2 databases

You can alter a DB2 database by changing the description of a database at the current server.

Procedure

To change clauses that are used to create a database:

Issue the ALTER DATABASE SQL statement.

Related concepts:

DB2 databases (Introduction to DB2 for z/OS)

Related reference:

"ALTER DATABASE options"

ALTER DATABASE (DB2 SQL)

ALTER DATABASE options

You can issue the ALTER DATABASE statement to change the description of a database at the current server.

The ALTER DATABASE statement allows you to change the following options:

STOGROUP

Use this option to 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
Use this option to 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
Use this option to 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.

Related reference:

ALTER DATABASE (DB2 SQL)

Altering DB2 storage groups

To change the description of a storage group at the current server, use the ALTER STOGROUP statement.

Procedure

To alter a storage group:

1. Issue an ALTER STOGROUP statement.
2. Specify whether you want SMS to manage your DB2 storage groups, or to add or remove volumes from a storage group.

What to do next

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.

Related concepts:

"Moving DB2 data" on page 195

"Moving a DB2 data set" on page 196

Related reference:

ALTER STOGROUP (DB2 SQL)

Related information:

"Implementing DB2 storage groups" on page 22

Letting SMS manage your DB2 storage groups

Using the SMS product Data Facility Storage Management Subsystem (DFSMS) to manage your data sets can result in a reduced workload for DB2 database and storage administrators.

Procedure

GUPI To let SMS manage the storage needed for the objects that the storage group supports:

1. Issue an ALTER STOGROUP statement. You can specify SMS classes when you alter a storage group.

2. Specify ADD VOLUMES ('*') and REMOVE VOLUMES (*current-vols*) where *current-vols* is the list of the volumes that are currently assigned to the storage group. For example,

```
ALTER STOGROUP DSN8G910
  REMOVE VOLUMES (VOL1)
  ADD VOLUMES ('*');
```

Example

The following example shows how to alter a storage group to SMS-managed using the DATACLAS, MGMTCLAS, or STORCLAS keywords.

```
ALTER STOGROUP SGOS5001
   MGMTCLAS REGSMMC2
  DATACLAS REGSMDC2
   STORCLAS REGSMSC2;
```

What to do next

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. GUPI

Related tasks:

"Migrating to DFSMShsm" on page 31

Related reference:

ALTER STOGROUP (DB2 SQL)

Adding or removing volumes from a DB2 storage group

When you add or remove volumes from a storage group, all the volumes in that storage group must be of the same type.

About this task

Also, 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

The changes that you make to the volume list by using the ALTER STOGROUP statement 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 the ALTER STOGROUP statement to remove volume 22222 from storage group DSN8G910, the DB2 data on that volume remains intact. However, when a new table space is defined by using DSN8G910, volume 22222 is not available for space allocation.

Procedure

To add a new volume to a storage group:

1. Use the SYSIBM.SYSTABLEPART catalog table to determine which table spaces are associated with the storage group. GUPI For example, the following query indicates which table spaces use storage group DSN8G910:

```
SELECT TSNAME, DBNAME
  FROM SYSIBM.SYSTABLEPART
  WHERE STORNAME ='DSN8G910' AND STORTYPE = 'I';
```

GUPI

2. Make an image copy of each table space. For example, issue the statement 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:

   GUPI

   ```
   ALTER STOGROUP DSN8G910
     REMOVE VOLUMES (VOL1)
     ADD VOLUMES (VOL2);
   ```

   GUPI

   Restriction: 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 utility or the REORG utility to move the data in each table space. For example, issue RECOVER *dbname.tsname*.
7. Start the table space with the DB2 command START DATABASE (*dbname*) SPACENAM (*tsname*).

Migrating existing data sets to a solid-state drive

You can migrate DB2-managed data sets from a hard disk drive (HDD) to a solid-state drive (SSD).

About this task

For user-managed data sets, you are responsible for defining and copying the data sets to an SSD. However, whether the data sets are DB2-managed or user-managed, all volumes that can contain secondary extents should have the same drive type as the drive type of the primary extent volume. In addition, you must define all of the pieces of a multi-piece data set on volumes that have the same drive type.

To migrate DB2-managed data sets to an SSD:

Procedure

Use one of the following options.

- Use DSN1COPY to move data sets from an HDD to an SSD. The drive type is found by DB2 the next time that you open the data set.
- Issue the ALTER STOGROUP statement. For data sets that are managed by SMS, use the ALTER STOGROUP statement to change DATACLAS, MGMTCLAS, or STORCLAS to identify the new SSD volumes. For data sets that are not managed by SMS, use the ALTER STOGROUP statement to add volumes that contain SSD and drop volumes that contain HDD.

  The storage group should be homogenous and contain either all SSDs or all HDDs. The data set is moved to the new SSD volume at the next REORG after the alter operation.

Using the ALTER STOGROUP statement has an availability advantage over using the CREATE STOGROUP statement, because the ALTER TABLESPACE USING STOGROUP statement must stop the object before the alter operation can succeed. If you cannot make an existing storage group homogenous, you must use the CREATE STOGROUP statement to define the storage groups to contain SSD volumes.

Altering table spaces

Use the ALTER TABLESPACE statement to change the description of a table space at the current server.

About this task

GUPI

Pending definition changes are changes that are not immediately materialized. For detailed information about pending definition changes, how to materialize them, and related restrictions, see "Pending data definition changes" on page 173

Immediate definition changes are changes that are materialized immediately. Most immediate definition changes are restricted while pending definition changes exist for an object. For a list of such restrictions, see "Restrictions for changes to objects that have pending data definition changes" on page 180.

Procedure

To alter table spaces, use any of the following approaches:

- Issue ALTER TABLESPACE statements from embedded application programs or interactively.
- Use the ALTER TABLESPACE statements to change the table space type and attributes, or to enable or disable MEMBER CLUSTER.
- Use the MAXPARTITIONS attribute of the ALTER TABLESPACE statement to change the maximum partition size for partition-by-growth table spaces. You can use this attribute to convert a simple table space, or a single-table segmented (non-universal) table space to a partition-by-growth universal table space.
- Use the SEGSIZE attribute of the ALTER TABLESPACE statement to convert a partitioned table space to a range-partitioned universal table space.

What to do next

You can use the DROP PENDING CHANGES clause to drop all pending definition changes for the table space and for any of the objects in the table space.

Related concepts:

Types of DB2 table spaces (Introduction to DB2 for z/OS)

"Pending data definition changes" on page 173

Member affinity clustering (DB2 Data Sharing Planning and Administration)

Related tasks:

"Changing the logging attribute" on page 118

Related reference:

ALTER TABLESPACE (DB2 SQL)

SYSIBM.SYSPENDINGDDL table (DB2 SQL)

Related information:

"Implementing DB2 table spaces" on page 41

Changing the logging attribute

You can use the ALTER TABLESPACE statement to set the logging attribute of a table space.

Before you begin

Important: Limit the use of the NOT LOGGED attribute. Logging is not generally a performance bottleneck, given that in an average environment logging accounts for less than 5% of the central processing unit (CPU) utilization. Therefore, you should use the NOT LOGGED attribute only when data is being used by a single task, where the table space can be recovered if errors occur.

Procedure

To change the logging attribute of a table space:

1. Issue an ALTER TABLESPACE statement.
2. Specify the LOGGED or NOT LOGGED attribute.
   - **LOGGED:** Specifies that changes made to data in this table space are to be recorded on the log.
   - **NOT LOGGED:** Specifies that changes made to data in this table space are not to be recorded on the log. The NOT LOGGED attribute suppresses the logging of undo and redo information.

Results

The change in logging applies to all tables in this table space and also applies to all indexes on those tables, as well as associated LOB and XML table spaces.

Related tasks:

"Altering table spaces" on page 117

"Loading data by using the INSERT statement" on page 81

"Registering an existing table as a materialized query table" on page 155

Related reference:

ALTER TABLESPACE (DB2 SQL)

The NOT LOGGED attribute

The NOT LOGGED attribute for a table space indicates that changes to tables in the table space are not recorded on the log.

You should use the NOT LOGGED attribute only for situations where the data is in effect being duplicated. If the data is corrupted, you can re-create it from its original source, rather than from an image copy and the log. For example, you could use NOT LOGGED when you are inserting large volumes of data with the INSERT statement.

Restrictions: If you use the NOT LOGGED logging attribute, you can use images copies for recovery with certain restrictions.

- The logging attribute applies to all partitions of a table space. NOT LOGGED suppresses only the logging of undo and redo information; control records of the table space continue to be logged.
- You can take full and incremental SHRLEVEL REFERENCE image copies even though the table space has the NOT LOGGED attribute. You cannot take SHRLEVEL CHANGE copies because the NOT LOGGED attribute suppresses the logging of changes necessary for recovery.
- System-level backups taken with the BACKUP SYSTEM utility will contain NOT LOGGED objects, but they cannot be used for object level recovery of NOT LOGGED objects.

You can set the NOT LOGGED attribute when creating or altering table spaces.

When to use the NOT LOGGED attribute

Consider using the NOT LOGGED attribute in the following specific situations:

- For tables that summarize information in other tables, including materialized query tables, where the data can be easily re-created.
- When you are inserting large volumes of data with the INSERT statement.
- When you are using LOAD RESUME.

  To use table spaces that are not logged, when using LOAD RESUME, complete the following steps:

  1. Alter the table space to not logged before the load. Altering the logging attribute requires exclusive use of the table space.
  2. Run the LOAD utility with the RESUME option.
  3. Before normal update processing, alter the table space back to logged, and make an image copy of the table space.

  Restriction: Online LOAD RESUME against a table space that is not logged is not recoverable if the load fails. If an online load attempt fails and rollback is necessary, the not logged table space is placed in LPL RECOVER-pending status. If this happens, you must terminate the LOAD job, recover the data from a prior image copy, and restart the online LOAD RESUME.

What happens when you change the logging attribute

Altering the logging attribute of a table space from LOGGED to NOT LOGGED establishes a recoverable point for the table space. Indexes automatically inherit the logging attribute of their table spaces. For the index, the change establishes a recoverable point that can be used by the RECOVER utility. Each subsequent image copy establishes another recoverable point for the table space and its associated indexes if the image copy is taken as a set.

Altering the logging attribute of a table space from NOT LOGGED to LOGGED marks the table space as COPY-pending (a recoverable point must be established before logging resumes). The indexes on the tables in the table space that have the COPY YES attribute are unchanged.

Related concepts:

"Recovery implications for objects that are not logged" on page 450

Changing the space allocation for user-managed data sets

If the table space is supported by user-managed data sets, you must complete several steps to change the space allocation.

Procedure

To change the space allocation for user-managed data sets, complete the following steps:

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.

What to do next

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, you can drop the table space and then re-create it. These table space changes include changing SEGSIZE, changing the number of partitions, or converting a table space to a large table space.

About this task

Alternatively, you can use the ALTER TABLESPACE statement to change the table space type and attributes, such as BUFFERPOOL, DSSIZE, SEGSIZE, and MAXPARTITIONS. For more information, see "Altering table spaces" on page 117.

Procedure

To change or convert a table space by dropping the table space and then recreating it:

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 (for example, TS2), create tables TB1, TB2, TB3, ... identical to TA1, TA2, TA3, ...

   GUPI

   For example, use a statement such as:

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

   ```
   REORG TABLESPACE DSN8D91A.TS1 LOG NO SORTDATA UNLOAD EXTERNAL;
   ```

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

4. Optional: Alternatively, instead of unloading the data, you can insert the data from your old tables into the new tables by issuing 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. For example, use a statement such as:

```
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. 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.

7. Create the new table space, TS1, and grant the appropriate user privileges. You can also create a partitioned table space. For example, use a statement such as:

```
CREATE TABLESPACE TS1
   IN DSN8D91A
   USING STOGROUP DSN8G910
     PRIQTY 4000
     SECQTY 130
     ERASE NO
 NUMPARTS 95
  (PARTITION 45 USING STOGROUP DSN8G910
     PRIQTY 4000
     SECQTY 130
     COMPRESS YES,
   PARTITION 62 USING STOGROUP DSN8G910
     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 grant user privileges on those tables.
10. Issue 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.

GUPI

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 any packages that were invalidated as a result of dropping the table space.

Related concepts:

"Implications of dropping a table" on page 162

Redistributing data in partitioned table spaces

When data becomes skewed across partitions, performance can be slower. For example, data is skewed if some partitions are almost full while other partitions have a considerable amount of excess space. Performance might improve if you can redistribute the data more evenly across the partitions.

About this task

Redistributing data in partitioned table spaces is not always possible because of application dependencies or other factors. If a partition is full and redistributing the data is not practical, you might need to increase the partition size.

Procedure

To redistribute data in partitioned table spaces, use one of the following two methods:

- Changing the boundary between partitions
- Redistributing data across partitions by using REORG (DB2 Utilities)

Example

GUPI

Assume that a table space contains product data that is partitioned by product ID as follows: The first partition contains rows for product ID values 1 through 99. The second partition contains rows for values 100 to 199. The third partition contains rows for values 200 through 299. And the subsequent partitions are empty.

Suppose that after some time, because of the popularity of certain products, you want to redistribute the data across certain partitions. You want the third partition to contain values 200 through 249, the fourth partition to contain values 250 through 279, and the fifth partition to contain values 280 through 299.

To change the boundary for these partitions, issue the following statements:

```
ALTER TABLE PRODUCTS ALTER PARTITION 3
ENDING AT ('249');
ALTER TABLE PRODUCTS ALTER PARTITION 4
ENDING AT ('279');
ALTER TABLE PRODUCTS ALTER PARTITION 5
ENDING AT ('299');
```

Partitions 3, 4, and 5 are placed in REORG-pending (REORP) status. Those partitions are not available until they are reorganized.

Alternatively, instead of using ALTER TABLE statements with the ALTER PARTITION clause, you can use the REBALANCE keyword as follows:

```
REORG TABLESPACE dbname.tsname PART(3:5) REBALANCE
```

In this case, DB2 determines the appropriate limit key changes and redistributes the data accordingly. This method avoids leaving the partitions in REORP status.

GUPI

Related concepts:

"Differences between partitioning methods" on page 62

Related tasks:

"Increasing the partition size of a partitioned table space"

Related reference:

Syntax and options of the REORG TABLESPACE control statement (DB2 Utilities)

ALTER INDEX (DB2 SQL)

ALTER TABLE (DB2 SQL)

Advisory or restrictive states (DB2 Utilities)

Increasing the partition size of a partitioned table space

If a partition is full and redistributing the data across partitions is not practical, you might need to increase the partition size.

About this task

You can increase the maximum partition size of a partitioned table space to 128 GB or 256 GB. Depending on the partition size and page size, increasing the maximum size of a partition can proportionally reduce the maximum number of partitions that can be specified.

Procedure

To increase the maximum partition size of a partitioned table space:

1. If the table space uses index-based partitioning, convert it to table-based partitioning by issuing the DROP statement to drop the partitioning index.
2. If the table space is not a universal table space, convert it to one by issuing the ALTER TABLESPACE statement with the SEGSIZE option. The value of the SEGSIZE option must not be zero.
3. Issue the ALTER TABLESPACE statement with the DSSIZE option to increase the maximum partition size to 128 GB or 256 GB.
4. Issue the ALTER TABLESPACE statement with the PRIQTY and SECQTY options to modify the primary and secondary space allocation for each partition. This change allows the partition to grow to its anticipated maximum size.
5. Run the REORG TABLESPACE utility with SHRLEVEL CHANGE or SHRLEVEL REFERENCE to materialize the pending definition changes and convert the table space. When reorganizing a table space that has pending definition changes, the entire table space must be included. Therefore, you

cannot reorganize by partition. In addition, partition parallelism is disabled during the UNLOAD and RELOAD phases.

A significant amount of disk space can be required when reorganizing an entire table space. The amount of space required for the table space and indexes is approximately two times of what is already allocated. If the amount of space that is required is not available, you might need to use an alternative strategy of unloading, dropping, creating, and loading the table space. With this method, you can reorganize individual partitions in parallel and requires significantly less disk space.

For tables that have LOB and XML columns, the table spaces are independent from the base table space. You can alter and reorganize these table spaces separately. Do not run the REORG utility with AUX YES to reorganize both the base and LOB table spaces together, because in this case, the pending definition changes for the LOB table space are not materialized.

Converting a 16 TB table space to a table with larger partitions or data sets can take a significant amount of time.

Related tasks:

"Redistributing data in partitioned table spaces" on page 122

Related reference:

DROP (DB2 SQL)

ALTER TABLESPACE (DB2 SQL)

REORG TABLESPACE (DB2 Utilities)

Altering a page set to contain DB2-defined extents

After you use the RECOVER utility to run the DFSMSdss RESTORE command, you must alter the page set to contain extents that are defined by DB2.

About this task

This step is required because the DFSMSdss RESTORE command extends a data set differently than DB2.

Procedure

To alter a page set to contain extents that are defined by DB2:

1. Issue the ALTER TABLESPACE SQL statement.

   After you use the ALTER TABLESPACE statement, the new values take affect only when you use REORG or LOAD REPLACE.
2. Enlarge the primary and secondary space allocation values for DB2-managed data sets.

What to do next

Using the RECOVER utility again does not resolve the extent definition.

For user-defined data sets, define the data sets with larger primary and secondary space allocation values.

Related concepts:

"The RECOVER utility and the DFSMSdss RESTORE command" on page 33

Related reference:

ALTER TABLESPACE (DB2 SQL)

Altering DB2 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.

Procedure

GUPI

To alter a table:

Issue the ALTER TABLE statement. With the ALTER TABLE statement, you can:

- Add a new column
- Rename a column
- Change the data type of a column, with certain restrictions
- Add or drop a parent key or a foreign key
- Add or drop a table check constraint
- Add a new partition to a table space, including adding a new partition to a partition-by-growth table space, by using the ADD PARTITION clause
- Change the boundary between partitions, extend the boundary of the last partition, rotate partitions, or instruct DB2 to insert rows at the end of a table or appropriate partition
- 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
- Change the VALIDPROC clause
- Change the DATA CAPTURE clause
- Change the AUDIT clause by using the options ALL, CHANGES, or NONE
- Add or drop the restriction on dropping the table and the database and table space that contain the table
- Alter the length of a VARCHAR column using the SET DATA TYPE VARCHAR clause
- Add or drop a clone table
- Alter APPEND attributes
- Drop the default value for a column
- Activate or deactivate row-level or column-level access control for the table

  Tip: When designing row-level or column-level access control for a table, first create the row permissions or column masks to avoid multiple invalidations to packages and dynamically cached statements. After you create row permissions or column masks, use the ALTER TABLE statement to activate row-level or column-level access control for the table. If you must drop or alter a column mask, first activate row-level access control to prevent access to the table, and then drop or alter the column mask. Otherwise, the rows are accessible, but the column values inside the rows are not protected.

  If a security administrator with SECADM authority activates row-level access control before the explicit creation of the row permission database object, a default row permission is created. This default row permission blocks all access to the table, including access by the owner.

GUPI

Related concepts:

Row and column access control (Managing Security)

Related tasks:

"Altering tables to enable hash access" on page 150

Related reference:

ALTER TABLE (DB2 SQL)

Adding a column to a table

When you use the ALTER TABLE statement to add a column to a table, the table space is placed in an advisory REORG-pending (AREO*) state.

About this task

Also, the new column might become the rightmost column of the table, depending on whether you use basic row format or reordered row format.

The physical records are not actually changed until values are inserted in the new column. When you use the ALTER TABLE ADD COLUMN statement, packages are not invalidated, unless the following criteria are true:

- The data type of the new column is DATE, TIME, or TIMESTAMP.
- You specify the DEFAULT keyword.
- You do not specify a constant (that is, you use the system default value).

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 also might need to modify any program that contains a static SQL statement SELECT *, which returns the new column after the plan or package is rebound. You also must modify any INSERT statement that does not contain 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 access time is affected immediately.

Procedure

To change the records to fixed length:

1. Run the REORG utility with the COPY option 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 that you ran in step 1.
3. Create a unique index if you add a column that specifies PRIMARY KEY.

Results

Tip: Inserting values in the new column might 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.

If the new column is a short string column, you can specify a field procedure for it. If you do specify a field procedure, you cannot also specify NOT NULL.

Example

GUPI

The following example adds a column to the table DSN8910.DEPT, which contains a location code for the department. The column name is LOCATION_CODE, and its data type is CHAR (4).

```
ALTER TABLE DSN8910.DEPT
  ADD LOCATION_CODE CHAR (4);
```

GUPI

Related concepts:

"Row format conversion for table spaces" on page 672

Specifying a default value when altering a column

You can use the ALTER TABLE statement to add, change, or remove the default value for a column.

About this task

GUPI

Restrictions:

- You cannot alter a column to specify a default value if the table is referenced by a view.
- If the column is part of a unique constraint or unique index, the new default to a value should not be the same as a value that already exists in the column.
- The new default value applies only to new rows.

Procedure

To alter the default value for a column:

1. To set the default value, issue the following statement:

   ```
   ALTER TABLE table-name ALTER COLUMN column-name
   SET default-clause
   ```

   You can use this statement to add a default value for a column that does not already have one, or to change the existing default value.
2. To remove the default value without specifying a new one, issue the following statement:

   ```
   ALTER TABLE table-name ALTER COLUMN column-name
   DROP DEFAULT
   ```

Example

For example, suppose that table MYEMP is defined as follows:

```
CREATE TABLE MYEMP LIKE EMP
```

Use the following statement to assign a default value to column JOB:

```
ALTER TABLE MYEMP ALTER COLUMN JOB SET DEFAULT 'PENDING'
```

Use the following statement to drop the default value from column JOB:

```
ALTER TABLE MYEMP ALTER COLUMN JOB DROP DEFAULT
```

GUPI

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 several ways.

About this task

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.

Restriction: 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.

Procedure

To alter the data type of a column:

1. Issue an ALTER TABLE statement.
2. Specify the data type change that you would like to make. Potential changes include:
   - 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.

Results

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 retrieved 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:

GUPI

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.

GUPI

Related concepts:

"Table space versions" on page 132

Related tasks:

"Altering the attributes of an identity column" on page 160

Related reference:

ALTER TABLE (DB2 SQL)

Related information:

What happens to an index on altered columns

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

GUPI

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.

GUPI

Index inaccessibility and data availability

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

REBUILD INDEX with the SHRLEVEL CHANGE option allows read and write access to the data for most of the rebuild operation.

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 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 an RBDP state.

In the case of static BINDs, DB2 might choose an index that is in an RBDP state as the best access path. DB2 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

GUPI

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.

Whether indexes are padded by default depends on the DB2 release in which the index was created and the release in which the system was originally installed:

- Indexes that were created in a pre-Version 8 release are padded by default. In this case, the value of the PADDED column of the SYSINDEXES catalog table is blank (PADDED = ' '). The PADDED column is also blank when there are no varying length columns.
- In subsystems that were migrated from a pre-Version 8 release, 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 catalog table is YES (PADDED = 'Y').
- In subsystems that were originally installed in Version 8 new-function mode or a later DB2 release, indexes that are created with at least one varying length column are not padded by default. In this case, the PADDED column of the SYSINDEXES catalog table is set to NO (PADDED = 'N').

GUPI

Related concepts:

Indexes that are padded or not padded (Introduction to DB2 for z/OS)

Related tasks:

Saving disk space by using non-Padded indexes (DB2 Performance)

Related reference:

SYSIBM.SYSINDEXES table (DB2 SQL)

Table space versions

DB2 creates a table space version each time that you commit one or more specific schema changes by using the ALTER TABLE statement.

Versioning is always done at the table space level. The version of a table matches the table space version that it corresponds with. For example, consider that you have two tables in one table space, `Table1` and `Table2`, and that the version for both tables and the table space is 0 (zero). If `Table1` is altered, the version for `Table1` becomes `SYSTABLES.VERSION = 1`, and the table space version becomes `SYSTABLESPACE.VERSION = 1`. At this point, the version for `Table2` is still `SYSTABLESPACE.VERSION = 0`. Now, when `Table2` is altered, the version for `Table2` becomes `SYSTABLES.VERSION = 2`, which corresponds with the table space version of `SYSTABLESPACE.VERSION = 2`. The version of `Table2` skipped from 0 to 2, because `SYSTABLESPACE.VERSION = 1` was already used by `Table1`.

The following schema changes result in DB2 creating a table space version:

- Extending the length of a character (CHAR data type) or graphic (GRAPHIC data type) column
- Changing the type of a column within character data types (CHAR, VARCHAR)
- Changing the type of a column within graphic data types (GRAPHIC, VARGRAPHIC)
- Changing the type of a column within numeric data types (SMALL INTEGER, INTEGER, FLOAT, REAL, FLOAT8, DOUBLE, DECIMAL)
- Adding a column to a table
- Extending the length of a varying character (VARCHAR data type) or varying graphic (VARGRAPHIC data type) column, if the table already has a version number that is greater than 0
- Altering the maximum length of a LOB column, if the table already has a version number that is greater than 0

- Altering the inline length of a LOB column
- Extending the precision of the TIMESTAMP column

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

- The table space was created with DEFINE NO.
- You extend the length of a varying character (VARCHAR data type) or varying graphic (VARGRAPHIC data type) column, and the table does not have a version number yet.
- You specify the same data type and length that a column currently has, so that its definition does not actually change.
- You add a column to a table in a non-partitioned table space, and the table is already versioned with no data in the current committed version format.
- You are altering the maximum length of a LOB column and the table does not have a version number yet.

GUPI

DB2 creates only one table space version if you make multiple schema changes in the same unit of work. 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;
```

GUPI

Related tasks:

"Altering the data type of a column" on page 128

"Altering the attributes of an identity column" on page 160

Reorganizing table spaces:

After you commit a schema change, DB2 puts the affected table space in an advisory REORG-pending (AREO*) state. The table space stays in this state until you reorganize the table space and apply the schema changes.

Procedure

To reorganize the table space and apply the schema changes:

Run the REORG TABLESPACE utility.
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 that 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.

Related concepts:

"Row format conversion for table spaces" on page 672

Related reference:

REORG TABLESPACE (DB2 Utilities)

Recycling table space version numbers:

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.

About this task

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." The range of used versions is stored in the catalog. If necessary, in-use versions can be recovered from image copies of the table space.

Procedure

To recycle table space version numbers:

1. First, determine the range of version numbers that are currently in use for a table space by querying the OLDEST_VERSION and CURRENT_VERSION columns of the SYSIBM.SYSTABLESPACE catalog table.

   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 utility before you can recycle the version numbers.
2. Then, recycle the unused table space version numbers by running the MODIFY RECOVERY utility.

Altering a table for referential integrity

You can alter a table to add, change, or remove referential constraints.

Before you begin

If you plan to let DB2 enforce referential integrity in a set of tables, see Referential constraints (DB2 Application programming and SQL) for a description of the requirements for referential constraints. DB2 does not enforce informational referential constraints.

Related concepts:

DB2 and data integrity (Introduction to DB2 for z/OS)

Creation of relationships with referential constraints (Introduction to DB2 for z/OS)

Related tasks:

Creating tables for data integrity (DB2 Application programming and SQL)

Related reference:

ALTER TABLE (DB2 SQL)

Adding referential constraints to existing tables

You can use the ALTER TABLE statement to add referential constraints to existing tables.

About this task

Assume that the tables in the sample application (the DB2 sample activity table, project table, project activity table, employee table, and department table) already exist, have the appropriate column definitions, and are already populated.

Now, suppose that you want to define relationships among the sample tables by adding primary and foreign keys with the ALTER TABLE statement. The following rules apply to these relationships:

- 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.
- You must add the parent key of the parent table before adding the corresponding foreign key of the dependent table.

You can build the same referential structure in several different ways; however, the following process might be the simplest to understand.

Procedure

To add a referential constraint to an existing table:

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

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

   Deletions by the CHECK DATA utility are not bound by delete rules. The deletions 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 deletion 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. Then, correct any rows in the exception table, and use INSERT to replace the rows in the department table. This table is now consistent with existing data.
4. Drop the foreign key on MGRNO in the department table. This step 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 the CHECK DATA utility 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. Add the foreign key on MGRNO to the department table, which again leaves the table space in CHECK-pending status. Then, run the CHECK DATA utility. If you have not changed the data since the previous check, you can use the DELETE(YES) option, and the deletions will not cascade.
7. For each of the following tables, in the order shown, add its foreign keys, run the CHECK DATA utility with the DELETE(YES) option, 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 primary parent keys, unique parent keys, and foreign keys to an existing table.

About this task

Introductory concepts:

DB2 and data integrity (Introduction to DB2 for z/OS)

Creation of relationships with referential constraints (Introduction to DB2 for z/OS)

Application of business rules to relationships (Introduction to DB2 for z/OS)

When you add parent keys and foreign keys to an existing table, you must consider certain 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. 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.

Procedure

To add a key to a table:

1. Choose the type of key that you want to add.
2. GUPI Add the key by using the ALTER TABLE statement.

Option	Description

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 DSN8910.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 DSN8910.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 DSN8910.PROJ`
	`  ADD CONSTRAINT REFKEY_DEPTNO FOREIGN`
	`  KEY (DEPTNO) REFERENCES DSN8910.DEPT`
	`  ON DELETE RESTRICT;`

GUPI

Related tasks:

Creating indexes to improve referential integrity performance for foreign keys (DB2 Performance)

Related reference:

ALTER TABLE (DB2 SQL)

Dropping parent keys and foreign keys

You can drop primary parent keys, unique parent keys, and foreign keys from an existing table.

Before you begin

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.

Procedure

GUPI

To drop a key, complete the following steps:

1. Choose the type of key that you want to drop.
2. Drop the key by using the ALTER TABLE statement.

Option	Description
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.

Option	Description
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.

GUPI

Adding or dropping table check constraints

You can add or drop check constraints by using the ALTER TABLE statement.

Procedure

GUPI

To add or drop check constraints:

Issue the ALTER TABLE statement.

Option	Description
Adding 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.

Option	Description
Dropping check constraints	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.

GUPI

Adding partitions

You can use ALTER TABLE statements to add partitions to all types of partitioned table spaces.

About this task

You do not need to allocate extra partitions for expected growth when you create partitioned table spaces because you can add partitions as needed.

You can add a partition as the last logical partition of any table in any type of partitioned table space.

When you add partitions DB2 always 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 you add a partition.

You cannot add or alter a partition for a materialized query table.

Procedure

To add partitions:

Add a partition after the last existing logical partition by issuing an ALTER TABLE statement. In the ADD PARTITION clause, specify an ENDING AT value beyond the existing limit of the last logical partition. If the table space is a large table space, you can use the new partition immediately after the ALTER statement completes. In this case, the partition is not placed in REORG-pending (REORP) status because it extends the high-range values that were not previously used. For non-large table spaces, the partition is placed in REORP status because the last partition boundary was not previously enforced.

Examples

GUPI

For example, consider a table space that contains a transaction table named TRANS. The table 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 each year. The following table shows a representation of the table space.

Table 13. An example table space with 10 partitions

Limit value	Physical partition number	Data set name that backs the partition
12/31/2010	1	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A001
12/31/2011	2	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A002
12/31/2013	3	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A003
12/31/2013	4	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A004
12/31/2014	5	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A005
12/31/2015	6	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A006
12/31/2016	7	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A007
12/31/2017	8	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A008
12/31/2018	9	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A009
12/31/2019	10	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A010

Example 1: adding a partition after the last logical partition

To add a partition for the next year, you can issue the following statement:

```
ALTER TABLE TRANS ADD PARTITION ENDING AT ('12/31/2020');
```

GUPI

The following table shows a representative excerpt of the table space after the partition for the year 2020 is added.

Table 14. An excerpt of the table space, showing the added partition 11

Limit value	Physical partition number	Data set name that backs the partition
12/31/2018	9	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A009
12/31/2019	10	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A010
12/31/2020	11	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A011

GUPI

What to do next

After you add partitions, you might need to complete any of the following actions.

Alter attributes for added partitions

The attributes of the new partition are inherited or calculated. If it is necessary to change specific attributes for the new partition, you must issue separate ALTER TABLESPACE and ALTER INDEX statements after you add the partition. Examine the catalog to determine whether the inherited values require changes.

The added partition inherits most attributes from the previous last logical partition. However, certain exceptions apply.

GUPI

For example, if you want to specify the space attributes for a new partition, use the ALTER TABLESPACE and ALTER INDEX statements. For example, suppose that 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;
```

GUPI

Create auxiliary objects for LOB columns

If you add a partition to a base table that contains LOB columns, the table space is explicitly created, and the CURRENT RULES special register is not 'STD', complete the following steps:

1. If necessary after you issue the ALTER TABLE ADD PARTITION statement, create a LOB table space in the same database as its associated base table space.
2. Create an auxiliary table and associate the new auxiliary table with the base table.
3. Issue the CREATE UNIQUE INDEX statement to create a unique index on the auxiliary table.

Related concepts:

"Pending data definition changes" on page 173

Related reference:

ALTER TABLE (DB2 SQL)

Altering partitions

You can use the ALTER TABLE statement to alter the partitions of table spaces.

About this task

You can make the following changes:

- Change the boundary between partitions
- Rotate any logical partition to be the last partition
- Extend the boundary of the last partition
- Instruct DB2 to insert rows at the end of a table or appropriate partition

Procedure

To alter a partition:

Issue the ALTER TABLE statement and specify the options that you want to change.

Changing the boundary between partitions

You can change the boundary of a partition by explicitly specifying a new value for the limit key. The *limit key* is the highest value of the partitioning key for a partition. The *partitioning key* is the column or columns that are used to determine the partitions.

About this task

Alternatively, you can let DB2 determine any appropriate limit key changes to more evenly distribute the data across partitions. If you want DB2 to determine any limit key changes, follow the instructions in Redistributing data across partitions by using REORG (DB2 Utilities).

Procedure

To change the boundary between partitions:

1. Use an ALTER statement to modify the limit key value for each partition boundary that you want to change.

   If the partitioned table space uses table-controlled partitioning, use an ALTER TABLE statement with the ALTER PARTITION clause to alter the limit key. If the partitioned table space uses index-controlled partitioning, use an ALTER INDEX statement with the ALTER PARTITION clause.

   You can change the limit key values of all or most of the partitions. You can apply the changes to one or more partitions at a time, which allows for relatively small parts of the data to be unavailable at a time.

   After you alter the limit keys, the partitions on either side of the boundary are placed in REORG-pending (REORP) status. This status means that the data is unavailable until the affected range of partitions are reorganized.
2. Run the REORG TABLESPACE utility to redistribute data in the partitioned table space based on the new limit key values.

   This action also resets the REORP status and makes the data available. The following example specifies options that help maximize performance while reorganizing the data:

   ```
   REORG TABLESPACE DSN8S10E PART 2:3
     NOSYSREC  COPYDDN SYSCOPY  STATISTICS TABLE INDEX(ALL)
   ```

   This example reorganizes a range of partitions and includes the STATISTICS keyword, which means that REORG collects statistics about the specified range of partitions.

   You can reorganize a range of partitions, even if the partitions are not in REORP status. However, you cannot reorganize only a subset of the range of partitions that are in REORP status. You must reorganize the entire range to reset the restrictive status.

   If you run REORG on partitions that are in REORP or advisory REORG-pending (AREOR) status, consider the values that you set for the following options:

   SHRLEVEL

   You can specify SHRLEVEL REFERENCE or SHRLEVEL CHANGE when objects are in the REORP status. REORG materializes any pending definition changes at the table space level and for the indexes.

KEEPDICTIONARY
REORG ignores the KEEPDICTIONARY option for any partition that is in REORP status. REORG automatically rebuilds the dictionaries for the affected partitions. However, if you specify a range of partitions that includes some partitions that are not in REORP status, REORG accepts the KEEPDICTIONARY option for those nonrestricted partitions.

DISCARDDN and PUNCHDDN
Specify the DISCARDDN and PUNCHDDN data sets when the limit key for the last partition was reduced for a table space that is defined as LARGE or DSSIZE. Otherwise, REORG terminates and issues message DSNU035I and return code 8.

REORG writes SYSCOPY records as follows:

- If any partition is in REORP status when REORG runs, DB2 writes a SYSCOPY record with STYPE=A for each partition that is specified on the REORG job.
- If you take an inline image copy of a range of partitions, DB2 writes one SYSCOPY record with ICTYPE=F for each partition. Each record has the same data set name.

Related concepts:

"Differences between partitioning methods" on page 62

Related reference:

ALTER INDEX (DB2 SQL)

ALTER TABLE (DB2 SQL)

Syntax and options of the REORG TABLESPACE control statement (DB2 Utilities)

Advisory or restrictive states (DB2 Utilities)

Rotating partitions

You can use the ALTER TABLE statement to rotate any logical partition to become the last partition. Rotating partitions is supported for partitioned (non-universal) table spaces and range-partitioned table spaces, but not for partition-by-growth table spaces.

About this task

Recommendation:

GUPI

When you create a partitioned table space, you do not need to allocate extra partitions for expected growth. Instead, you can use the ALTER TABLE ADD PARTITION statement to add partitions as needed. If rotating partitions is appropriate for your application, use the ALTER TABLE ROTATE PARTITION statement to avoid adding another partition.

GUPI

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.

Procedure

GUPI

To rotate a partition to be the last partition:

1. Issue the ALTER TABLE statement and specify the ROTATE PARTITION option.
2. Optional: Run the RUNSTATS utility.

GUPI

Example

For example, assume that the partition structure of the table space is sufficient through the year 2006. The following table shows a representation of the table space through the year 2006. When another partition is needed for the year 2007, you determined that the data for 1996 is no longer needed. You want to recycle the partition for the year 1996 to hold the transactions for the year 2007.

Table 15. An excerpt of a partitioned table space

Partition	Limit value	Data set name that backs the partition
P008	12/31/2004	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A008
P009	12/31/2005	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A009
P010	12/31/2006	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A010

GUPI

To rotate the first partition for table TRANS to be the last partition, issue the following statement:

```
ALTER TABLE TRANS ROTATE PARTITION FIRST TO LAST
   ENDING AT ('12/31/2007') 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 first logical partition is deleted, and no delete triggers are activated. 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 12/31/2007 and the previous limit value of 12/31/2006. 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, if the table is large, or if the last partition before the rotation is empty.

GUPI

The following table shows a representation of the table space after the first partition is rotated to become the last partition.

Table 16. Rotating the first partition to be the last partition

Partition	Limit value	Data set name that backs the partition
P002	12/31/1997	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A002
P003	12/31/1998	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A003
P004	12/31/1999	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A004
P005	12/31/2000	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A005
P006	12/31/2001	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A006
P007	12/31/2002	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A007
P008	12/31/2003	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A008
P009	12/31/2004	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A009
P010	12/31/2005	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A010
P011	12/31/2006	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A011
P001	12/31/2007	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A001

Extending the boundary of the last partition

You can extend the boundary of the last partition of a table that uses table-controlled partitioning without impacting data availability.

Procedure

GUPI

To extend the boundary of the last partition:

Issue the ALTER TABLE statement with the ALTER PARTITION clause to specify a new boundary for the last partition.

GUPI

For more details on this process, see "Changing the boundary between partitions" on page 143.

Example

The following table shows a representation of a table space through the year 2007. You rotated the first partition to be the last partition. Now, you want to extend the last partition so that it includes the year 2008.

Table 17. Table space through the year 2007

Partition	Limit value	Data set name that backs the partition
P002	12/31/1997	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A002
P003	12/31/1998	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A003
P004	12/31/1999	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A004
P005	12/31/2000	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A005

Table 17. Table space through the year 2007 (continued)

Partition	Limit value	Data set name that backs the partition
P006	12/31/2001	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A006
P007	12/31/2002	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A007
P008	12/31/2003	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A008
P009	12/31/2004	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A009
P010	12/31/2005	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A010
P011	12/31/2006	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A011
P001	12/31/2007	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A001

GUPI

To extend the boundary of the last partition to include the year 2008, issue the following statement:

```
ALTER TABLE TRANS ALTER PARTITION 1 ENDING AT ('12/31/2008');
```

GUPI

You can use the partition immediately after the ALTER statement completes. The partition is not placed in REORG-pending (REORP) status, because it extends the high-range values that were not previously used.

Related concepts:

"Differences between partitioning methods" on page 62

Related reference:

ALTER TABLE (DB2 SQL)

Advisory or restrictive states (DB2 Utilities)

Splitting the last partition into two

To allow for future growth, you can truncate the last partition of a table space and move some of the data into a new partition.

About this task

If you truncate a partition and some of the data could fall outside of the new boundary that is defined by the limit key value, the partition is placed in REORG-pending (REORP) status.

You can reset the advisory REORG-pending or REORG-pending status in one of the following ways:

- Run REORG with the DISCARD option to reset the REORG-pending status, set the new partition boundary, and discard the data rows that fall outside of the new boundary.
- Add a partition for the data rows that fall outside of the current partition boundaries.

The topic describes the procedure for the second choice.

Procedure

To split a partition into two:

1. Issue the ALTER TABLE statement with the ALTER PARTITION clause to specify a new boundary for the last partition. For more details on this process, see "Changing the boundary between partitions" on page 143.
2. Issue the ALTER TABLE statement with the ADD PARTITION clause to add a partition.
3. Issue the REORG TABLESPACE utility on the new second-to-last and last partitions to remove the REORG-pending status.

Example

GUPI

For example, the following table shows a representation of a table space through the year 2015, where each year of data is saved in separate partitions. Assume that you want to split the data for 2015 into two partitions.

You want partition P001 to include only the data for the first six months of 2015 (through 06/30/2015). Then, you want to create a partition to include the data for the last six months of 2015 (from 07/01/2015 to 12/31/2015).

Table 18. Table space with each year of data in a separate partition

Partition	Limit value	Data set name that backs the partition
P002	12/31/2005	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A002
P003	12/31/2006	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A003
P004	12/31/2007	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A004
P005	12/31/2008	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A005
P006	12/31/2009	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A006
P007	12/31/2010	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A007
P008	12/31/2011	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A008
P009	12/31/2012	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A009
P010	12/31/2013	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A010
P011	12/31/2014	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A011
P001	12/31/2015	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A001

To truncate partition P001 to include data only through 06/30/2015, issue the following statement:

```
ALTER TABLE TRANS ALTER PARTITION 1 ENDING AT ('06/30/2015');
```

Next, to create a partition that holds the remainder of the year 2015, issue the following statement:

```
ALTER TABLE TRANS ADD PARTITION ENDING AT ('12/31/2015');
```

The following table shows a portion of the table space and the modified partitions:

Table 19. Table space with one year split into two partitions

Partition	Limit value	Data set name that backs the partition
P011	12/31/2014	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A011
P001	06/30/2015	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A001
P012	12/31/2015	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A012

GUPI

Related reference:

ALTER TABLE (DB2 SQL)

Advisory or restrictive states (DB2 Utilities)

Inserting rows at the end of a partition

To specify how you want DB2 to insert rows at the end of a partition, you can use the CREATE TABLE or ALTER TABLE statement.

Procedure

To insert rows at the end of a partition:

Issue a CREATE TABLE or ALTER TABLE statement and specify the APPEND option. The APPEND option has the following settings:

YES Requests data rows to be placed into the table by disregarding the clustering during SQL INSERT and online LOAD operations. Rather than attempting to insert rows in cluster-preserving order, rows are appended at the end of the table or appropriate partition.

NO Requests standard behavior of SQL INSERT and online LOAD operations, namely that they attempt to place data rows in a well clustered manner with respect to the value in the row's cluster key columns. NO is the default option.

After populating a table with the APPEND option in effect, you can achieve clustering by running the REORG utility.

Restriction: You cannot specify the APPEND option for tables created in XML or work file table spaces.

Adding XML columns

You can add XML columns to regular relational tables by using the ALTER TABLE statement.

About this task

When you add an XML column to a table, an XML table and XML table space are implicitly created to store the XML data. If the new XML column is the first XML column that you created for the table, DB2 also implicitly creates a BIGINT DOCID column to store a unique document identifier for the XML columns of a row.

DB2 also implicitly creates indexes. If this is the first XML column that you created for the table, DB2 implicitly creates an index on the DOCID column of the base table.

An XML column has several restrictions. The column cannot:

- Be specified as part of the primary key
- Be specified as part of a UNIQUE key
- Have a DEFAULT value specified
- Be specified as part of the FOREIGN KEY references clause
- Be specified as part of the REFERENCES table-name clause
- Be specified in the PARTITION BY RANGE clause
- Be used in a materialized query table even if the table is specified WITH NO DATA
- Be referenced in CHECK constraints
- Have GENERATED specified
- Have a FIELDPROC specified
- Have AS SECURITY LABEL specified
- Be added to a created temporary table
- The table that contains an XML column will not have its XML column value passed to a VALIDPROC
- Be part of a transition table

Procedure

GUPI

To add an XML column to an existing table:

Issue the ALTER TABLE statement and specify the ADD *column-name* XML option.

Example

```
ALTER TABLE orders ADD shipping_info XML;
```

GUPI

Related tasks:

"Altering implicitly created XML objects" on page 184

Related reference:

ALTER TABLE (DB2 SQL)

Altering tables to enable hash access

You can alter existing tables to take advantage of hash access organization and improve the performance of queries that access individual rows in a table.

Before you begin

Hash organization is only available on universal table spaces (UTS). If you want to enable hash access on a table space that is not already a UTS, you must first alter the table space to UTS.

About this task

Enabling hash access requires a table space reorganization, and disables some features such as index clustering.

Procedure

To alter an existing table to take advantage of hash organization:

1. Specify ADD ORGANIZE BY HASH in the organization-clause of your ALTER TABLE statement.
   a. Specify UNIQUE followed by the column names for one or more columns that contain unique values in each row. You can specify more than one column-name as long as no two rows in the table have the same values in those columns. You can only specify columns that are defined as NOT NULL. You can specify a maximum of 64 columns to be used as unique identifiers for hash access. The sum of the column length attributes must not exceed 255. DB2 maintains the uniqueness of the hash key columns, and an index is not needed for this purpose.
   b. Specify HASH SPACE followed by an integer and a modifier that specifies the size of the hash space. You can specify the size of the hash space in kilobytes, megabytes, and gigabytes. Specify:
      - `K` for kilobytes
      - `M` for megabytes
      - `G` for gigabytes

      You can specify a size that is larger than your data to minimize the overhead of access to data that overflows the hash space. The size that you specify is most important if you do not intend to immediately reorganize the table space and specify the AUTOESTSPACE(YES) option, as is recommended below. In that case, for more information about choosing an appropriate size for the hash space, see Fine-tuning hash space and page size (DB2 Performance).
2. Commit the ALTER TABLE statement.
3. Run the REORG TABLESPACE utility on the table space where your altered table is located. If you specify AUTOESTSPACE(YES) in the REORG TABLESPACE statement,DB2 automatically estimates the best size for the hash space based on information from the real-time statistics tables. If you specify AUTOESTSPACE(NO) in the REORG TABLESPACE statement, DB2 uses the hash space that you specified.

Example

GUPI

Consider the following ALTER TABLE statement:

```
ALTER TABLE EMP
ADD ORGANIZE BY HASH UNIQUE (EMPNO)
HASH SPACE 64 M;
```

In this example the user alters the EMP table, specifies to ADD ORGANIZE BY HASH, sets the EMPNO column as the unique identifier, and specifies a HASH SPACE of *64* with the modifier *M* for megabytes.

GUPI

What to do next

Monitor the real-time-statistics information about your table to verify that the hash access path is used regularly and to verify that the use of disk space is optimized.

Related tasks:

Organizing tables by hash for fast access to individual rows (DB2 Performance)

Managing space and page size for hash-organized tables (DB2 Performance)

Monitoring hash access (DB2 Performance)

"Altering the size of your hash spaces"

"Creating tables that use hash organization" on page 47

Related reference:

ALTER TABLE (DB2 SQL)

REORG TABLESPACE (DB2 Utilities)

Altering the size of your hash spaces

You can alter the size of your hash spaces when you are monitoring and tuning the performance of tables that are organized by hash.

About this task

When you tune the performance of tables that are organized by hash, you can alter the size of the hash space with the ALTER TABLE statement.

Procedure

To alter the size of the hash space for a table, use one of the following approaches:

- Run the REORG TABLESPACE utility on the table space and specify AUTOESTSPACE YES in the REORG TABLESPACE statement. DB2 automatically estimates a size for the hash space based on information from the real-time statistics tables. If you specify AUTOESTSPACE NO in the REORG TABLESPACE statement, DB2 uses the hash space that you explicitly specified for the table space.
- Specify ALTER ORGANIZATION in an ALTER TABLE statement.
  1. Specify SET HASH SPACE followed by an integer and a modifier specifying the size of the hash space. You can specify the size of the hash space in kilobytes, megabytes, and gigabytes. Specify:
     - `K` for kilobytes
     - `M` for megabytes
     - `G` for gigabytes

     Specify the size of your hash space based on the predicted size of the table. For more information about choosing an appropriate size for the hash space, see Fine-tuning hash space and page size (DB2 Performance). For example, the following statement specifies a size of 64 megabytes for the hash space of the EMP table:

     ```
     ALTER TABLE EMP
     ALTER ORGANIZATION  SET HASH SPACE 64 M;
     ```
  2. Commit the ALTER TABLE statement.

What to do next

Monitor the real-time-statistics information about your table to ensure that the hash access path is used regularly and that your disk space is used efficiently.

Related tasks:

Organizing tables by hash for fast access to individual rows (DB2 Performance)

Managing space and page size for hash-organized tables (DB2 Performance)

Monitoring hash access (DB2 Performance)

Fine-tuning hash space and page size (DB2 Performance)

"Altering tables to enable hash access" on page 150

Related reference:

ALTER TABLE (DB2 SQL)

REORG TABLESPACE (DB2 Utilities)

Adding a system period and system-period data versioning to an existing table

You can alter a table to add a system period so that DB2 maintains the beginning and ending timestamp values for a row.

Procedure

To add a system period to a table and define system-period data versioning:

1. Issue the ALTER TABLE statement on the base table to alter or add begin and end columns, and to add the system period. By adding a system period to the table, the table becomes a system-period temporal table.
2. Issue a CREATE TABLE statement to create a history table that will correspond with the system-period temporal table.
3. Issue the ALTER TABLE ADD VERSIONING statement with the USE HISTORY TABLE clause to define system-period data versioning on the table. This step establishes a link between the system-period temporal table and the history table.

Example

GUPI

For example, consider that you created a table named policy_info by issuing the following CREATE TABLE statement:

```
CREATE TABLE policy_info
(policy_id CHAR(10) NOT NULL,
coverage INT NOT NULL);
```

Issue the following ALTER TABLE statements to add the begin and end columns and a system period to the table:

```
ALTER TABLE policy_info ADD COLUMN sys_start TIMESTAMP(12) NOT NULL
GENERATED ALWAYS AS ROW BEGIN;

ALTER TABLE policy_info ADD COLUMN sys_end TIMESTAMP(12) NOT NULL
GENERATED ALWAYS AS ROW END;
```

```
ALTER TABLE policy_info ADD COLUMN trans_id TIMESTAMP(12);
GENERATED ALWAYS AS TRANSACTION START ID;

ALTER TABLE policy_info
ADD PERIOD SYSTEM_TIME(sys_start, sys_end);
```

To create a history table for this system-period temporal table, issue the following CREATE TABLE statement:

```
CREATE TABLE hist_policy_info
(policy_id CHAR(10) NOT NULL,
coverage INT NOT NULL,
sys_start TIMESTAMP(12) NOT NULL,
sys_end TIMESTAMP(12) NOT NULL,
trans_id TIMESTAMP(12));
```

To define system-period data versioning between the system-period temporal table and the history table, issue the following ALTER TABLE statement:

```
ALTER TABLE policy_info
ADD VERSIONING USE HISTORY TABLE hist_policy_info;
```

GUPI

Adding an application period to a table

You can alter a table to add an application period so that you maintain the beginning and ending values for a row.

Procedure

To add an application period to a table:

Issue the ALTER TABLE statement with the ADD PERIOD BUSINESS_TIME clause. The table becomes an application-period temporal table.

Example

For example, consider that you created a table named policy_info by issuing the following CREATE TABLE statement:

```
CREATE TABLE policy_info
(policy_id CHAR(4) NOT NULL,
coverage INT NOT NULL,
bus_start DATE NOT NULL,
bus_end DATE NOT NULL);
```

You can add an application period to this table by issuing the following ALTER TABLE statement:

```
ALTER TABLE policy_info ADD PERIOD BUSINESS_TIME(bus_start, bus_end);
```

You also can add a unique index to the table by issuing the following CREATE INDEX statement:

```
CREATE UNIQUE INDEX ix_policy
ON policy_info (policy_id, BUSINESS_TIME WITHOUT OVERLAPS);
```

Restriction: You cannot issue the ALTER INDEX statement with ADD BUSINESS_TIME WITHOUT OVERLAPS. DB2 issues SQL error code -104 with SQLSTATE 20522.

Manipulating data in a system-period temporal table

You can do update, insert, delete, and merge operations on a system-period temporal table.

Procedure

To manipulate data in a system-period temporal table:

Issue INSERT, UPDATE, DELETE, or MERGE statements to make the changes that you want. Timestamp information is stored in the timestamp columns, and historical rows are moved to the history table.

Restriction: You cannot issue SELECT FROM DELETE or SELECT FROM UPDATE statements when the FOR PORTION OF option is specified for either the UPDATE statement or the DELETE statement. DB2 issues an error in both of these cases (SQL error code -104 with SQLSTATE 20522).

Example

GUPI

The following example shows how you can insert data in the POLICY_INFO table by specifying the DEFAULT keyword in the VALUES clause for each of the generated columns:

```
INSERT INTO POLICY_INFO
VALUES ('A123', 12000, DEFAULT, DEFAULT, DEFAULT);
```

GUPI

Related concepts:

"Temporal tables" on page 53

Altering materialized query tables

You can use the ALTER TABLE statement to change a materialized query table to a base table, or to change the attributes of a materialized query table.

You can also use the ALTER TABLE statement to register an existing table as 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.

Registering an existing table as a materialized query table

You can take advantage of automatic query rewrite for an existing table by registering it as a materialized query table.

Procedure

GUPI To register an existing table as a materialized query table:

Issue an ALTER TABLE statement and specify the ADD MATERIALIZED QUERY AS option. GUPI

Example

For example, 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

GUPI 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. GUPI

Related tasks:

"Loading data by using the INSERT statement" on page 81

Related reference:

ALTER TABLE (DB2 SQL)

REFRESH TABLE (DB2 SQL)

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.

Procedure

GUPI

To change a materialized query table to a base table:

Issue an ALTER TABLE statement and specify the DROP MATERIALIZED QUERY option. For example,

```
ALTER TABLE TRANSCOUNT DROP MATERIALIZED QUERY;
```

What to do next

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.

GUPI

Changing the attributes of a materialized query table

You can use the ALTER TABLE statement to change the attributes of an existing materialized query table.

Procedure

GUPI

To change the attributes of an existing materialized query table:

1. Issue the ALTER TABLE statement.
2. Decide which attributes to alter.

Option	Description
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;`
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;`
Change back to a system-maintained materialized query table.	Specify the MAINTAINED BY SYSTEM option.

GUPI

Changing the definition of a materialized query table

After you create a materialized query table, you can change the definition in one of two ways.

Procedure

To change the definition of an existing materialized query table, use one of the following approaches:

- Optional: Drop and re-create the materialized query table with a different definition.
- Optional: Use ALTER TABLE statement 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

You can use the ALTER TABLE statement to make certain changes to a validation exit routine that is associated with a table, if one exists.

About this task

GUPI

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 DSN8910.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 DSN8910.EMP
  VALIDPROC EMPLNEWE;
```

GUPI

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.

Procedure

To ensure that the rows of a table conform to a new validation routine by using the REORG and LOAD utilities:

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 DSN8D91A.DSN8S91E
     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 DSN8910.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 to capture changed data

You can use the ALTER TABLE statement to write data changes for that table to a log in an expanded format.

Procedure

To alter a table to capture changed data:

1. Issue an ALTER TABLE statement.
2. Specify the DATA CAPTURE CHANGES option.

What to do next

You can 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.

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.

What to do next

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, VARCHAR, and CLOB.

About this task

The subtype is stored in the FOREIGNKEY column of SYSIBM.SYSCOLUMNS.

An M in the FOREIGNKEY column when the MIXED DATA installation option is NO for a string column in an ASCII or EBCDIC table is interpreted as SBCS data, not MIXED data.

Procedure

To alter the subtype of an existing string column:

Issue the ALTER TABLE statement.

GUPI

For example:

```
ALTER TABLE table-name ALTER COLUMN column-name
      SET DATA TYPE altered-data-type
```

GUPI

Related reference:

ALTER TABLE (DB2 SQL)

Altering the attributes of an identity column

You can change the attributes of an identity column by using the ALTER TABLE statement.

Procedure

To change the attributes of an identity column:

1. Issue an ALTER TABLE statement.
2. Specify the ALTER COLUMN option. This clause changes 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.

What to do next

Changing the data type of an identity column, like changing some other data types, requires that you drop and then re-create the table.

Related concepts:

"Table space versions" on page 132

Related tasks:

"Altering the data type of a column" on page 128

"Changing data types by dropping and re-creating the table"

Related reference:

ALTER TABLE (DB2 SQL)

Changing data types by dropping and re-creating the table

Some changes to a table cannot be made with the ALTER TABLE statement.

About this task

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

Procedure

To change data types:

1. Unload the table.
2. Drop the table.

   Attention: 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 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.

   GUPI 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.

     GUPI
5. Reload the table.

Related tasks:

"Altering the attributes of an identity column" on page 160

Implications of dropping a table

Dropping a table has several implications that you should be aware of.

GUPI

The DROP TABLE statement deletes a table. For example, to drop the project table, run the following statement:

```
DROP TABLE DSN8910.PROJ;
```

The statement deletes the row in the SYSIBM.SYSTABLES catalog table that contains information about DSN8910.PROJ. This statement also drops any other objects that depend on the project table. This action results in the following implications:

- 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.
- 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, along with any pending changes that are associated with the index.
- 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 the CREATE TABLE statement without the TABLESPACE clause), the table space and any pending changes that are associated with the table space are 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.
  - If the table space containing the table is partitioned, or contains only the one table, you can drop the table space.
  - If the table space containing the table is segmented, DB2 reclaims the space.
  - If the table space containing the table is 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).

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.

GUPI

Related tasks:

"Dropping, re-creating, or converting a table space" on page 120

Objects that depend on the dropped table

Before dropping a table, check to see what objects are dependent on the table. 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

GUPI

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 = 'DSN8910'
  AND BTYPE = 'T';
```

GUPI

Finding dependent packages

GUPI

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 = 'DSN8910'
  AND BTYPE = 'T';
```

GUPI

Finding dependent plans

GUPI

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 = 'DSN8910'
  AND BTYPE = 'T';
```

GUPI

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.

Re-creating a table

You can re-create a DB2 table to decrease the length attribute of a string column or the precision of a numeric column.

Procedure

To re-create a DB2 table:

1. If you do not have the original CREATE TABLE statement and all authorization statements for the table (for example, call the table 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 (for example, call the table T2) with the attributes that you want.
3. Copy the data from the old table T1 into the new table T2 by using one of the following methods:
   a. Issue the following INSERT statement:

      GUPI

      ```
      INSERT INTO T2
        SELECT * FROM T1;
      ```

      GUPI

   b. 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. Issue 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, you can drop the table space instead. By dropping the table space, 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.

What to do next

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 can alter a table to use a different page size, or you can move a table to a table space of a different page size.

Procedure

To 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 SYSREC*nn* data set and the control statements from the SYSPUNCH data set, which was created when the table was unloaded.

Altering DB2 views

To alter a view, you must drop the view and create a new view with your modified specifications.

Procedure

To drop and re-create a view:

1. Issue the DROP VIEW SQL statement.
2. Commit the drop. When you drop a view, DB2 also drops the dependent views.
3. Re-create the modified view using the CREATE VIEW SQL statement.

What to do next

Attention: When you drop a view, DB2 invalidates 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 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, see the following table.

Table 20. Catalog tables to check after dropping a view

Catalog table	What to check
SYSIBM.SYSPACKDEP	Packages dependent on the view
SYSIBM.SYSVIEWDEP	Views dependent on the view
SYSIBM.SYSTABAUTH	Users authorized to use the view

Related tasks:

"Creating DB2 views" on page 69

"Dropping DB2 views" on page 72

Related reference:

DROP (DB2 SQL)

COMMIT (DB2 SQL)

CREATE VIEW (DB2 SQL)

Altering views by using the INSTEAD OF trigger

Typically, you can only do normal insert, update, and delete operations on specific types of views, but you can use the INSTEAD OF trigger to extend the updatability of views.

About this task

Unlike other forms of triggers that are defined only on tables, INSTEAD OF triggers are defined only on views. If you use the INSTEAD OF trigger, the requested update operation against the view is replaced by the trigger logic, which performs the operation on behalf of the view.

Procedure

To alter a view by using the INSTEAD OF trigger:

Issue the CREATE TRIGGER statement and specify the INSTEAD OF trigger for insert, update, and delete operations on the view.

Related reference:

CREATE TRIGGER (DB2 SQL)

Altering DB2 indexes

You can add a new column to an index or change the description of an index at the current server by issuing the ALTER INDEX statement.

About this task

With the ALTER INDEX statement, you can:

- Add a new column to an index.
- Alter the PADDED or NOT PADDED attribute to change how varying-length columns are stored in the index.
- Alter the CLUSTER or NOT CLUSTER attribute to change how data is stored.
- Alter the compression setting using ALTER COMPRESS YES or ALTER COMPRESS NO.
- Change the limit key for index-controlled partitioning to rebalance data among the partitions in a partitioned table space.

For other changes, you must drop and re-create the index.

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.

Restrictions:

- If the padding of an index is changed, the index is placed in REBUILD-pending (RBDP) status and a new version of the index is not created.
- Any alteration to use index compression places the index in RBDP status.
- You cannot add a column with the DESC attribute to an index if the column is a VARBINARY column or a column with a distinct type that is based on the VARBINARY type.

Procedure

To change the description of an index at the current server:

Issue the ALTER INDEX statement. The ALTER INDEX statement can be embedded in an application program or issued interactively.

Related concepts:

Indexes that are padded or not padded (Introduction to DB2 for z/OS)

Related tasks:

Designing indexes for performance (DB2 Performance)

Related reference:

ALTER INDEX (DB2 SQL)

Related information:

"Implementing DB2 indexes" on page 72

Alternative method for altering an index

You can minimize the potential for data outages by using the ALTER INDEX statement with the BUFFERPOOL option.

The BUFFERPOOL option is supported as a pending definition change. If pending changes do not exist at the table space level, you can materialize the pending changes by running one of the following utilities:

- REORG INDEX with SHRLEVEL CHANGE or SHRLEVEL REFERENCE
- REORG TABLESPACE with SHRLEVEL CHANGE or SHRLEVEL REFERENCE

If pending changes exist at the table space level, you can materialize the pending changes that are associated with the table space (including the pending changes for the index) by running REORG TABLESPACE with SHRLEVEL CHANGE or SHRLEVEL REFERENCE.

Adding columns to an index

You can add columns to an index in two ways. You can add a column to an index when you add the column to a table, or you can specify that additional columns be appended to the set of index key columns of a unique index.

Adding a column to an index when you add the column to a table

When you use the ALTER INDEX statement to add a column to an existing index, the new column becomes the rightmost column of the index key.

About this task

Restriction: You cannot add columns to IBM-defined indexes on the DB2 catalog.

Procedure

To add a column to an existing index:

1. Issue the ALTER INDEX ADD COLUMN SQL statement when you add a column to a table.
2. Commit the alter procedure.

Results

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.

If you add a column to an index and to a table within the same unit of work, this will cause table and index versioning.

Example

GUPI For example, assume that you created a table 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 immediately can use the new index with the key STATE, ZIPCODE for data access.

GUPI

Related reference:

ALTER INDEX (DB2 SQL)

Adding columns to the set of index keys of a unique index

You can use the ALTER INDEX statement to specify that additional columns be appended to the set of index key columns of a unique index.

About this task

Restriction: You cannot add columns to IBM-defined indexes on the DB2 catalog.

If you want to add a column to a unique index to allow index-only access of the data, you first must determine whether existing indexes on a unique table are being used to query the table. You can use the RUNSTATS utility, real-time statistics, or the EXPLAIN statement to find this information. Those indexes with the unique constraint in common are candidates for consolidation. Other non-unique indexes might be candidates for consolidation, depending on their frequency of use.

Procedure

To specify that additional columns be appended to the set of index key columns of a unique index:

1. Issue the ALTER INDEX statement with the INCLUDE clause. Any column that is included with the INCLUDE clause is not used to enforce uniqueness. These included columns might improve the performance of some queries through index only access. Using this option might eliminate the need to access data pages for more queries and might eliminate redundant indexes.
2. Commit the alter procedure. As a result of this alter procedure, the index is placed into page set REBUILD-pending (PSRBD) status, because the additional columns preexisted in the table.
3. To remove the PSRBD status from the index, complete one of the following options:
   - Run the REBUILD INDEX utility on the index that you ran the alter procedure on.
   - Run the REORG TABLESPACE utility on the index that you ran the alter procedure on, or you can wait to run the alter procedure until just before the REORG TABLESPACE utility is scheduled to run.
4. Run the RUNSTATS utility. The results will be used after the next step.
5. Perform REBIND on the static plans and packages.
6. Run the EXPLAIN statement to verify that the optimizer is choosing the index with the included columns.
7. Drop the indexes that are consolidated and no longer needed.

8. Verify that the new index is satisfying your query needs by using the RUNSTATS utility, real-time statistics, or the EXPLAIN statement.

Related reference:

ALTER INDEX (DB2 SQL)

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.

Procedure

GUPI

To alter how varying-length column values are stored in an index, complete the following steps:

1. Choose the padding attribute for the columns.
2. Issue the ALTER INDEX SQL statement.
   - Specify 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.
   - Specify 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.
3. Commit the alter procedure.

Results

The ALTER INDEX statement is successful only if the index has at least one varying-length column.

What to do next

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.

GUPI

Related concepts:

Indexes that are padded or not padded (Introduction to DB2 for z/OS)

Related reference:

ALTER INDEX (DB2 SQL)

Altering the clustering of an index

You can use the ALTER INDEX SQL statement to change the clustering index for a table.

Procedure

GUPI

To change the clustering option of an index:

1. Issue the ALTER INDEX statement.
2. Specify the clustering option.

   Restriction: You can only specify CLUSTER if there is not already another clustering index. In addition, an index on a table that is organized by hash cannot be altered to a clustering index.

   - 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.

     If you specify NOT CLUSTER for an index that is not a clustering index, that specification is ignored.
3. Commit the alter procedure.

GUPI

Related reference:

ALTER INDEX (DB2 SQL)

Dropping and redefining a DB2 index

Dropping an index does not cause DB2 to drop any other objects. The consequence of dropping indexes is that DB2 invalidates packages that use the index and automatically rebinds them when they are next used.

Before you begin

Any primary key, unique key, or referential constraints associated with a 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 9.

Commit the drop before you create any new table spaces or indexes by the same name.

Procedure

GUPI

To drop and re-create an index:

1. Issue a DROP INDEX statement.
2. Commit the drop procedure. The index space associated with the index is also dropped.
3. Re-create the modified index by issuing a CREATE INDEX statement.

4. Rebind any application programs that use the dropped index.

   GUPI

   If you drop and index and then run an application program using that index (and thereby automatically rebound), that application program does not use the old index. If, at a later time, you re-create the index and the application program is not rebound, the application program cannot take advantage of the new index.

Related tasks:

"Creating DB2 indexes" on page 72

Related reference:

DROP (DB2 SQL)

CREATE INDEX (DB2 SQL)

Reorganizing indexes

A schema change that affects an index might cause performance degradation. In this case, you might need to reorganize indexes to correct any performance degradation.

About this task

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. 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.

Procedure

To reorganize an index:

Run the REORG INDEX utility as soon as possible after a schema change that affects an index. You can also run the REORG TABLESPACE utility.

Related concepts:

"Index versions" on page 75

Related reference:

REORG INDEX (DB2 Utilities)

REORG TABLESPACE (DB2 Utilities)

Recycling index version numbers

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.

About this task

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, because 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.

Version numbers are considered to be unused if the schema changes that are associated with them have been applied and no image copies contain data at those versions.

Procedure

To recycle unused index version numbers:

1.

GUPI

Determine the range of version numbers that are currently in use for an index by querying the OLDEST_VERSION and CURRENT_VERSION columns of the SYSIBM.SYSINDEXES catalog table.

GUPI

2. Next, run the appropriate utility to recycle unused index version numbers.
   - For indexes that are defined as COPY YES, run the MODIFY RECOVERY utility.

     If all reusable version numbers (1 to 15) are currently in use, reorganize the index by running REORG INDEX or REORG TABLESPACE before you recycle the version numbers.
   - For indexes that are defined as COPY NO, run the REORG TABLESPACE, REORG INDEX, LOAD REPLACE, or REBUILD INDEX utility. These utilities recycle the version numbers as they perform their primary functions.

Related concepts:

"Index versions" on page 75

Pending data definition changes

Pending data definition changes are data definition changes that do not take effect immediately because the object must be reorganized to apply change. When you are ready to materialize pending data definition changes, you run the REORG utility to apply the pending changes to the definition and data. Objects that have pending definition changes remain available for use until it is convenient to apply the changes.

ALTER statements with certain options can cause pending changes to the definition of database objects. When an ALTER statement is issued that causes pending changes to the definition of an object, semantic validation and authorization checking are performed. However, changes to the table definition and data are not applied and the object is placed in advisory REORG-pending state (AREOR), until the REORG utility is run to resolve the pending changes.

Most pending data definition changes are supported only for universal table spaces, with the following exceptions:

- Converting single-table simple or segmented (non-universal) table spaces to partition-by-growth universal table spaces, with the MAXPARTITIONS attribute.
- Converting partitioned (non-universal) table spaces to range-partitioned universal table space, with the SEGSIZE attribute.

The pending changes are recorded in the SYSIBM.SYSPENDINGDDL catalog table. When the pending changes are applied, dependent packages are invalidated, the corresponding entries in the SYSIBM.SYSPENDINGDDL catalog table are removed, and the advisory REORG-pending state is removed.

When pending definition changes occur

The following situations can result in pending definition changes:

ALTER TABLESPACE

The following ALTER TABLESPACE options can cause pending changes to the definition of a table space:

- BUFFERPOOL is a pending change to the definition of the table space if the data sets of the table space are already created and if one of the following conditions is true:
  - Pending definition changes already exist for the table space or any associated indexes.
  - The specified buffer pool has a different page size than the buffer pool that is currently being used for the table space.
- DSSIZE is a pending change to the definition of the table space if the data sets of the table space are already created and if one of the following conditions is true:
  - Pending definition changes already exist for the table space or any associated indexes.
  - The specified DSSIZE is different than the value that is currently being used for the table space.
- MAXPARTITIONS is a pending change if the table space is not a partition-by-growth universal table space.
- SEGSIZE is a pending change to the definition of the table space if the data sets of the table space are already created and one of the following conditions is true:
  - Pending changes to the definition of the table space or its associated indexes already exist.
  - The specified SEGSIZE value for a universal table space is different than the existing value.
  - The table space is converted from a partitioned table space to a range-partitioned universal table space.

ALTER TABLE

The following ALTER TABLE options can cause pending changes to the definition of the table under certain conditions:

- DROP COLUMN, if the data sets of the table space are already created
- ALTER PARTITION, to change the limit keys for the following types of partitioned table spaces:
  - Range-partitioned universal table spaces
  - Partitioned table spaces (non-universal) with table-controlled partitioning. However, this operation is not a pending definition change under the following circumstances:
    - There is no possibility that data would move between partitions, and no other definition change is pending on the partition or the previous partition. In this case, the changes are immediate, and the partitions are not placed in a REORG-pending or advisory REORG-pending status.

- The values in the limit key for the last partition are all altered from MAXVALUE to a value less than MAXVALUE, or from MINVALUE to a value greater than MINVALUE. In this case, the changes are immediate, and the partition is placed in REORG-pending status.

- MEMBER CLUSTER is a pending change.

ALTER INDEX

The following ALTER INDEX options can cause pending changes to the definition of the specified index under certain conditions:

- BUFFERPOOL is a pending change if all of the following conditions are true:
  - The data sets of the index are created
  - The index is defined on one of the following:
    - A table that is in a universal table space
    - An XML table that is associated with a base table that is in a universal table space
    - An auxiliary table that is associated with a base table that is in a universal table space
  - There are pending definition changes for the index or the table space
  - The buffer pool is changed to a buffer pool with a different size
- COMPRESS is a pending change if all of the following conditions are true:
  - The data sets of the index are created
  - The index is defined on one of the following:
    - A table that is in a universal table space
    - An XML table that is associated with a base table that is in a universal table space
    - An auxiliary table that is associated with a base table that is in a universal table space
  - There are pending definition changes for the index or the table space
  - The buffer pool is changed to a buffer pool with a different size

ALTER INDEX BUFFERPOOL results in an immediate definition change except when all of the following conditions are true:

- The data sets of the index are created
- The index is defined on one of the following objects:
  - A table that is in a universal table space
  - An XML table that is associated with a base table this is in a universal table space
  - An auxiliary table that is associated with a base table that is in a universal table space
- There are pending definition changes for the index or the table space, or the buffer pool is changed to a buffer pool with a different size.

When pending changes are restricted

ALTER TABLESPACE, ALTER TABLE and ALTER INDEX statements that result in pending definition changes are not supported in the following cases:

- Options that cause pending changes cannot be specified with options that take effect immediately

- Options that cause pending changes cannot be specified for the following objects:
  - The DB2 catalog
  - System objects
  - Objects in a work file database
- The DROP PENDING CHANGES clause cannot be specified for a catalog table space
- If the DROP PENDING CHANGES clause is specified, no other clauses can be specified on the ALTER TABLESPACE statement
- If the table space, or any table it contains is in an incomplete state, you cannot specify options that cause pending changes
- For ALTER INDEX, if the definition of the table space or table on which the index is defined it not complete.

Most immediate definition changes are restricted while pending definition changes exist for an object. For a list of such restrictions, see "Restrictions for changes to objects that have pending data definition changes" on page 180.

Related concepts:

Types of DB2 table spaces (Introduction to DB2 for z/OS)

Related tasks:

"Altering table spaces" on page 117

Related reference:

ALTER TABLE (DB2 SQL)

ALTER TABLESPACE (DB2 SQL)

ALTER INDEX (DB2 SQL)

Materializing pending definition changes

After generating pending definition changes by issuing the ALTER TABLESPACE statement, you must materialize pending definition changes at the table space level. *Materialization* of the pending definition changes means implementing the changes in the database system.

About this task

Pending definition changes are data definition changes that do not take effect immediately. When definition changes are pending, the affected objects are available until it is convenient to implement the changes.

Most pending data definition changes are supported only for universal table spaces, with the following exceptions:

- Converting single-table simple or segmented (non-universal) table spaces to partition-by-growth universal table spaces, with the MAXPARTITIONS attribute.
- Converting partitioned (non-universal) table spaces to range-partitioned universal table space, with the SEGSIZE attribute.

Tip: Try to run REORG at a time when the data is not heavily accessed. Otherwise, application outages might occur, as described in Reorganization with pending definition changes (DB2 Utilities).

Procedure

To materialize pending data definition changes, use the following approaches:

- Run the REORG TABLESPACE utility with SHRLEVEL REFERENCE or SHRLEVEL CHANGE. Do not specify FASTSWITCH NO.

  Also note the restrictions for REBALANCE in Syntax and options of the REORG TABLESPACE control statement (DB2 Utilities).

  Restriction: Using the REORG TABLESPACE utility with SHRLEVEL REFERENCE or SHRLEVEL CHANGE does not drop empty partitions from a partition-by-growth universal table space.
- For pending definition changes for indexes, issue REORG INDEX statements. Only pending definition changes to the reorganized index are materialized. Pending definition changes to the table space or table remain pending.

Examples

Example: The following example provides a scenario that shows how you can use the ALTER TABLESPACE statement to generate pending definition changes, and then use the REORG TABLESPACE utility with SHRLEVEL REFERENCE to materialize pending definition changes at the table space level.

GUPI

Consider the following scenario:

1. In Version 8, you created the simple table space TS1 in database DB1, such as:

```
CREATE DATABASE DB1;
CREATE TABLESPACE TS1
BUFFERPOOL BP0
IN DB1;
CREATE TABLE USER1.TB1
(
COL1 INTEGER,
COL2 VARCHAR(10)
)
IN DB1.TS1;

CREATE INDEX USER1.IX1
ON USER1.TB1
( COL2 )
BUFFERPOOL BP0
COPY YES
;
```

2. After migrating to new-function mode in Version 10, you issue the following ALTER TABLESPACE statement to convert the simple table space to a partition-by-growth table space, and to change the buffer pool page size. Those changes are pending definition changes. Suppose that the changes take place at time 2012-10-04-07.14.20.204010:

```
ALTER TABLESPACE DB1.TS1 BUFFERPOOL BP8K0 MAXPARTITIONS 20 ;
```

   For each pending option in an ALTER statement, there is a corresponding entry in the SYSPENDINGDDL table. If you specify multiple pending options in one ALTER statement, each change has its own SYSPENDINGDDL entry, but the changes have the same create timestamp. In addition, the same ALTER statement text is stored repeatedly with each pending option entry that is specified with the ALTER statement. Therefore, issuing this ALTER TABLESPACE statement results in the table space being placed in AREOR state,

and two pending option entries are inserted into the SYSPENDINGDDL table with OBJTYPE = 'S' for table space. This ALTER statement has not changed the current definition or data, so the buffer pool in SYSTABLESPACE still indicates BP0, and the table space is still a simple table space.

3. Later at the time of 2012-10-09-07.15.22.216020, you issue the following ALTER TABLESPACE statement that has one pending option:

```
ALTER TABLESPACE DB1.TS1 SEGSIZE 64 ;
```

   This statement results in one entry being inserted into the SYSPENDINGDDL table with OBJTYPE = 'S', for table space. This ALTER statement has not changed the current definition or data, so the SEGSIZE in SYSTABLESPACE is still 0.

4. Next, you issue the following ALTER statement with one pending option at the time of 2012-12-14-07.20.10.405008:

```
ALTER INDEX USER1.IX1 BUFFERPOOL BP16K0;
```

   This statement results in the index being placed in AREOR state, and an entry is inserted into the SYSPENDINGDDL table with OBJTYPE = 'I', for index. This ALTER statement has not changed the current definition or data, so the buffer pool in SYSINDEXES still indicates BP0 for the index.

5. You issue another ALTER statement that is exactly the same as the previous one, at the time of 2012-12-20-04.10.10.605058. This statement results in another entry being inserted into the SYSPENDINGDDL table with OBJTYPE = 'I', for index.

6. You run the following SELECT statement to query the SYSPENDINGDDL catalog table:

```
SELECT DBNAME, TSNAME, OBJSCHEMA, OBJNAME, OBJTYPE, OPTION_SEQNO,
OPTION_KEYWORD, OPTION_VALUE, CREATEDTS, STATEMENT_TEXT
FROM SYSIBM.SYSPENDINGDDL
WHERE DBNAME = 'DB1'
AND TSNAME = 'TS1'
ORDER BY CREATEDTS
;
```

   This query results in the following output:

Table 21. Output from the SELECT statement for the SYSPENDINGDDL catalog

DBNAME	TSNAME	OBJSCHEMA	OBJNAME	OBJTYPE
DB1	TS1	DB1	TS1	S
DB1	TS1	DB1	TS1	S
DB1	TS1	DB1	TS1	S
DB1	TS1	USER1	IX1	I
DB1	TS1	USER1	IX1	I

Table 22. Continuation of output from the SELECT statement for the SYSPENDINGDDL catalog

OPTION_SEQNO	OPTION_KEYWORD	OPTION_VALUE	CREATEDTS
1	BUFFERPOOL	BP8K0	2012-10-04-07.14.20.204010
2	MAXPARTITIONS	20	2012-10-04-07.14.20.204010
1	SEGSIZE	64	2012-10-09-07.15.22.216020

Table 22. Continuation of output from the SELECT statement for the SYSPENDINGDDL catalog (continued)

OPTION_SEQNO	OPTION_KEYWORD	OPTION_VALUE	CREATEDTS
1	BUFFERPOOL	BP16K0	2012-12-14-07.20.10.405008
1	BUFFERPOOL	BP16K0	2012-12-20-04.10.10.605058

Table 23. Statement text output for the SELECT statement for the SYSPENDINGDDL catalog

STATEMENT_TEXT
ALTER TABLESPACE DB1.TS1 BUFFERPOOL BP8K0 MAXPARTITIONS 20;
ALTER TABLESPACE DB1.TS1 BUFFERPOOL BP8K0 MAXPARTITIONS 20;
ALTER TABLESPACE DB1.TS1 SEGSIZE 64;
ALTER INDEX USER1.IX1 BUFFERPOOL BP16K0;
ALTER INDEX USER1.IX1 BUFFERPOOL BP16K0;

GUPI

7. Next, you run the REORG INDEX utility with SHRLEVEL CHANGE on the index. For example:

   ```
   REORG INDEX USER1.IX1 SHRLEVEL CHANGE
   ```

   However, because pending definition changes exist for the table space, the REORG utility proceeds without materializing the pending definition changes for the index, and issues warning DSNU275I with RC = 4 to indicate that no materialization has been done on the index, because there are pending definition changes for the table space. After the REORG utility runs, all the SYSPENDINGDDL entries still exist, and the AREOR state remains the same.
8. Now, you run the REORG TABLESPACE utility with SHRLEVEL REFERENCE on the entire table space. For example:

   ```
   REORG TABLESPACE DB1.TS1 SHRLEVEL REFERENCE
   ```

   The REORG utility materializes all of the pending definition changes for the table space and the associated index, applying the changes in the catalog and data. After the REORG utility runs, the AREOR state is cleared and all entries in the SYSPENDINGDDL table for the table space and the associated index are removed. The catalog and data now reflect a buffer pool of BP8K0, MAXPARTITIONS of 20, and SEGSIZE of 64.

Related concepts:

- Reorganization with pending definition changes (DB2 Utilities)
- Types of DB2 table spaces (Introduction to DB2 for z/OS)

Related reference:

- ALTER TABLE (DB2 SQL)
- ALTER TABLESPACE (DB2 SQL)
- ALTER INDEX (DB2 SQL)

Restrictions for changes to objects that have pending data definition changes

When data definition statements specify pending changes, immediate data definition pages cannot be issued in the same statement. Certain immediate changes are also restricted in subsequent data definition statements until the REORG utility is run to materialized the pending data definition changes.

The following table lists immediate data definition changes that are restricted until any pending data definition changes are materialized for specific types of objects. DB2 issue SQLCODE -20385 for statements that cannot be processed because of pending data definition changes.

Table 24. Restrictions for data definition changes for pending data definition changes

Scope of change	Pending data definition changes	Restricted immediate definition changes for the table space and contained objects
Table space	ALTER TABLESPACE with any of the following options: BUFFERPOOL DSSIZE MAXPARTITIONS SEGSIZE	ALTER INDEX with ADD COLUMN option ALTER INDEX with PIECESIZE option ALTER INDEX with REGENERATE option ALTER INDEX with VCAT option ALTER TABLE with immediate option(s) ALTER TABLESPACE with CCSID option ALTER TABLESPACE with FREEPAGE option ALTER TABLESPACE with VCAT option if table space is not partitioned by growth CREATE INDEX on table in table space CREATE TABLE in table space DROP INDEX of index enforcing ROWID GENERATED BY DEFAULT column in explicitly created table space DROP INDEX of empty auxiliary index in explicitly created LOB table space if pending changes exist for base table space or objects in base table space DROP TABLE if table space was explicitly created DROP TABLE of empty auxiliary table if pending changes exist for base table space or objects in base table space

Table 24. Restrictions for data definition changes for pending data definition changes (continued)

Scope of change	Pending data definition changes	Restricted immediate definition changes for the table space and contained objects
Index	ALTER INDEX statements with any of the following options: BUFFERPOOL	ALTER INDEX with ADD COLUMN option ALTER INDEX with COMPRESS YES option ALTER INDEX with PIECESIZE option ALTER INDEX with REGENERATE option ALTER INDEX with VCAT option ALTER TABLE with immediate option(s) CREATE INDEX on table in table space DROP INDEX of index enforcing ROWID GENERATED BY DEFAULT column in explicitly created table space DROP TABLE of empty auxiliary table if pending changes exist for base table space or objects in base table space RENAME INDEX

Related concepts:

Reorganization with pending definition changes (DB2 Utilities)

Related tasks:

"Materializing pending definition changes" on page 176

Related reference:

ALTER TABLE (DB2 SQL)

ALTER TABLESPACE (DB2 SQL)

ALTER INDEX (DB2 SQL)

Related information:

-20385 (DB2 Codes)

Altering stored procedures

The process that you follow to alter a stored procedure depends on the type of stored procedure and how you want to alter it.

About this task

You can alter stored procedures in the following ways:

- For a native SQL procedure, you can alter the options and the body, and you can manage multiple versions.
- For an external SQL procedure, you can alter only the options.
- For an external stored procedure (a procedure that is written in a host language), you can alter the procedure options. If you alter the host language code, you need to prepare the code again.

Procedure

To alter an existing stored procedure:

1. Follow the process for the type of change that you want to make:
   - To alter the host language code for an external stored procedure, modify the source and prepare the code again. (Precompile, compile, and link-edit the application, and then bind the DBRM into a package.)
   - To alter the body of a native SQL procedure, issue the ALTER PROCEDURE statement with the REPLACE clause.
   - To alter the description of any type of stored procedure, issue the ALTER PROCEDURE statement with the options that you want.
2. Refresh the WLM environment if either of the following situations applies:
   - For external SQL procedures or external procedures, you changed the stored procedure logic or parameters.
   - You changed the startup JCL for the stored procedures address space.

     Restriction: In some cases, refreshing the WLM environment might not be enough. For example, if the change to the JCL is to the NUMTCB value, refreshing the WLM environment is not enough. The refresh fails because it cannot start a new WLM address space that has a different NUMTCB from the existing one. In this case, you need to do a WLM quiesce, followed by a WLM resume.

   Tip: To refresh the WLM environment, use the DB2-supplied WLM_REFRESH stored procedure rather than the REFRESH command. (The REFRESH command starts a new WLM address space and stops the existing one.)
3. If you disabled automatic rebinds, rebind any plans or packages that refer to the stored procedure that you altered.

Example

GUPI

Example of changing the WLM environment: The following example changes the stored procedure SYSPROC.MYPROC to run in the WLM environment PARTSEC:

```
ALTER PROCEDURE SYSPROC.MYPROC
   WLM ENVIRONMENT PARTSEC;
```

Example of changing the stored procedure to use another authorization ID: Assume that you defined the stored procedure SYSPROC.MYPROC with the SECURITY DEFINER option. When you specify the SECURITY DEFINER option, the external security environment for the stored procedure uses the authorization ID of the owner of the stored procedure to control access to non-SQL resources. The following example changes the stored procedure SYSPROC.MYPROC so that it uses the authorization ID of the person who is running the stored procedure to control access to non-SQL resources:

```
ALTER PROCEDURE SYSPROC.MYPROC
   SECURITY USER;
```

GUPI

Related tasks:

"Implementing DB2 stored procedures" on page 84

Related reference:

WLM_REFRESH stored procedure (DB2 SQL)

ALTER PROCEDURE (external) (DB2 SQL)

ALTER PROCEDURE (SQL - external) (DB2 SQL)

ALTER PROCEDURE (SQL - native) (DB2 SQL)

Altering user-defined functions

You can use the ALTER FUNCTION statement to update the description of user-defined functions.

Procedure

To alter a user-defined function:

Issue the ALTER FUNCTION SQL statement.

Results

Changes to the user-defined function take effect immediately.

Example

GUPI

Example 1: In the following 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;
```

Example 2: The following example changes the second function when any arguments are null:

```
ALTER FUNCTION SMITH.CENTER (CHAR(25), DEC(5,2), INTEGER)
   RETURNS ON NULL CALL;
```

GUPI

Related concepts:

User-defined functions (DB2 SQL)

Related tasks:

"Creating user-defined functions" on page 86

Defining a user-defined function (DB2 Application programming and SQL)

Related reference:

ALTER FUNCTION (external) (DB2 SQL)

ALTER FUNCTION (compiled SQL scalar) (DB2 SQL)

ALTER FUNCTION (SQL table) (DB2 SQL)

Altering implicitly created XML objects

You can alter implicitly created XML objects; however, you can change only some of the properties for an XML object.

Procedure

GUPI

To alter implicitly created XML objects:

Determine the restrictions on the XML object that you want to change. The following table provides information about the properties that you can or cannot change for a particular XML object.

Option	Description
XML table space	You can alter the following properties: • BUFFERPOOL (16 KB buffer pools only) • COMPRESS • PRIQTY • SECQTY • MAXROWS • FREEPAGE • PCTFREE • GBPCACHE • USING STOGROUP • ERASE • LOCKSIZE (The only possible values are XML and TABLESPACE.) • SEGSIZE • DSSIZE • MAXPARTITIONS XML table space attributes that are inherited from the base table space, such as LOG, are implicitly altered if the base table space is altered.
XML table	The ALTER TABLE ALTER PARTITION statement is not supported if the table contains an XML column.
Index	You cannot alter the following properties: • CLUSTER • PADDED • ADD COLUMN.

GUPI

Related tasks:

"Adding XML columns" on page 149

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*.

Before you begin

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 Access Method Services Commands 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.

About this task

You cannot change the high-level qualifier for DB2 data sets by using the DB2 installation or migration update process. You must use other methods to change this qualifier for both system data sets and user data sets.

The following procedures do not actually move or copy data.

Changing the high-level qualifier for DB2 data sets is a complex task. You should have experience with both DB2 and managing user catalogs.

Related concepts:

"Moving DB2 data" on page 195

Related information:

DFSMS Access Method Services Commands

Defining a new integrated catalog alias

You can define a new integrated catalog alias any time before you change the high-level qualifier for system data sets or user data sets.

Procedure

To define the new high-level qualifier as an alias to a current integrated catalog:

Issue the following access method services command:

```
DEFINE ALIAS (NAME (newcat) RELATE (usercat) CATALOG (master-cat))
```

Related information:

DFSMS Access Method Services Commands

Changing the qualifier for system data sets

To change the qualifier for system data sets, 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.

About this task

Important: The following steps must be done in sequence.

Changing 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 follow the installation process.

Procedure

To specify the new qualifier:

1. Run the installation CLIST, and specify INSTALL TYPE=INSTALL and DATA SHARING FUNCTION=NONE.
2. Enter new values for the fields shown in the following table.

Table 25. 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.
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 run job steps DSNTIZP and DSNTIZQ to update the DSNHDECP or a user-specified application defaults module (parameter SSID). Make sure that the updated *dsnhdecp* parameter does not go to the SDSNEXIT or SDSNLOAD library that is used by the active DB2 subsystem. Use caution when changing the subsystem ID. For more information, see "MVS PARMLIB

updates panel: DSNTIPM" for the discussion of panel DSNTIPM for PARMLIB members where the subsystem ID has to be changed.

Stopping DB2 when no activity is outstanding

Before stopping DB2, make sure the subsystem does not have any outstanding activity, such as outstanding units of recovery or pending writes. Ensuring that at restart, DB2 does not need to access the data sets through the log, which contains the old data set qualifiers.

Procedure

GUPI

GUPI

To stop DB2 when no activity is outstanding:

1. Stop DB2 by entering the following command:

   ```
   -STOP DB2 MODE(QUIESCE)
   ```

   This command allows DB2 to complete processing currently executing programs.
2. Start DB2 by entering the following command:

   ```
   -START DB2 ACCESS(MAINT)
   ```
3. Use the following commands to make sure the subsystem is in a consistent state.

   -DISPLAY THREAD(*) TYPE(*)
   -DISPLAY UTILITY (*)
   -TERM UTILITY(*)
   -DISPLAY DATABASE(*) RESTRICT
   -DISPLAY DATABASE(*) SPACENAM(*) RESTRICT
   -RECOVER INDOUBT

   Correct any problems before continuing.
4. Stop DB2 by entering the following command:

   ```
   -STOP DB2 MODE(QUIESCE)
   ```

   GUPI
5. Run the print log map utility (DSNJU004) to identify the current active log data set and the last checkpoint RBA.
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.

   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.

Renaming system data sets with the new qualifier

When renaming system data sets with a new qualifier, assume that the new qualifier and the old qualifier reside in the same user catalog.

Before you begin

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.

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. 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.

Procedure

To rename the system data sets:

1. Using IDCAMS, change the names of the catalog and directory table spaces. Be sure to specify the instance qualifier of your data set, *y*, which can be either I or J. For example,

```
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. For example,

```
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. For example,

```
ALTER oldcat.BSDS01  -
   NEWNAME (newcat.BSDS01)
ALTER oldcat.BSDS01.*  -
   NEWNAME (newcat.BSDS01.*)
ALTER oldcat.BSDS02  -
   NEWNAME (newcat.BSDS02)
ALTER oldcat.BSDS02.*  -
   NEWNAME (newcat.BSDS02.*)
```

Updating the BSDS with the new qualifier

Update the first BSDS with the new alias and correct data set names for the active logs. In this step, you do not attempt to change the names of existing archive log data sets.

Before you begin

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.

Procedure

To update the BSDS:

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.

Establishing a new xxxxMSTR cataloged procedure

After updating BSDS with the new qualifier and before you start DB2, establish a new xxxxMSTR cataloged procedure.

Procedure

To establish a new xxxxMSTR cataloged procedure:

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.

Starting DB2 with the new xxxxMSTR and load module

You can start DB2 with the new xxxxMSTR cataloged procedure and load module.

Procedure

To start DB2 with the new xxxxMSTR cataloged procedure and load module:

1. Issue a **START DB2** command with the module name as shown in the following example.

   ```
   -START DB2 PARM(new_name)
   ```
2. Optional: If you stopped DSNDB01 or DSNDB06 in "Stopping DB2 when no activity is outstanding" on page 187, you must explicitly start them in this step.

Changing qualifiers for other databases and user data sets

You can change qualifiers for DB2 databases other than the catalog and directory.

Before you begin

DSNDB07 is a system database that contains no permanent data, and can be deleted and redefined with the new qualifier. If you are changing the qualifier for DSNDB07, do that before changing the rest of the user databases.

About this task

You can change the databases in the following list that apply to your environment:
- DSNDB07 (work file database)
- DSNDB04 (system 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.

Important: 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 the system database DSNDB07.

The method that you use depends on if you have a new installation or a migrated installation of DB2 for z/OS.

Changing your work database for a new installation of DB2:

You can change the high-level qualifier for your work database if you have a new installation of DB2 for z/OS.

Procedure

To change your work database:

1. Reallocate the database by using the installation job DSNTIJTM from *prefix*.SDSNSAMP.
2. Modify your existing job by changing the job to remove the BIND step for DSNTIAD and renaming the data set names in the DSNTTMP step to your new names. Make sure that you include your current allocations.

Changing your work database for a migrated installation of DB2:

You can change the high-level qualifier for your work database if you have a migrated installation of DB2 for z/OS.

About this task

Migrated installations do not have a usable DSNTIJTM, because the IDCAMS allocation step is missing.

Procedure

To change your work database:

1. Stop the database by using the following command (for a database named DSNDB07):

   ```
   -STOP DATABASE (DSNDB07)
   ```

2. Drop the database by using the following SQL statement:

   ```
   DROP DATABASE DSNDB07;
   ```

3. Re-create the database by using the following SQL statement:

   ```
   CREATE DATABASE DSNDB07;
   ```

4. Define the clusters by using the following access method services commands. You must 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 by using the following commands:

   ```
   CREATE TABLESPACE DSN4K01
     IN DSNDB07
     BUFFERPOOL BP0
     CLOSE NO
     USING VCAT DSNC910;

   CREATE TABLESPACE DSN32K01
     IN DSNDB07
     BUFFERPOOL BP32K
     CLOSE NO
     USING VCAT DSNC910;
   ```

6. Start the database by using the following command:

   ```
   -START DATABASE (DSNDB07)
   ```

Changing user-managed objects to use the new qualifier

You can change user-managed objects to use the new high-level qualifier.

Procedure

To change user-managed objects:

1. Stop the table spaces and index spaces by 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.

What to do next

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

You can keep an existing DB2 storage group and change only the high-level qualifier.

About this task

The following procedure applies to most DB2-managed objects, but not to partition-by-growth table spaces. For information about changing the high-level qualifier for partition-by-growth table spaces, follow the procedure in "Moving DB2-managed data with REORG, RECOVER, or REBUILD" on page 197.

Procedure

To change DB2-managed objects:

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(*)
      ```

      Restriction: Some databases must be explicitly stopped to allow any alterations. For these databases, 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.

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.

Tools for moving DB2 data

Moving DB2 data can be complicated. Fortunately, several tools exist that can help to simplify the process.

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).

The following tools can help to simplify the operations:

- The REORG and LOAD utilities move data sets from one disk device type to another within the same DB2 subsystem.

  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.
- 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.
- 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.

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.
- DFSMS, which contains the following functional components:
  - Data Set Services (DFSMSdss)

    Use DFSMSdss to copy data between disk devices. You can use online panels to control this, through the Interactive Storage Management Facility (ISMF) that is available with DFSMS.
  - 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.
  - 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).

The following table shows which tools are applicable to specific operations.

Table 26. Tools applicable to data-moving operations

Tool	Moving a data set	Copying a database	Copying an entire subsystem
REORG and LOAD	Yes	Yes	No
UNLOAD	Yes	No	No
COPY and RECOVER	Yes	No	No
DSNTIAUL	Yes	Yes	No

Table 26. Tools applicable to data-moving operations (continued)

Tool	Moving a data set	Copying a database	Copying an entire subsystem
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.

Related concepts:

DB2 online utilities (DB2 Utilities)

DFSMShsm Storage Administration Reference

z/OS DFSMSdss Storage Administration

Related tasks:

"Loading data from DL/I" on page 83

Related information:

DFSMS Access Method Services Commands

DFSMShsm Managing Your Own Data

Moving DB2 data

DB2 provides several tools and options to make moving data easier.

You can move data within DB2 in several ways: copying a database, copying a DB2 subsystem, or by moving data sets within a particular DB2 subsystem.

Copying a relational database

Copying your relational database involves not only copying data, but also finding or generating, and executing, SQL statements to create storage groups, databases, table spaces, tables, indexes, views, synonyms, and aliases.

You can copy a database by using the DSN1COPY utility. 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

Copying a DB2 subsystem from one z/OS system to another involves the following:

- 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

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.

Related concepts:

"Moving a DB2 data set"

Related tasks:

"Changing the high-level qualifier for DB2 data sets" on page 185

Related reference:

DSN1COPY (DB2 Utilities)

Moving a DB2 data set

You can move DB2 data by using the RECOVER, REORG, or DSN1COPY utilities, or by using 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.

If you use storage groups, then you can change the storage group definition to include the new volumes.

The following procedures differ mainly in that the first procedure assumes that 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.

Choose between the following methods for moving data sets:

- "Moving data without REORG or RECOVER"
- "Moving DB2-managed data with REORG, RECOVER, or REBUILD" on page 197

Related tasks:

"Altering DB2 storage groups" on page 114

Moving data without REORG or RECOVER

You can move data that you do not want to reorganize or recover.

Procedure

To move data without using the REORG or RECOVER utilities:

1. Stop the database by issuing a STOP DATABASE command.

   GUPI

   ```
   -STOP DATABASE(dbname) SPACENAM(*)
   ```

   GUPI

2. Move the data, using DSN1COPY or a non-DB2 facility.
3. GUPI 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 by issuing a START DATABASE command.

   ```
   -START DATABASE(dbname) SPACENAM(*)
   ```

   GUPI

Related reference:

DSN1COPY (DB2 Utilities)

Moving DB2-managed data with REORG, RECOVER, or REBUILD

You can create a storage group (possibly using a new catalog alias) and move the data to that new storage group.

Procedure

To create a new storage group that uses the correct volumes and the new alias:

1. Execute the CREATE STOGROUP SQL statement to create the new storage group.

   GUPI For example:

   ```
   CREATE STOGROUP stogroup-name
      VOLUMES (VOL1,VOL2)
      VCAT (newcat);
   ```

   GUPI

2. Issue the STOP DATABASE command on the database that contains the table spaces or index spaces whose data sets you plan to move, to prevent access to those data sets.

   GUPI

   ```
   -STOP DATABASE(dbname) SPACENAM(*)
   ```

   GUPI

3. Execute ALTER TABLESPACE or ALTER INDEX SQL statements to assign the table spaces or indexes to the new storage group.

   GUPI

   ```
   ALTER TABLESPACE dbname.tsname
     USING STOGROUP stogroup-name;
   ALTER INDEX creator.index-name
     USING STOGROUP stogroup-name;
   ```

   GUPI

4. Using IDCAMS, rename the data sets for the index spaces and table spaces to use the high-level qualifier for the new storage group. 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. Issue the START DATABASE command to start the database for utility processing only.

   GUPI

   ```
   -START DATABASE(dbname) SPACENAM(*) ACCESS(UT)
   ```

   GUPI

6. Run the REORG utility or the RECOVER utility on the table space or index space, or run the REBUILD utility on the index space.
7. Issue the START DATABASE command to start the database for full processing.

   GUPI

   ```
   -START DATABASE(dbname) SPACENAM(*)
   ```

   GUPI

Scenario: Moving from index-controlled to table-controlled partitioning

You can change an existing index-controlled partitioned table space to a table-controlled partitioned table space and implement a DPSI.

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:

GUPI

- 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);
  ```

GUPI

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:

GUPI

```
DROP INDEX IX1;
CREATE INDEX IX3 ON TRANS(ACCTID)
   PARTITIONED CLUSTER;
COMMIT;

DROP INDEX IX2;
COMMIT;
```

GUPI

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:

GUPI

```
REORG TABLESPACE dbname.tsname PART 11
   SHRLEVEL CHANGE
```

GUPI

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.

Recommendations:

- 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.
-

  GUPI

  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.

  GUPI

Related concepts:

"Automatic conversion to table-controlled partitioning" on page 63

"Differences between partitioning methods" on page 62

Related tasks:

"Creating tables that use table-controlled partitioning" on page 61

Related reference:

CREATE INDEX (DB2 SQL)

CREATE TABLE (DB2 SQL)

Part 2. Operation and recovery

© Copyright IBM Corp. 1982, 2016

Chapter 4. Controlling DB2 operations by using commands

You can control most aspects of the operational environment by issuing commands. For most commands, DB2 issues output in the form of messages. DB2 also issues system messages for other situations. To operate and recover DB2 successfully, you must know how to issue commands and retrieve and interpret command output messages and system messages.

Before you begin

GUPI Before you can issue commands, you must have the required authorities and privileges. For descriptions of the authorities and privileges that are required for particular commands, see the "Authorities" sections in the topics for each command under DB2 and related commands (DB2 Commands).

For more information about specific privileges and authorities, see Privileges and authorities (Managing Security).

About this task

Introductory concepts:

Commands for controlling DB2 and related facilites (Introduction to DB2 for z/OS)

DB2 attachment facilities (Introduction to DB2 for z/OS)

You can control most aspects of the operational environment by using the DSN command of TSO and its subcommands and DB2 commands. However, you might also any of the following types of commands to control connections from various attachment facilities, the z/OS internal resource lock manager (IRLM), and the Administrative task scheduler:

- The TSO command DSN and its subcommands
- DB2 commands
- CICS attachment facility commands
- IMS commands
- Administrative task scheduler commands
- z/OS IRLM commands
- TSO CLISTs

For more information about the different types of commands that you can use control DB2 operations, see Types of commands (DB2 Commands).

Within the z/OSenvironment, you can issue most types of commands from different interactive contexts, including the following consoles and terminals:

- z/OS consoles
- TSO terminals, by any of the following methods:
  - Issuing the DSN command from the TSO READY prompt
  - Entering commands in the DB2 Commands ISPF panel of DB2I
- IMS terminals

© Copyright IBM Corp. 1982, 2016

- Authorized CICS terminals

You might notice the similarities between the types of commands and the types of consoles and terminals. However, do not confuse the types of commands with the types of consoles and terminals. Although they are related, you can issue many of the different commands types, and receive output messages, from many of the different consoles or terminals.

The following table summarizes the capabilities to issue different type of commands and receive output messages from specific consoles and terminals.

Table 27. 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
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. Does not apply to START DB2. Commands that are issued from IMS must have the prefix /SSR. Commands that are issued from CICS must have the prefix DSNC.
2. This applies when using outstanding WTOR.
3. The "Attachment facility unsolicited output" does not include "DB2 unsolicited output."
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 RDO.

You can issue many commands from within applications programs, such as programs of the following types:

- z/OS application programs
- Authorized CICS programs
- IMS programs

- AFP-authorized programs
- IFI application programs

GUPI

Related information:

About DB2 and related commands (DB2 Commands)

Issuing commands from the z/OS console

You can issue commands to DB2 subsystem from the z/OS console.

About this task

GUPI More than one DB2 subsystem can run under z/OS. From the console, you must add a prefix to a DB2 command with special characters that identify the subsystem that the command is directed to. The 1 - 8-character prefix is called the *command prefix*.

The command prefix of a DB2 subsystem is specified by the value of the COMMAND PREFIX field on the DSNTIPM installation panel. The default command prefix is a hyphen character concatenated with the subsystem name.

Procedure

To issue commands from a z/OS console:

Specify the command prefix for the DB2 subsystem before the command. For example, the following command starts the DB2 subsystem that is uses `-DSN1` for the command prefix:

```
-DSN1 START DB2
```

GUPI

Related reference:

COMMAND PREFIX field (DB2 Installation and Migration)

Related information:

DB2 and related commands (DB2 Commands)

Issuing commands from TSO terminals

You can connect and issue commands from TSO terminals by issuing a DSN command to invoke the DSN command processor explicitly, or through the DB2I (DB2 Interactive) ISPF panels.

About this task

GUPI

Introductory concepts:

Common ways to interact with DB2 for z/OS (Introduction to DB2 for z/OS)

TSO attachment facility (Introduction to DB2 for z/OS)

Procedure

To issue commands from a TSO terminal take one of the following actions:

- Issue a DSN command to start an explicit DSN session. The DSN command can be issued in the foreground or background, when running under the TSO terminal monitor program (TMP).

  Examples:

  Invoking a DSN session with five retries at 3-second intervals
  For example, the following TSO command invokes a DSN session, requesting five retries at 30-second intervals:

  ```
  DSN SYSTEM (DB2) RETRY (5)
  ```

  Displaying information about threads from a TSO session
  The TSO terminal displays:

  ```
  READY
  ```

  You enter:

  ```
  DSN SYSTEM (subsystem-name)
  ```

  The TSO terminal displays:

  ```
  DSN
  ```

  You enter:

  ```
  -DISPLAY THREAD
  ```

  DB2 returns the following messages:

  ```
  DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
  DSNV402I - ACTIVE THREADS -
  ⋮
  ```

- Start a DB2I session to invoke and implicit DSN session. The following figure shows the options of the DB2I PRIMARY OPTION MENU.
  When you complete operations by using the DB2I panels, DB2I invokes CLISTs,

```
DSNEPRI                    DB2I PRIMARY OPTION MENU          SSID: DSN
COMMAND ===>

Select one of the following DB2 functions and press ENTER.

 1  SPUFI                 (Process SQL statements)
 2  DCLGEN                (Generate SQL and source language declarations)
 3  PROGRAM PREPARATION   (Prepare a DB2 application program to run)
 4  PRECOMPILE            (Invoke DB2 precompiler)
 5  BIND/REBIND/FREE      (BIND, REBIND, or FREE plans or packages)
 6  RUN                   (RUN an SQL program)
 7  DB2 COMMANDS          (Issue DB2 commands)
 8  UTILITIES             (Invoke DB2 utilities)
 D  DB2I DEFAULTS         (Set global parameters)
 X  EXIT                  (Leave  DB2I)
```

Figure 14. The ISPF panel for the DB2I PRIMARY OPTION MENU

which start the DSN session and invoke appropriate subcommands. GUPI

Related concepts:

DSN command processor (DB2 Application programming and SQL)

Related tasks:

"Running TSO application programs" on page 217

"Controlling TSO connections" on page 287

Related reference:

The DB2I primary option menu (Introduction to DB2 for z/OS)

Related information:

DB2 and related commands (DB2 Commands)

Issuing commands from CICS terminals

You can enter all DB2 commands except START DB2 from a CICS terminal that is authorized to enter the DSNC transaction code.

Procedure

GUPI To issue commands from CICS terminals:

Use the DSNC transaction. CICS can attach to only one DB2 subsystem at a time, so it does not use the DB2 command prefix. Instead, each command that is entered through the CICS attachment facility must be preceded by a hyphen (-). The CICS attachment facility routes the commands to the connected DB2 subsystem and obtains the command responses.

Example

For example, you enter the following command:

```
DSNC -DISPLAY THREAD
```

DB2 returns the following messages:

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV402I - ACTIVE THREADS -
⋮
```

GUPI

Related information:

Issuing commands to DB2 using the DSNC transaction (CICS Transaction Server for z/OS)

Issuing commands from IMS terminals

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 IMS /SSR command.

About this task

GUPI An IMS subsystem can attach to more than one DB2 subsystem, so you need to add a prefix. Commands that are directed from IMS to DB2 with a special character that identifies which subsystem to direct the command to. That character is called the *command recognition character* (CRC). It is specified it when you define DB2 to IMS, in the subsystem member entry in IMS.PROCLIB.

Tip: You can use the same character for the CRC and the command prefix for a single DB2 subsystem. However, to do that, you must specify a one character command prefix. Otherwise you cannot match these identifiers.

Most examples in this information 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:

Example

You enter the following command:

```
/SSR ?DISPLAY THREAD
```

DB2 returns the following messages:

```
DFS058  SSR COMMAND COMPLETED
DSNV401I ? DISPLAY THREAD REPORT FOLLOWS -
DSNV402I ? ACTIVE THREADS -
⋮
```

GUPI

Related reference:

COMMAND PREFIX field (DB2 Installation and Migration)

/SSR (IMS) (DB2 Commands)

Issuing commands from application programs

Certain kinds of application programs can issue DB2 commands.

About this task

GUPI

APF-authorized programs

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 to the invoking application program. Use of the CART 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 that runs as a batch job, use the following 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
```

DB2 returns the following messages:

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV402I - ACTIVE THREADS -
⋮
DSN9022I - DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
```

IFI application programs

An application program can issue DB2 commands using the instrumentation facility interface (IFI). This method of issuing commands returns information about the success or failure of the command to your program. If the command issues a non-zero return code, the information returned to your program includes diagnostic information about the command processed. For commands that are executed asynchronously, the return code indicates whether the command started successfully.

The IFI application program protocols are available through the IMS, CICS, TSO attachment facilities, the call attachment facility (CAF), and the Resource Recovery Services attachment facility (RRSAF).

GUPI

Related concepts:

Submitting commands from monitor programs (DB2 Performance)

Related tasks:

Chapter 6, “Submitting work to DB2,” on page 217

Related reference:

COMMAND (DB2 Performance)

Destinations for command output messages

In most cases, DB2 command response are sent to the entering terminal or, for batch jobs, to the printed listing. In CICS, you can direct command responses to another terminal.

GUPI

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. Subsequent messages are routed to one or more z/OS consoles using the WTO function. For example, START DATABASE issues several messages one after another.

You can choose alternative consoles to receive the subsequent messages by assigning them the routing codes that are 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 hardcopy logging is available. Hardcopy logging is controlled by the z/OS system command VARY.

GUPI

Related reference:

z/OS VARY command (MVS System Commands)

-DISPLAY THREAD (DB2) (DB2 Commands)

-START DATABASE (DB2) (DB2 Commands)

Related information:

DSNV401I (DB2 Messages)

Unsolicited DB2 messages

Unsolicited subsystem messages can be sent to the z/OS console that issues the **START DB2** command. They also can be sent to consoles that have been assigned the routing codes that you listed in the DSNZP*xxx* module during DB2 installation.

However, the following messages from the IMS and the CICS attachment facilities are exceptions:

- Specific IMS attachment facility messages are sent to the IMS master terminal.
- Unsolicited CICS messages are sent to the transient data entries that are specified for the MSGQUEUE*n*(*name*) attribute in the RDO (resource definition online).
- CICS statistics messages that are issued because of shutdown are sent to the transient data entry that is specified in the RDO (STATSQUEUE).

Some DB2 messages that are sent to the z/OS console are marked as critical with 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 they are specifically deleted. This prevents the messages from being missed by the operator, who is required to take a specific action.

Related concepts:

How to interpret message numbers (DB2 Messages)

Related information:

Troubleshooting for CICS DB2 (CICS DB2 Guide)

Chapter 5. Starting and stopping DB2

You start and stop DB2 by using the **START DB2** and **STOP DB2** commands.

Before you begin

Before DB2 is stopped, the system takes a shutdown checkpoint. This checkpoint and the recovery log give DB2 the information it needs to restart.

About this task

You can limit access to data at startup and startup after an abend.

Starting DB2

When DB2 is installed, it is defined as a formal z/OS subsystem.

About this task

GUPI

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.

Procedure

To start a DB2 subsystem:

Issue the **START DB2** command by using one of the following methods:

- Issue the **START DB2** command from a z/OS console that is authorized to issue system control commands (z/OS command group SYS).

  The command must be entered from the authorized console and cannot be submitted through JES or TSO.

  Starting DB2 by a JES batch job or a z/OS START command is impossible. The attempt is likely to start an address space for DB2 that will abend (most likely with reason code X'00E8000F').
- Start DB2 from an APF-authorized program by passing a START DB2 command to the MGCRE (SVC 34) z/OS service.

GUPI

Messages at start

DB2 issues a variety of messages when you start DB2. The specific messages vary based on the parameters that you specify.

GUPI

At start time, DB2 issues some or all of the following messages.

© Copyright IBM Corp. 1982, 2016

```
$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.

GUPI

Subsystem parameters at start

Starting DB2 invokes the load module for subsystem parameters. This load module contains information that was 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. You can specify PARM (*module-name*) on the **START DB2** command to provide a parameter module other than the one that is specified at installation.

The **START DB2** command starts the system services address space, the database services address space, and, depending on specifications in the load module for subsystem parameters (DSNZPARM by default), the distributed data facility address space. Optionally, another address space, for the internal resource lock manager (IRLM), can be started automatically.

A conditional restart operation is available, but no parameters indicate normal or conditional restart on the **START DB2** command.

Related concepts:

"Conditional restart" on page 371

Related tasks:

"Starting DDF" on page 312

Related reference:

-START DB2 (DB2) (DB2 Commands)

Application defaults module name at start

You can use the DECP option of the **START DB2** command to specify the application defaults module that is loaded at DB2 startup.

Important: If your DB2 environment has a dependency on the name DSNHDECP, you should not include the DECP option. Tools from an independent software vendor, or your own applications might have a dependency on this module name. For example, if a CAF or RRSAF application does an implicit identify or connect operation, the module with the name DSNHDECP is used.

Restricting access to data

You can restrict access to data with an option of the START DB2 command.

Procedure

GUPI

To restrict access to data:

Issue the **START DB2** command with one of the following options:

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 omitting the ACCESS keyword or specifying ACCESS(*).

ACCESS(*)
To allow all authorized users to connect to DB2.

GUPI

Ending the wait state at startup

JCL errors sometimes occur (for example, a device allocation error or an incorrect region size). When JCL errors occur during startup of the database services address space, the DB2 subsystem goes into wait status.

Procedure

To end the wait status:

Cancel the system services address space and the distributed data facility address space from the console.

What to do next

After DB2 stops, check the start procedures of all three DB2 address spaces for correct JCL syntax.

To accomplish this check, compare the expanded JCL in the SYSOUT output with the correct JCL provided in MVS JCL Reference. Then, take the member name of

the erroneous JCL procedure, which is also provided in the SYSOUT data set, 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.

Restart options after an abend

Starting DB2 after it abends is different from starting it after the STOP DB2 command is issued.

After the STOP DB2 command, DB2 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 by using information on the recovery log, DB2 then 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 that is traditionally performed during system restart. You can delay the backout of long-running units of recovery by using installation options LIMIT BACKOUT and BACKOUT DURATION on panel DSNTIPL.

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.

Related tasks:

Chapter 10, “Restarting DB2 after termination,” on page 363

Related reference:

DSNTIPL: Active log data set parameters (DB2 Installation and Migration)

Stopping DB2

Before DB2 stops, all DB2-related write to operator with reply (WTOR) messages must receive replies.

About this task

Procedure

GUPI

To stop a DB2 subsystem:

Issue one of the following STOP DB2 commands:

- `-STOP DB2 MODE(QUIESCE)`
- `-STOP DB2 MODE(FORCE)`

The following messages are returned:

```
DSNY002I - SUBSYSTEM STOPPING
DSN9022I - DSNYASCP '-STOP DB2' NORMAL COMPLETION
DSN3104I - DSN3EC00 - TERMINATION COMPLETE
```

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.

What to do next

Before restarting DB2, 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
```

GUPI

Related concepts:

"Normal termination" on page 363

Related reference:

-START DB2 (DB2) (DB2 Commands)

-STOP DB2 (DB2) (DB2 Commands)

Chapter 6. Submitting work to DB2

Application programs that run under TSO, IMS, or CICS can use DB2 resources by executing embedded SQL statements.

About this task

Application programs must meet certain conditions to embed SQL statements and to authorize the use of DB2 resources and data. These conditions vary based on the environment of the application program.

All application programming default values, including the subsystem name that the programming attachment facilities use, are in the DSNHDECP load module. Make sure that your JCL specifies the proper set of program libraries.

Submitting work by using DB2I

Using the interactive program DB2I (DB2 Interactive), 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.

About this task

Introductory concepts:

Common ways to interact with DB2 for z/OS (Introduction to DB2 for z/OS)

Procedure

To submit work by using DB2I:

1. Log on to TSO by following your local procedures.
2. Enter ISPF.
3. Enter parameters to control operations.

Related concepts:

DSN command processor (DB2 Application programming and SQL)

Related tasks:

"Issuing commands from TSO terminals" on page 205

Running TSO application programs

You use the DSN command and a variety of DSN subcommands to run TSO applications.

Procedure

GUPI To run TSO application programs:

1. Log on to TSO.
2. Enter the DSN command.
3. Respond to the prompt by entering the RUN subcommand.

© Copyright IBM Corp. 1982, 2016

Results

The terminal monitor program (TMP) attaches the DB2-supplied DSN command processor, which in turn attaches the application program.

Example

The following example runs application program DSN8BC3. The program is in library *prefix*.RUNLIB.LOAD, which is the name that is assigned to the load module library.

```
DSN SYSTEM (subsystem-name)
RUN PROGRAM (DSN8BC3) PLAN(DSN8BH10) LIB ('prefix.RUNLIB.LOAD')
END
```

GUPI

Sources that DB2 checks to find authorization access for an application program

DB2 checks multiple sources to find authorization access for an application program.

DB2 checks the sources in the order that they are listed. If the first source is unavailable, DB2 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

You can modify either the RACF USER parameter or the TSO user ID by a locally defined authorization exit routine.

Running IMS application programs

To run IMS application programs, you can enter transactions from an IMS terminal. You also can invoke IMS transactions and commands by using the DB2-supplied stored procedures DSNAIMS or DSNAIMS2.

About this task

Use the DSNAIMS stored procedure to send commands and single-segment transactions. Use the DSNAIMS2 stored procedure to send commands and multi-segment transactions.

Application programs that contain SQL statements run in the message processing program (MPP), the batch message processing (BMP), the Fast Path region, or the IMS batch region.

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 that IMS provides 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 seven characters. If the ASXBUSER field contains binary zeros or blanks (which indicates that RACF or another security package is not active), IMS uses the PSB name instead.

An IMS terminal operator probably notices few differences between application programs that access DB2 data and programs that access DL/I data because IMS sends no DB2-related messages to a terminal operator. However, your program can signal DB2 error conditions with a message of your choice. For example, at its first SQL statement, a program receives an SQL error code if the resources that are 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.

You can run batch DL/I jobs to access DB2 resources; DB2-DL/I batch support uses the IMS attachment facility.

Related tasks:

Loading and running a batch program (DB2 Application programming and SQL)

Related reference:

DSNAIMS stored procedure (DB2 SQL)

DSNAIMS2 stored procedure (DB2 SQL)

Related information:

Application programming design

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.

About this task

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 connects 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 that is related to the transaction against a plan that is assigned to it. The authorization ID for the transaction can be the operator ID, terminal ID, transaction ID, RACF-authenticated user ID, or another identifier that is explicitly provided by the RDO (resource definition online).

Related concepts:

Ways to control access to DB2 subsystems (Introduction to DB2 for z/OS)

Related reference:

DSNACICS stored procedure (DB2 SQL)

Running batch application programs

Batch DB2 work can run in the TSO background under the TSO terminal monitor program (TMP) or in an IMS batch message processing (BMP) region. IMS batch regions can issue SQL statements.

About this task

For batch work that runs in the TSO background, the input stream can invoke TSO command processors, particularly the DSN command processor for DB2. This input stream can include DSN subcommands, such as RUN.

Example

The following example shows 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 this example:

- IKJEFT01 identifies an entry point for TSO TMP invocation. Alternative entry points that are 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 on 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 that are 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 that are invoked by the RUN subcommand. If batch programs stop because of errors, DB2 backs out any changes that were made since the last commit point.
- (*ssid*) is the subsystem name or group attachment name.

Related tasks:

Chapter 12, “Backing up and recovering your data,” on page 395

Running application programs using CAF

The call attachment facility (CAF) allows you to customize and control execution environments more extensively than the TSO, z/OS, or IMS attachment facilities. Programs that run in TSO foreground or TSO background can use either the DSN session or CAF. z/OS batch and started task programs can use only CAF.

About this task

IMS batch applications can also access DB2 databases through CAF, however, this method does not coordinate the commitment of work between the IMS and DB2 subsystems. Using the DB2 DL/I batch support for IMS batch applications is highly recommended.

Procedure

GUPI

To use the CAF:

Either link-edit or make available a load module known as the call attachment language interface, or DSNALI. Alternatively, you can link-edit with the Universal Language Interface program (DSNULI).
When the language interface is available, your program can use CAF to connect to DB2 in the following ways:

- DSNALI only: Implicitly, by including SQL statements or IFI calls in your program just as you would any program.
- DSNALI or DSNULI: Explicitly, by writing CALL DSNALI or CALL DSNULI statements.

GUPI

Related concepts:

Call attachment facility (DB2 Application programming and SQL)

Running application programs using RRSAF

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 subsystems.

Before you begin

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.

Procedure

GUPI

To use RRSAF:

Either link-edit or make available a load module known as the RRSAF language interface, or DSNRLI. Alternatively, you can link-edit with the Universal Language Interface program (DSNULI).
When the language interface is available, your program can use RRSAF to connect to DB2 in the following ways:

- DSNRLI only: Implicitly, by including SQL statements or IFI calls in your program just as you would any program.
- DSNRLI or DSNULI: Explicitly, by using CALL DSNRLI or CALL DSNULI statements to invoke RRSAF functions. Those functions establish a connection between DB2 and RRS and allocate DB2 resources.

Related concepts:

Resource Recovery Services attachment facility (DB2 Application programming and SQL)

Related tasks:

"Controlling RRS connections" on page 307

Chapter 7. 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.

About this task

You manage the task list of the administrative task scheduler 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 administrative task scheduler.

About this task

At each point in time when the administrative task 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 administrative task 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 administrative task scheduler stores the execution status of the task in its redundant task lists, in relation with the task itself.

Adding a task

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.

About this task

The request and the parameters are transmitted to the administrative task scheduler that is 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 to the 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 administrative task scheduler analyzes the task to schedule its next execution.

Related reference:

ADMIN_TASK_ADD stored procedure (DB2 SQL)

© Copyright IBM Corp. 1982, 2016

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

Only one of these definitions can be specified for any single task. The other parameters must be null.

Table 28. 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

You can use different combinations of parameters to define schedules for task executions.

Procedure

To define a new scheduled task:

Connect to the DB2 subsystem with sufficient authorization to call the ADMIN_TASK_ADD stored procedure. The following task definitions show some common scheduling options.

To define	Do this
A task that executes only one time:	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 one administrative task scheduler. A task can be added by one administrative task scheduler, and then executed by any of the administrative task schedulers that are in the data sharing group.

Procedure

To force a task to be executed on a particular administrative task scheduler:

Specify the associated DB2 subsystem ID in the *DB2-SSID* parameter when you schedule the task.

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 ADMIN_TASK_LIST function to list tasks that are scheduled for execution by the administrative task scheduler.

Procedure

To list scheduled tasks:

Connect to the DB2 subsystem with sufficient authorization to call the function ADMIN_TASK_LIST. The function contacts the administrative task scheduler to update the DB2 task list in the table SYSIBM.ADMIN_TASKS, if necessary, and then reads the tasks from the DB2 task list. 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.

Related reference:

ADMIN_TASK_LIST (DB2 SQL)

Listing the status of scheduled tasks

You can use user-defined table functions to view the last execution status of scheduled tasks, to list multiple execution statuses of scheduled tasks, and to display the results of a stored procedure task. Scheduled tasks are defined in the task list of an administrative task scheduler.

Listing the last execution status of scheduled tasks

You can use the ADMIN_TASK_STATUS() table function to view the last execution status of scheduled tasks.

About this task

Before a task is first scheduled, all columns of its execution status contain null values, as returned by the ADMIN_TASK_STATUS() table function. Afterwards, at least the TASK_NAME, USERID, DB2_SSID, STATUS, NUM_INVOCATIONS and START_TIMESTAMP columns contain non-null values. This information indicates when and under which user ID the task status last changed and the number of times this task was executed. You can interpret the rest of the execution status according to the different values of the STATUS column.

The ADMIN_TASK_STATUS() table function contacts the administrative task scheduler to update the DB2 task list in the SYSIBM.ADMIN_TASKS table, if necessary, and reads the tasks from this task list directly.

Procedure

To determine the last execution status of a scheduled task:

1. Issue the ADMIN_TASK_STATUS() table function to generate the status table.
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 the ADMIN_TASK_LIST and ADMIN_TASK_STATUS() table functions on the TASK_NAME column.

Results

The table that is created by the ADMIN_TASK_STATUS() table function 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 administrative task 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 administrative task 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 administrative task scheduler was stopped during the execution of a task, the status remains RUNNING until the administrative task scheduler is restarted. When the administrative task scheduler starts again, the status is changed to UNKNOWN, because the administrative task scheduler cannot determine if the task was completed.

Related tasks:

"Listing multiple execution statuses of scheduled tasks"

Related reference:

ADMIN_TASK_STATUS (DB2 SQL)

Listing multiple execution statuses of scheduled tasks

You can use the ADMIN_TASK_STATUS table function with the *max-history* parameter to view multiple execution statuses of scheduled tasks.

About this task

The ADMIN_TASK_STATUS(MAX_HISTORY) table function returns a row of data for the most recent executions of each task (up to the *max-history* value). This function contacts the administrative task scheduler to update the DB2 task list in the SYSIBM.ADMIN_TASKS table and the status history for the task in the SYSIBM.ADMIN_TASKS_HIST table, if necessary, and reads the task statuses from theses tables.

To prevent the SYSIBM.ADMIN_TASKS_HIST table from containing too many status entries, the administrative task scheduler limits the number of status entries per task that are stored. This limit is specified by the MAXHIST parameter. This

parameter is a positive integer with a default value of 10. When the limit is reached, the oldest status entries are deleted. You can specify this parameter in the started task when the administrative task scheduler starts, or you can modify this parameter dynamically by using the MODIFY console command.

Procedure

To list multiple execution statuses of scheduled tasks:

1. Issue the ADMIN_TASK_STATUS(MAX_HISTORY) table function to generate the status table. The *max-history* parameter specifies the number of execution statuses that you want to view.
2. Select the rows in the table that correspond to the task name.

   To order the execution statuses, you can use the NUM_INVOCATIONS and START_TIMESTAMP columns in the table that is returned.

   If a task ran fewer times than the *max-history* value, this function returns all of the execution statuses. If the SYSIBM.ADMIN_TASKS_HIST table is not available, this function returns only the last five execution statuses for each task. If a task has not run yet, a row of data is returned with all columns containing null values, except that the TASK_NAME column contains the name of the task.

   Tip: You can relate the task execution status to the task definition by joining the output tables from the ADMIN_TASK_LIST and ADMIN_TASK_STATUS(MAX_HISTORY) table functions on the TASK_NAME column.

Related tasks:

"Listing the last execution status of scheduled tasks" on page 229

Related reference:

ADMIN_TASK_STATUS (DB2 SQL)

Displaying the results of a stored procedure task

If the task that was executed is a stored procedure, you can use the ADMIN_TASK_OUTPUT table function to display the output parameter values and result sets.

Before you begin

To call this user-defined table function, you must have MONITOR1 privilege.

About this task

If the task that was executed is not a stored procedure, the ADMIN_TASK_OUTPUT table function returns an empty table. Also, if the SYSIBM.ADMIN_TASKS_HIST table is not accessible (for example, if the DB2 subsystem is down), the output of previous executions are not available to the ADMIN_TASK_OUTPUT table function and an empty table is returned.

Procedure

To display the results of a stored procedure task:

Call the ADMIN_TASK_OUTPUT table function. This user-defined table function returns up to one row for each output parameter of the stored procedure and up to one row for each column value of each row of each result set of the stored

procedure. If the output values and the result set values are too large to be stored in the OUTPUT column of the SYSIBM.ADMIN_TASKS_HIST table, only the last rows of the result sets are returned. The column and parameter values are returned as strings.

Related reference:

ADMIN_TASK_OUTPUT (DB2 SQL)

Updating the schedule for a task

You can modify the schedule of a task that is in the task list for the administrative task scheduler.

Procedure

To update the schedule for a task:

Call the ADMIN_TASK_UPDATE stored procedure. If the task that you want to update is running, the changes go into effect after the current execution finishes.

Related reference:

ADMIN_TASK_UPDATE stored procedure (DB2 SQL)

Stopping the execution of a task

You can attempt to stop the execution of a task that is currently running.

About this task

Not all tasks can be canceled as requested. Only the administrative task scheduler that currently executes the task can cancel a JCL task or a stored procedure task.

Procedure

To stop a task that is currently running:

Issue the ADMIN_TASK_CANCEL stored procedure on the DB2 subsystem that is specified in the DB2_SSID column of the task status. For a task that is running, the stored procedure cancels the DB2 thread or the JES job that the task runs in, and issues a return code of 0 (zero). If the task is not running or if cancellation of the task cannot be initiated, the stored procedure issues a return code of 12.

Related reference:

ADMIN_TASK_CANCEL stored procedure (DB2 SQL)

Removing a scheduled task

You can remove a scheduled task from the task list by using the ADMIN_TASK_REMOVE stored procedure.

About this task

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 ADMIN_TASK_REMOVE stored procedure .

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.

Procedure

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 ADMIN_TASK_REMOVE stored procedure. 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.

Related reference:

ADMIN_TASK_REMOVE stored procedure (DB2 SQL)

Manually starting the administrative task scheduler

The administrative task scheduler normally starts when DB2 starts, but you can start it manually if necessary.

Procedure

Use one of the following commands to start the administrative task scheduler:

- To start an administrative task scheduler that is named *admtproc* from the operator's console using the default tracing option, issue the MVS system command:

  ```
  start admtproc
  ```

- To start an administrative task scheduler that is named *admtproc* from the operator's console with tracing enabled, issue the MVS system command:

  ```
  start admtproc,trace=on
  ```

- To start an administrative task scheduler that is named *admtproc* from the operator's console with tracing disabled, issue the MVS system command:

  ```
  start admtproc,trace=off
  ```

Results

When the administrative task scheduler starts, message DSNA671I displays on the console.

Manually stopping the administrative task scheduler

You can manually stop the administrative task scheduler. You might want to do this when you are doing problem determination or in preparation for maintenance.

Procedure

To stop the administrative task scheduler:

- **Recommended method:** To stop an administrative task scheduler that is named *admtproc* from the operator's console, issue the following MVS system command:

  ```
  modify admtproc,appl=shutdown
  ```

  You can expect the following results:
  - The administrative task 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 administrative task scheduler, issue the following MVS system command:

  ```
  stop admtproc
  ```

  You can expect the following results:
  - Any task that was invoked by the administrative task 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 administrative task 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, check to see if the task has completed. If an interrupted task has not completed, you must 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 ADMIN_TASK_ADD stored procedure is called by a DB2 member. The administrative task scheduler that is associated with this DB2 member adds the task to the common task list. The task list is shared among all administrative task schedulers that are associated with the data sharing group members. The next time that an administrative task scheduler accesses the list, it detects the new task.

All administrative task schedulers of the data sharing group access this task list once per minute to check for new tasks. The administrative task scheduler that adds a task does not have to check the list, and can execute the task immediately. Any other administrative task scheduler can execute a task only after finding it in the updated task list. Any administrative task scheduler can remove a task without waiting.

To remove a task, the ADMIN_TASK_REMOVE stored procedure is called by a DB2 member. The administrative task scheduler that is associated with this DB2 member removes the task from the common task list. The next time that an administrative task scheduler checks the list, which is within one minute after the task has been removed, it detects that the task was deleted.

An administrative task scheduler cannot execute a task without first locking it in the task list. Locking a task prevents deleted tasks from being executed, because a deleted task is no longer in the list and cannot be locked. If a task cannot be locked, it cannot be executed. The locking also prevents double executions. A task that one administrative task scheduler has in progress is already locked, so no other administrative task scheduler can lock and execute the task.

Troubleshooting the administrative task scheduler

Error and informational messages from the administrative task scheduler are displayed on the console.

Error messages typically indicate a problem with the configuration of the administrative task scheduler, or an error accessing its resources. Informational messages identify important steps in the life cycle of the administrative task scheduler, such as 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.

About this task

Use the MVS system command MODIFY to enable or disable tracing. When tracing is enabled, the trace goes to SYSOUT in the job output of the started task of the administrative task scheduler. The trace output helps IBM Software Support to troubleshoot problems.

Procedure

To enable tracing for problem determination:

- To start a trace for an administrative task scheduler that is named *admtproc*, issue the following MVS system command:

  ```
  modify admtproc,appl=trace=on
  ```

  Substitute the name of your administrative task scheduler for *admtproc*.
- To stop a trace for an administrative task scheduler that is named *admtproc*, issue the following MVS system command:

  ```
  modify admtproc,appl=trace=off
  ```
- To configure the system so that tracing starts automatically when the administrative task scheduler starts, modify the procedure parameter TRACE in the JCL job that starts the administrative task scheduler. This job has the name that was assigned when the administrative task 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 to protect your system in case there is a media failure. Console message DSNA679I indicates that one of these copies is not accessible. If this happens, you can recover the task list.

About this task

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. You can change the frequency of this message by modifying the ERRFREQ parameter. You can modify this parameter as part of the started task or with a console command.)

Procedure

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 task scheduler started task user.

  As soon as the ADMIN_TASKS table is accessible again, the administrative task 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 administrative task 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 administrative task scheduler is no longer operable. Messages DSNA681I and DSNA683I display on the console and the administrative task 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 task scheduler.

  As soon as both task lists are accessible again, the administrative task scheduler performs an autonomic recovery of the VSAM task list using the content of the recovered ADMIN_TASKS table.

Related tasks:

Installation step 22: Set up the administrative task scheduler (DB2 Installation and Migration)

Problems executing a task

When the administrative task scheduler has problems executing a task, the error description is written into the last execution status. The problem might be that the task did not execute successfully, or the administrative task scheduler detected an error at the end of task execution.

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 administrative task 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.

Problems in user-defined table functions

An SQL code and a few characters of an SQL error message are returned in response to either the ADMIN_TASK_LIST function or the ADMIN_TASK_STATUS function.

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 administrative task scheduler. Troubleshoot the task itself.

Problems in stored procedures

An SQL code and a few characters of an SQL error message are returned in response to either the ADMIN_TASK_ADD stored procedure or the ADMIN_TASK_REMOVE stored procedure.

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 administrative task 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 it is in a separate process. The administrative task scheduler is accessed through an SQL API and stores the scheduled tasks in two redundant task lists.

The administrative task scheduler is part of DB2 for z/OS. When properly configured, it is available and operable with the first DB2 start. The administrative task scheduler starts as a task on the z/OS system during DB2 startup. The administrative task scheduler has its own address space, named after the started task name.

Each DB2 subsystem has its own distinct administrative task scheduler connected to it. DB2 is aware of the administrative task scheduler whose name is defined in the subsystem parameter ADMTPROC. The administrative task scheduler is aware of DB2 by the subsystem name that is defined in the DB2SSID parameter of the started task.

The administrative task 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 administrative task 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.

The following figure shows the architecture of the administrative task scheduler.

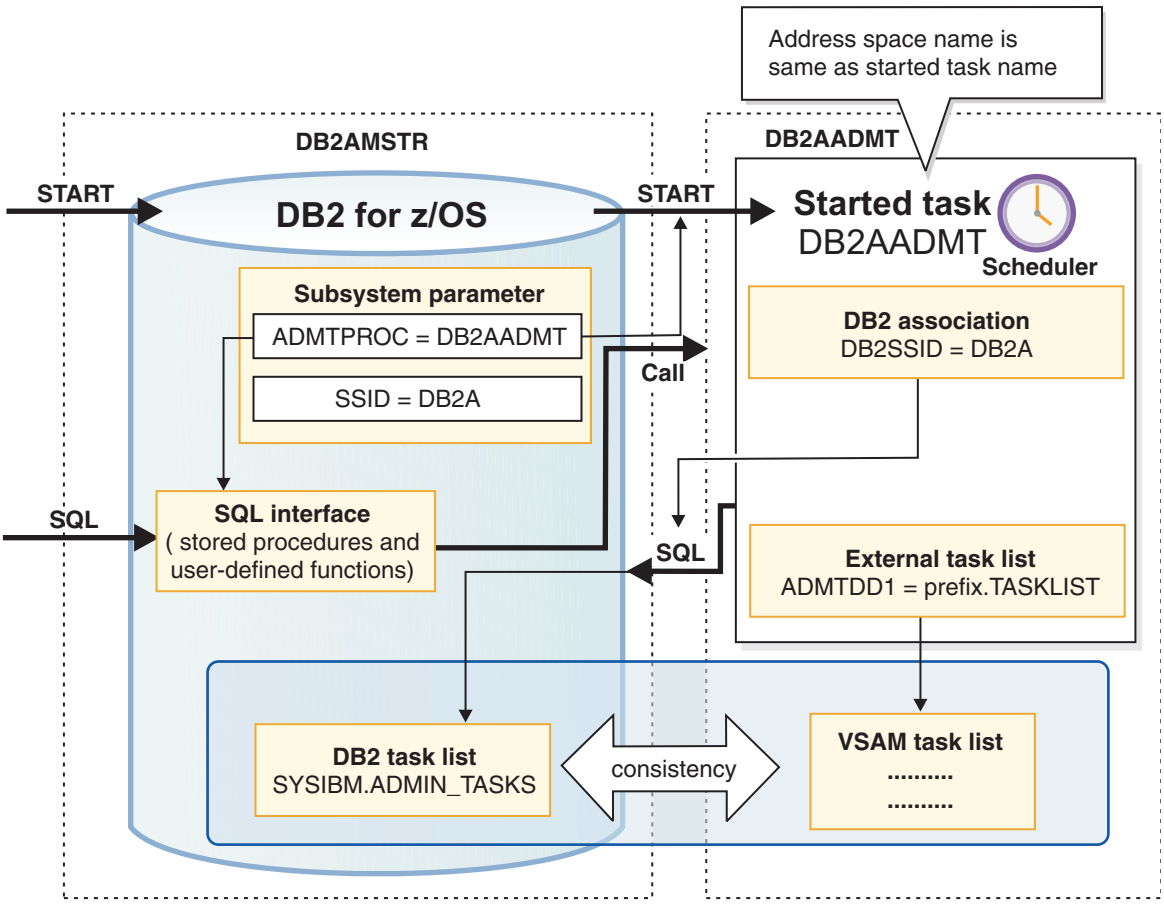


Figure 15. Architecture of the administrative task scheduler

Related reference:

- ADMIN_TASK_ADD stored procedure (DB2 SQL)
- ADMIN_TASK_REMOVE stored procedure (DB2 SQL)

The lifecycle of the administrative task scheduler

The administrative task scheduler starts as a task on the z/OS system during DB2 startup or initialization. The administrative task scheduler 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 administrative task scheduler cannot run simultaneously. To avoid starting up a duplicate administrative task scheduler, at startup the administrative task scheduler 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 administrative task scheduler immediately shuts down and displays a console message (DSNA674I). The administrative task scheduler can check the address spaces only in the same system, not the entire Sysplex.

When DB2 terminates, the administrative task scheduler remains active so that scheduled JCL jobs can run. When DB2 starts again, it connects to the

administrative task scheduler automatically. It does not need to be restarted.

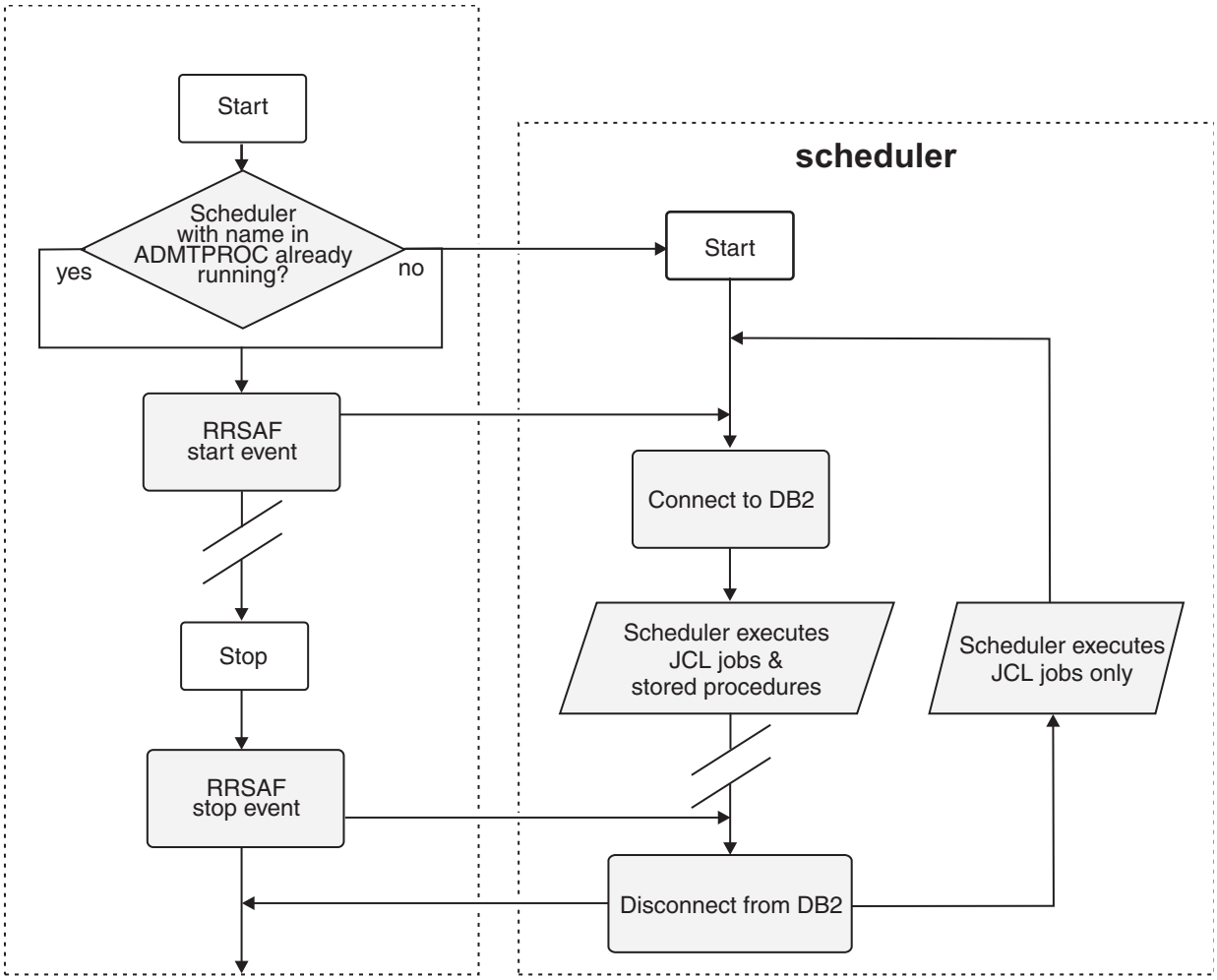


Figure 16. The lifecycle of the administrative task scheduler

If you want the administrative task scheduler to terminate when DB2 is stopped, you can specify the STOPONDB2STOP parameter in the started task before restarting the administrative task scheduler. This parameter has no value. You specify this parameter by entering STOPONDB2STOP without an equal sign (=) or a value. When you specify this parameter, the administrative task scheduler terminates after it finishes executing the tasks that are running and after executing the tasks that are triggered by DB2 stopping. When DB2 starts again, the administrative task scheduler is restarted.

Important: When you use the STOPONDB2STOP parameter to stop the administrative task scheduler when DB2 is stopped, JCL tasks will not run. These JCL tasks will not run, even if they could have run successfully had an administrative task scheduler remained active.

Related tasks:

Installation step 22: Set up the administrative task scheduler (DB2 Installation and Migration)

Task lists of the administrative task scheduler

The administrative task scheduler manages tasks that are defined by users and stores them in two redundant task lists. As a result, the administrative task 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 started task of the administrative task scheduler. The administrative task 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 that is identified in the *DB2SSID* parameter of the started task of the administrative task scheduler.

The administrative task scheduler works and updates both task lists redundantly, and remains operable so long as at least one of the task lists is available. Therefore, the administrative task scheduler continues working when DB2 is offline. If a task list becomes unavailable, the administrative task scheduler continues to update the task list. When both task lists are available again, the administrative task scheduler automatically synchronizes them.

The SYSIBM.ADMIN_TASKS_HIST table is a history table that stores the status and execution output of the tasks that are in the SYSIBM.ADMIN_TASKS table. The administrative task scheduler manages the SYSIBM.ADMIN_TASKS_HIST table simultaneously to the task lists that are in the SYSIBM.ADMIN_TASKS table and the VSAM data set.

If the SYSIBM.ADMIN_TASKS_HIST table is not accessible (for example, if the DB2 subsystem is down), the output of previous executions are not available to the DSNADM.ADMIN_TASK_OUTPUT user-defined function. However, even if the SYSIBM.ADMIN_TASKS_HIST table is not accessible, the last five execution statuses are available to the DSNADM.ADMIN_TASK_STATUS user-defined function. The status and output of previous executions are stored in the task list until they can be copied to the SYSIBM.ADMIN_TASKS_HIST table.

To prevent the SYSIBM.ADMIN_TASKS_HIST table from containing too many status entries, the administrative task scheduler limits the number of status entries per task that are stored. This limit is specified by the MAXHIST parameter. This parameter is a positive integer with a default value of 10. When the limit is reached, the oldest status entries are deleted. You can specify this parameter in the started task when the administrative task scheduler starts, or you can modify this parameter dynamically by using the MODIFY console command.

Related tasks:

Installation step 22: Set up the administrative task scheduler (DB2 Installation and Migration)

Architecture of the administrative task scheduler in a data sharing environment

In a data sharing environment, each DB2 for z/OS Version 9.1 or later member of a data sharing group is associated with its own administrative task scheduler. Each member is associated with its own administrative task scheduler, even when those members run in the same z/OS system. The administrative task schedulers share their resources and interface.

The task list is shared by all administrative task 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.

The following figure shows a data sharing group with two DB2 members and their associated administrative task schedulers.

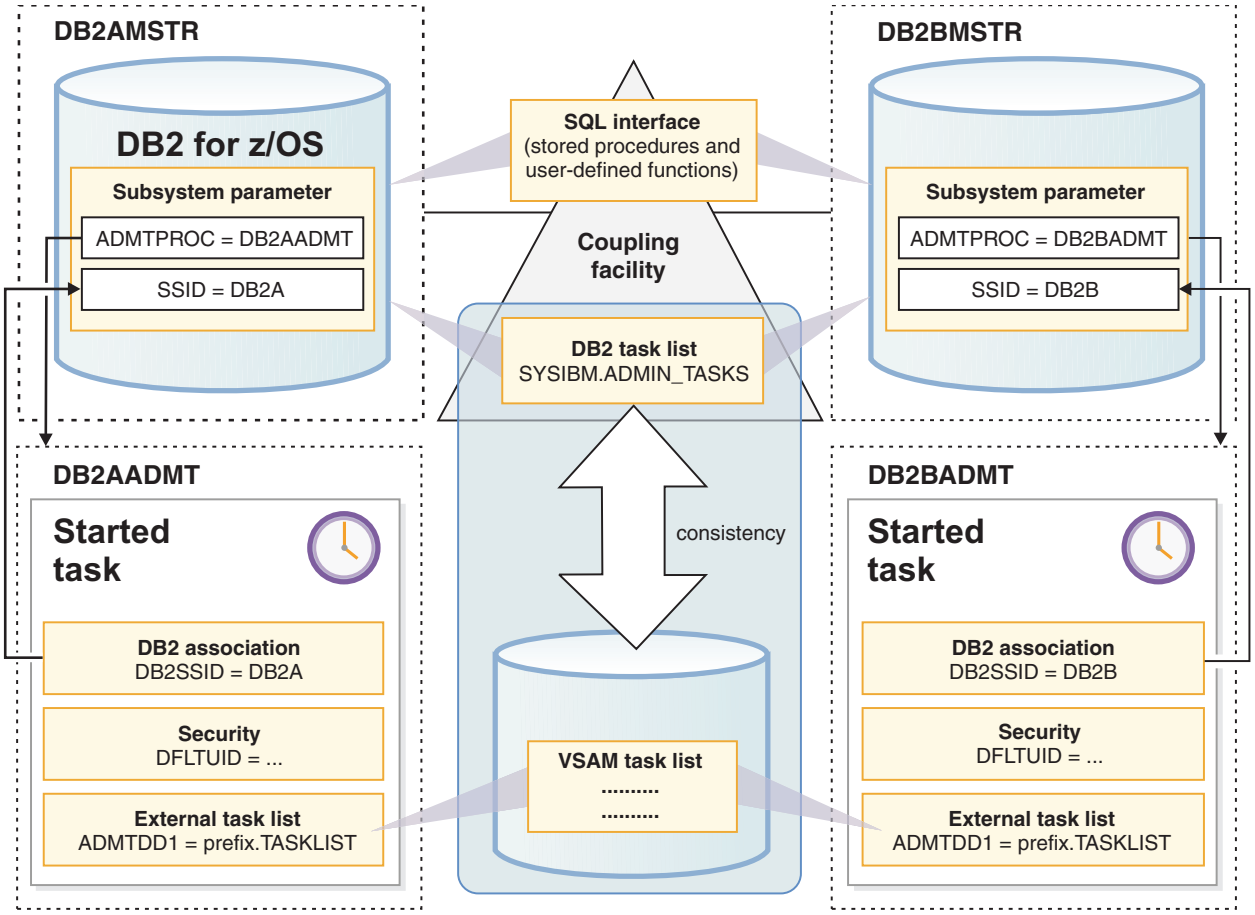


Figure 17. Administrative task schedulers in a data sharing group

Tasks are not localized to a administrative task scheduler. They can be added, removed, or executed in any of the administrative task schedulers in the data sharing group with the same result. However, you can force the task to execute on a given administrative task scheduler by specifying the associated DB2 subsystem ID in the *DB2SSID* parameter when you schedule the task. The tasks that have no affinity to a given DB2 subsystem are executed among all administrative task schedulers. Their distribution cannot be predicted.

Accounting information for stored procedure tasks

When executing a stored procedure task, the administrative task scheduler passes accounting information to the stored procedure task when establishing a Resource Recovery Services attachment facility (RRSAF) connection to DB2.

When executing a stored procedure task, the administrative scheduler sets the following information in these special registers:

- CURRENT CLIENT_USERID: The client user ID is set to the name of the execution user. If the user ID is longer than 16 characters, only the first 16 characters of the user ID are set.
- CURRENT CLIENT_APPLNAME: The client application name is set to the name of the administrative task scheduler.
- CURRENT CLIENT_ACCTNG: The DDF accounting string is set to the task name.

This information enables the stored procedure to relate its own outputs to the task execution status in the administrative task scheduler. For example, the stored procedure can save a log in a table together with the name of the task that executed it, so that you can relate the task execution status to this log.

Security guidelines for the administrative task scheduler

The administrative task scheduler uses 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 DSNTIJRT is responsible for establishing the security environment in DB2 for accessing the administrative task scheduler interface.

The following figure shows all of the security checkpoints that are associated with using the administrative task scheduler.

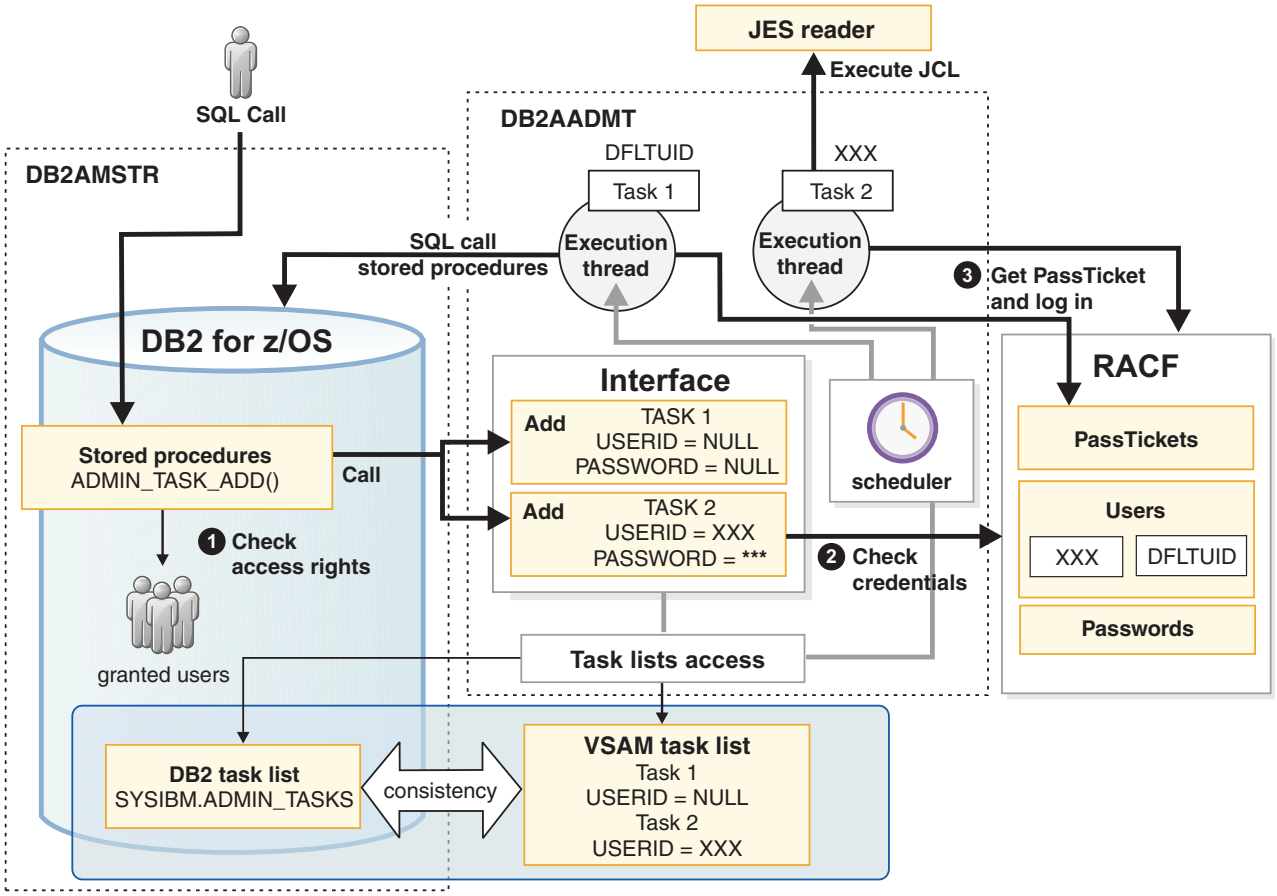


Figure 18. Security checkpoints for the administrative task scheduler

Related tasks:

Installation step 20: Set up DB2-supplied routines (DB2 Installation and Migration)

Migration step 26: Set up DB2-supplied routines (DB2 Installation and Migration)

Installation step 22: Set up the administrative task scheduler (DB2 Installation and Migration)

Migration step 29: Set up the administrative task scheduler (DB2 Installation and Migration)

Related reference:

z/OS UNIX System Services Planning

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 started task of an administrative task scheduler is associated to the STARTUID user in RACF. The administrative task scheduler runs in the security context of this user. This user, the started task user, should have access to the resources of the

administrative task scheduler. This user needs UPDATE access on the DB2 table SYSIBM.ADMIN_TASKS and write access for the VSAM data set that contains the redundant task list.

The users or groups of users who have access to the SQL interface of the administrative task 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 administrative task scheduler stored procedures and user-defined table functions. They also are granted READ access on the SYSIBM.ADMIN_TASKS table.

Each scheduled task in the administrative task scheduler is associated with an execution user who executes the task. When an execution user is not explicitly defined, the administrative task scheduler uses a default execution user, DFLTUID, that is defined in the started task. The execution threads of the administrative task scheduler switch to the security context of the execution user before executing a 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 administrative task 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 administrative task 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 administrative task 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 task list of the administrative task scheduler 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 task lists of the administrative task scheduler are protected against unauthorized use by users other than the started task execution user.

The VSAM resource (by default DSNCAT.TASKLIST, where DSNCAT is the DB2 catalog prefix) that stores the task list of the administrative task scheduler must be protected in RACF against unauthorized access. Only the 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 started tasks user has SELECT, INSERT, DELETE, or UPDATE authority on this resource. Users with EXECUTE rights on the ADMIN_TASK_LIST and ADMIN_TASK_STATUS user-defined functions have only SELECT authority on the SYSIBM.ADMIN_TASKS table.

Secure execution of tasks in the administrative task scheduler

The execution threads of the administrative task scheduler 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 administrative task scheduler when starting task execution is to switch its security context to the context of the execution user. The execution user can be explicitly identified by the *user-ID* parameter of the ADMIN_TASK_ADD stored procedure, or can be the default user.

If the task is a stored procedure, and the stored procedure must be executed in DB2 with the privileges and rights of the secondary authentication IDs of the execution user, make sure that DB2 was started with a sign-on exit routine.

If the task must run under a certain authority, including an authority that is 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 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 task scheduler executes the tasks under its default execution user.

The administrative task 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 administrative task scheduler, but the administrative task scheduler is defined as a trusted program in RACF, and is allowed to get PassTickets for any user. The administrative task 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 started task module (DSNADMT0) of the administrative task scheduler uses the pthread_security_np() function to switch users. To set a thread-level security environment for the scheduler, follow the steps in Additional steps for enabling the administrative task scheduler and administrative enablement routines (DB2 Installation and Migration).

The administrative task scheduler resources should be protected from unintended impact when executing a task. Therefore, the started task user (STARTUID), which has access to the administrative task scheduler resources, must not be used as the default execution user (DFLTUID). Also, it should not be specified in the *user-ID* parameter of the ADMIN_TASK_ADD stored procedure. The administrative task 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.

Related reference:

Processing of sign-on requests (Managing Security)

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 task execution.

The work of the administrative task scheduler is based on scheduled tasks that you define. Each task is associated with a unique task name. Up to 9999 tasks are supported in the administrative task scheduler at one time.

A scheduled task consists of a schedule and a work definition. The schedule tells the administrative task scheduler when to execute the task. You define a window of time during which execution is permitted, and either a time-based or event-based schedule that triggers the execution of the job during 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 administrative task scheduler is multi-threaded. The administrative task scheduler starts the execution of scheduled tasks, and waits for the tasks to complete. The execution of a task is delegated by the administrative task scheduler to one of its sub-threads that starts the execution, waits until the execution completes, and gathers the execution status. Each sub-thread can be used for any type of task.

The maximum number of sub-threads is determined by the MAXTHD parameter of the started task, which by default is 99. Therefore, the administrative task scheduler can execute up to 99 tasks simultaneously. To reduce the memory usage of the administrative task scheduler, reduce the number of sub-threads by specifying a lower value for the MAXTHD parameter. You specify this parameter in the JCL of the started task. The JCL has the same name as the administrative task scheduler, as defined in job DSNTIJMV.

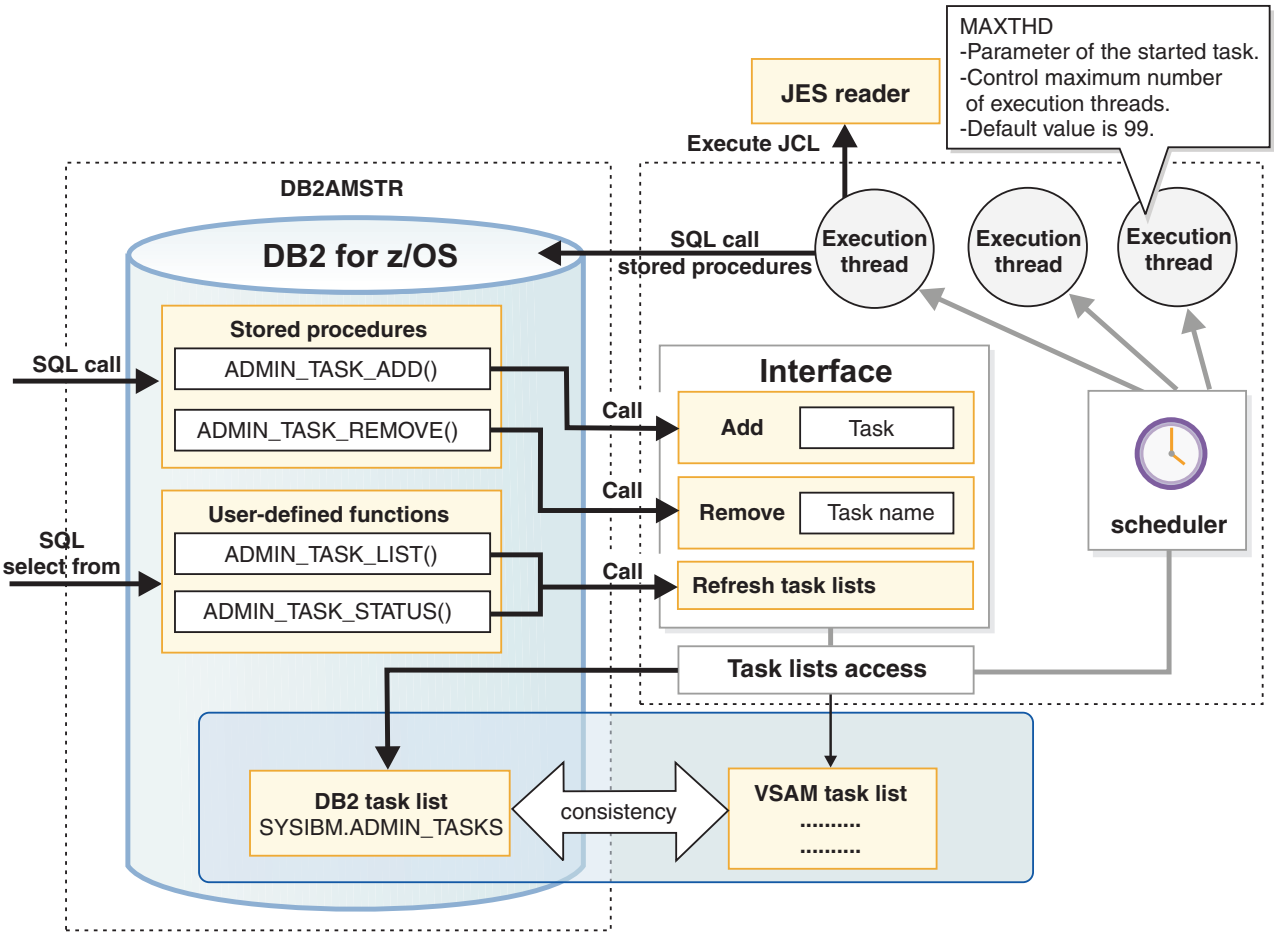


Figure 19. Multi-threading in the administrative task scheduler

The minimum permitted value for the MAXTHD parameter is 1, but this value 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 there are available sub-threads, some tasks will not start executing immediately. The administrative task scheduler tries 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.

The parameters of the started task are not positional. Place parameters in a single string separated by blank spaces.

If the execution of a task still cannot start one minute after it should have started, 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 you receive this message, increase the MAXTHD parameter value and restart the administrative task scheduler.

Related tasks:

Installation step 22: Set up the administrative task scheduler (DB2 Installation and Migration)

Related information:

DSNA677I (DB2 Messages)

DSNA678I (DB2 Messages)

Scheduling execution of a stored procedure

You can schedule a stored procedure to run at a particular time, at an interval, or when a specified event occurs. The administrative task scheduler manages these requests.

Procedure

To schedule execution of a stored procedure:

1. Add a task for the administrative task scheduler by using the ADMIN_TASK_ADD stored procedure. When you add your task, specify which stored procedure to run and when to run it. Use one of the following parameters or groups of parameters of ADMIN_TASK_ADD to control when the stored procedure is run:

Option	Description
interval	The stored procedure is to execute at the specified regular interval.
point-in-time	The stored procedure is to execute at the specified times.
trigger-task-name	The stored procedure is to execute when the specified task occurs.
trigger-task-name trigger-task-cond trigger-task-code	The stored procedure is to execute when the specified task and task result occur.

   Optionally, you can also use one or more of the following parameters to control when the stored procedure runs:

   begin-timestamp
   Earliest permitted execution time

   end-timestamp
   Latest permitted execution time

   max-invocations
   Maximum number of executions

   When the specified time or event occurs for the stored procedure to run, the administrative task scheduler calls the stored procedure in DB2.
2. Optional: After the task finishes execution, check the status by using the ADMIN_TASK_STATUS function. This function returns a table with one row that indicates the last execution status for each scheduled task. If the scheduled task is a stored procedure, the JOB_ID, MAXRC, COMPLETION_TYPE, SYSTEM_ABENDCD, and USER_ABENDCD fields contain null values. In the case of a DB2 error, the SQLCODE, SQLSTATE, SQLERRMC, and SQLERRP fields contain the information that DB2 returned from calling the stored procedure.

Related tasks:

"Adding a task" on page 223

Related reference:

ADMIN_TASK_ADD stored procedure (DB2 SQL)

ADMIN_TASK_STATUS (DB2 SQL)

Scheduling stored procedures (DB2 for z/OS Stored Procedures: Through the CALL and Beyond)

How the administrative task scheduler executes a stored procedure

You can use the administrative task scheduler to execute stored procedures at a specific time. You must first define a task for the stored procedure execution. Then, when the specified time or event occurs for the stored procedure to run, the administrative task scheduler calls the stored procedure.

Specifically, the administrative task scheduler performs the following actions:

1. The administrative task scheduler connects to the DB2 member that is specified in the task parameter *DB2SSID*. If the administrative task scheduler cannot establish a connection, it skips execution of the stored procedure and sets the last execution status to the NOTRUN state.
2. The administrative task scheduler retrieves parameter values for the stored procedure from DB2 by using the SELECT statement that is defined in the task parameter *procedure-input*. If an error occurs when the administrative task scheduler retrieves those parameter values, the administrative task scheduler:
   - Does not call the stored procedure.
   - Sets the last execution status of the task to the error code that is returned by DB2.
3. The administrative task scheduler issues an SQL CALL statement with the retrieved parameter values and a stored procedure name. The procedure name is concatenated from the task parameters *procedure-schema* and *procedure-name*. The SQL CALL statement is synchronous, and the execution thread is blocked until the stored procedure finishes execution. The administrative task scheduler sets the last execution status to the values that are returned by DB2.
4. The administrative task scheduler issues a COMMIT statement.
5. The administrative task scheduler closes the connection to DB2.

How the administrative task scheduler works with Unicode

The administrative task scheduler can 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. The execution sub-thread of the administrative task scheduler can optionally wait for the job to terminate and purge the job from the JES job list.

One execution sub-thread of the administrative task 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

In a data sharing environment, all administrative task schedulers cooperate in the execution of scheduled tasks.

However, if a task has a member affinity (for example, if the *DB2-SSID* parameter specifies the name of a DB2 member), only the administrative task scheduler that is associated with that DB2 member will execute the task. Therefore, if this administrative task scheduler is unavailable, the task will not be executed.

When a task has no member affinity (if the *DB2-SSID* parameter is a null value), the first administrative task scheduler that is available executes the task. If the task execution can complete on this administrative task scheduler, other administrative task schedulers will not try executing this task. However, if the administrative task scheduler cannot start the execution, other administrative task schedulers will try to start executing the task until one succeeds, or all fail in executing the task. An administrative task scheduler is unavailable if its associated DB2 member is not running, or if all of its execution threads are busy.

You cannot predict which administrative task scheduler will execute a task. Successive executions of the same task can be done on different administrative task schedulers.

Time zone considerations for the administrative task scheduler

The administrative task scheduler is configured to work in a given time zone.

By default, the administrative task scheduler works in the time zone that is defined for the z/OS operating system as the local time, provided that the z/OS environment variable TZ is not set. If the TZ environment variable is set, the administrative task scheduler works in the time zone that is defined for this variable. You can set the TZ environment variable either in the z/OS Language Environment® default settings, or in the CEEOPTS definition of the started task of the administrative task scheduler.

When you add a task, you must specify the values for the *begin-timestamp*, *end-timestamp*, and *point-in-time* parameters of the stored procedure in the time zone that the administrative task scheduler works in. When listing scheduled tasks, the BEGIN_TIMESTAMP, END_TIMESTAMP, and POINT_IN_TIME columns are returned with values that are in time zone for the administrative task scheduler. Likewise, when listing the status of a task, the LAST_MODIFIED, START_TIMESTAMP, and END_TIMESTAMP columns are returned with values that are in time zone for the administrative task scheduler.

Important: You must configure all of the administrative task schedulers for a data sharing environment in the same time zone. Otherwise, the scheduled tasks might run at a time when they are not intended to execute.

The following example shows how to set the time zone for Germany.

```
//CEEOPTS  DD  *
ENVAR("TZ=MEZ-1MESZ,M3.5.0,M10.5.0")
```

Related tasks:

Using the CEEOPTS DD statement (z/OS Language Environment Customization)

Related reference:

ENVAR (z/OS Language Environment Customization)

Chapter 8. Monitoring and controlling DB2 and its connections

You can control and monitor various aspects of a DB2 for z/OS environment. These operations require you to have more of an understanding of what DB2 is doing.

Related tasks:

Chapter 4, "Controlling DB2 operations by using commands," on page 203

Monitoring and analyzing DB2 performance data (DB2 Performance)

Controlling resource usage (DB2 Performance)

Setting limits for system resource usage by using the resource limit facility (DB2 Performance)

Monitoring threads and connections by using profiles (DB2 Performance)

Updating subsystem parameter values (DB2 Installation and Migration)

Controlling DB2 databases and buffer pools

You can use various commands to control DB2 databases and buffer pools. The commands that you issue depend on the type of action that you want to take.

Procedure

GUPI

To control databases and buffer pools:

Issue the START DATABASE, STOP DATABASE, or DISPLAY DATABASE commands.

START DATABASE
Makes a database or individual partitions available. Also removes pages from the logical page list (LPL).

DISPLAY DATABASE
Displays status, user, and locking information for a database.

STOP DATABASE
Makes a database or individual partitions unavailable after existing users have quiesced. DB2 also closes and deallocates the data sets.

GUPI

Related tasks:

"Monitoring databases" on page 256

"Starting databases" on page 254

"Making objects unavailable" on page 264

Related reference:

-START DATABASE (DB2) (DB2 Commands)

© Copyright IBM Corp. 1982, 2016

-DISPLAY DATABASE (DB2) (DB2 Commands)

-STOP DATABASE (DB2) (DB2 Commands)

Starting databases

Issue the START DATABASE (*) command to start all databases for which you have the STARTDB privilege.

About this task

The privilege can be explicitly granted, or it can belong implicitly to a level of authority (DBMAINT and above). The command starts the database, but not necessarily all the objects that it contains. Any table spaces or index spaces in a restricted mode remain in a restricted mode and are not started.

The START DATABASE (*) command 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). Start these databases explicitly by using the SPACENAM option. Also, the START DATABASE (*) command does not start table spaces or index spaces that have been explicitly stopped by the STOP DATABASE command.

Use the PART keyword of the START DATABASE command to start the 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.

The START DATABASE and STOP DATABASE commands can be used with the SPACENAM and PART options to control table spaces, index spaces, or partitions.

Example

GUPI

For example, the following command starts two partitions of table space DSN8S10E in the database DSN8D10A:

```
-START DATABASE (DSN8D10A) SPACENAM (DSN8S10E) PART (1,2)
```

GUPI

Related reference:

-START DATABASE (DB2) (DB2 Commands)

-STOP DATABASE (DB2) (DB2 Commands)

Starting an object with a specific status

You can start a database, table space, or an index space with a specific status that limits access to the object.

About this task

Objects can have the following status:

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.
RREPL
Read-or-replication-only. The object is available as read-only unless the program is identified as a replication program.

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 STOP DATABASE command. DB2 can also make them unavailable when it detects an error.

Procedure

GUPI

To start objects that have a specific status:

In cases when the object was explicitly stopped, you can make it available again by issuing the START DATABASE command.

Example

For example, the following command starts all table spaces and index spaces in database DSN8D10A for read-only access:

```
-START DATABASE (DSN8D10A) SPACENAM(*) ACCESS(RO)
```

The system responds with this message:

```
DSN9022I - DSNTDDIS '-START DATABASE' NORMAL COMPLETION
```

GUPI

Related reference:

-START DATABASE (DB2) (DB2 Commands)

-STOP DATABASE (DB2) (DB2 Commands)

Related information:

DSN9022I (DB2 Messages)

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.

About this task

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 of the object is taken.

Important: 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.

Procedure

GUPI

To start an object that has restrictions:

Issue the START DATABASE command with the ACCESS(FORCE) option.
This command releases most restrictions for the named objects. These objects must be explicitly named in a list following the SPACENAM option.

Example

For example:

```
-START DATABASE (DSN8D10A) SPACENAM (DSN8S10E) ACCESS(FORCE)
```

DB2 cannot process the START DATABASE ACCESS(FORCE) request if postponed-abort or indoubt units of recovery 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.

GUPI

Related tasks:

"Resolving postponed units of recovery" on page 375

Related reference:

-START DATABASE (DB2) (DB2 Commands)

Monitoring databases

You can use the DISPLAY DATABASE command 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.

Procedure

GUPI

To monitor databases:

1. Issue the DISPLAY DATABASE command as follows:

   ```
   -DISPLAY DATABASE (dbname)
   ```

   This command 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.
   - *dbname* is an 8-byte character string that indicates 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 commas. 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.

2. Optional: Use the pattern-matching character, *, in the DISPLAY DATABASE, START DATABASE, and STOP DATABASE commands. You can use the pattern-matching character in the beginning, middle, and end of the database and table space names.

3. Use additional keywords to tailor the DISPLAY DATABASE command so that you can monitor what you want:

   - 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 command 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, and then the letters DB followed by any letters.
- ONLY restricts the display to databases names that fit the criteria.

- The RESTRICT option of the DISPLAY DATABASE command limits 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.
- The ADVISORY option on the DISPLAY DATABASE command limits 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.
- The OVERVIEW option of the DISPLAY DATABASE command displays all objects within a database. This option shows each object in the database on one line, does not isolate 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 command 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 five objects are 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.

GUPI

Related reference:

Advisory or restrictive states (DB2 Utilities)

-DISPLAY DATABASE (DB2) (DB2 Commands)

Obtaining information about application programs

You can obtain various kinds of information about application programs that use particular databases, table spaces, or index spaces by using the DISPLAY DATABASE command. You can identify who or what is using an object and what locks are being held on various objects.

Identifying who and what are using an object

You can obtain information about users and applications that are using an object, and about the units of work that are accessing data.

About this task

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 that access data on behalf of the specified remote locations

Procedure

GUPI

To obtain information about objects:

Issue the DISPLAY DATABASE command with the SPACENAM option.

Example

For example, you can issue a command 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
```

GUPI

Determining which programs are holding locks on an object

You can use the DISPLAY DATABASE command to determine which programs are holding locks on an object.

Procedure

GUPI

To determine which application programs currently hold locks on the database or space:

Issue the DISPLAY DATABASE command.

Example

For example, issue a command that 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
```

Use the LOCKS ONLY keywords on the DISPLAY DATABASE command to display only spaces that have locks. You can substitute the LOCKS keyword 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 command 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
```

GUPI

Related reference:

-DISPLAY DATABASE (DB2) (DB2 Commands)

Related information:

DSNT361I (DB2 Messages)

Obtaining information about and handling pages in error

You can view information about pages that are in error.

Characteristics of pages that are in error

A page that is in error can be logically or physically 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 physical errors exist, 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 (LOB) table spaces that are 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 that is 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 logical page list (LPL) entries by issuing the DISPLAY DATABASE command with the LPL option.

Procedure

GUPI

To display a LPL entries:

Issue the DISPLAY DATABASE command with the LPL option. The ONLY option restricts the output to objects that have LPL pages.

Example

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 that are listed in the LPL PAGES column are unavailable for access.

GUPI

Related reference:

-DISPLAY DATABASE (DB2) (DB2 Commands)

Removing pages from the logical page list

Although the DB2 subsystem always attempts automated recovery of logical page list (LPL) pages when the pages are added to the LPL, you can also perform manual recovery.

About this task

You can run only the following utilities on an object with pages in the LPL:

- LOAD with the REPLACE option
- MERGECOPY
- REBUILD INDEX
- RECOVER, except for:
  - RECOVER...PAGE
  - RECOVER...ERROR RANGE
- REPAIR with the SET statement
- REPORT

Procedure

GUPI

When an object has pages on the LPL, to manually remove those pages and make them available for access when DB2 is running:

Use one of the following methods.

- Start the object with access (RW) or (RO). That command is valid even if the table space is already started.

  When you issue the START DATABASE command, message DSNI006I is displayed, indicating that LPL recovery has begun. Message DSNI022I is

displayed periodically to give you the progress of the recovery. When recovery is complete, message DSNI021I is displayed.

When you issue the START DATABASE command for a LOB table space that is defined as LOG NO, and DB2 detects that log records that are 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 is in the LPL and has RECP status. If you want to recover the logical partition by using REBUILD INDEX with the PART keyword, you must first use the START DATABASE command to clear the LPL pages.
- Run the LOAD utility with the REPLACE option on the object.
- Issue an SQL DROP statement for the object.

GUPI

Related reference:

-START DATABASE (DB2) (DB2 Commands)

LOAD (DB2 Utilities)

REBUILD INDEX (DB2 Utilities)

RECOVER (DB2 Utilities)

DROP (DB2 SQL)

Related information:

DSNI006I (DB2 Messages)

DSNI021I (DB2 Messages)

DSNI022I (DB2 Messages)

Displaying a write error page range

You can easily display the range of error pages.

Procedure

GUPI

To display the range of error pages:

Issue the DISPLAY DATABASE command.

Example

For example:

```
-DISPLAY DATABASE (DBPARTS) SPACENAM (TSPART01) WEPR
```

The preceding command returns a list that is similar to this one:

```
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 that is found in error, and PHYERRHI represents the highest page that is found in error.
- PIECE, a 3-digit integer, is a unique identifier for the data set and supports the page set that contains physical I/O errors.

GUPI

Related reference:

-DISPLAY DATABASE (DB2) (DB2 Commands)

Related information:

DSNT361I (DB2 Messages)

Making objects unavailable

You can make databases, table spaces, and index spaces unavailable by using the STOP DATABASE command.

Before you begin

If an object is in STOP-pending (STOPP) status, you must issue the START DATABASE command to remove the STOPP status.

About this task

When you issue the STOP DATABASE command for a table space, the data sets that contain that table space are closed and deallocated.

You also can stop DB2 subsystem databases (catalog, directory, and work file). After the directory is stopped, installation SYSADM authority is required to restart it.

Procedure

GUPI

To stop databases, table spaces, or index spaces:

Issue the STOP DATABASE command with the appropriate options.

Type of object that you want to stop	How to issue the STOP DATABASE command
To stop a physical partition of a table space:	Use the PART option.
To stop a physical partition of an index space:	Use the PART option.
To stop a logical partition within a nonpartitioning index that is associated with a partitioned table space:	Use the PART option.
To stop any object as quickly as possible:	Use the AT(COMMIT) option.
To stop user-defined databases:	Start database DSNDB01 and table spaces DSNDB01.DBD01, DSNDB01.SYSDBDXA, and DSNDB01.SYSLGRNX before you stop user-defined databases. If you do not do start these objects, you receive message DSNI003I. Resolve the problem and run the job again.
To stop the work file database:	Start database DSNDB01 and table spaces DSNDB01.DBD01 DSNDB01.SYSDBDXA, and DSNDB01.SYSLGRNX before you stop the work file database. If you do not do start these objects, you receive message DSNI003I. Resolve the problem and run the job again.

GUPI

Related reference:

-STOP DATABASE (DB2) (DB2 Commands)

-START DATABASE (DB2) (DB2 Commands)

Related information:

DSNI003I (DB2 Messages)

Commands to stop databases

The STOP DATABASE command has several options that you can use to control how the database stops.

GUPI

The following examples illustrate ways to use the STOP DATABASE 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.

GUPI

Related reference:

-STOP DATABASE (DB2) (DB2 Commands)

Altering buffer pools

DB2 stores buffer pool attributes in the DB2 bootstrap data set (BSDS). You can change buffer pool attributes.

About this task

Buffer pool attributes, including buffer pool sizes, sequential steal thresholds, deferred write thresholds, and parallel sequential thresholds, are initially defined during the DB2 installation process.

Introductory concepts:

Buffer pools (Introduction to DB2 for z/OS)

The role of buffer pools in caching data (Introduction to DB2 for z/OS)

Procedure

To alter buffer pool attributes:

Issue the **ALTER BUFFERPOOL** command.

Related concepts:

Buffer pool thresholds (DB2 Performance)

Related reference:

-ALTER BUFFERPOOL (DB2) (DB2 Commands)

Monitoring buffer pools

You can display the current status for one or more active or inactive buffer pools. You can also request a summary or detail report.

About this task

Buffer pools are areas of virtual storage that temporarily store pages of table spaces or indexes.

Introductory concepts:

Buffer pools (Introduction to DB2 for z/OS)

The role of buffer pools in caching data (Introduction to DB2 for z/OS)

Procedure

GUPI

To monitor buffer pools:

Issue the **DISPLAY BUFFERPOOL** command.

Example

For example:

```
-DISPLAY BUFFERPOOL(BP0)
```

This command might produce a summary report such as the following:

```
!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 !   PGFIX ATTRIBUTE -
             CURRENT                    = NO
             PENDING                    = YES
             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
```

GUPI

Related concepts:

Obtaining information about group buffer pools (DB2 Data Sharing Planning and Administration)

Related tasks:

Tuning database buffer pools (DB2 Performance)

Monitoring and tuning buffer pools by using online commands (DB2 Performance)

Related reference:

-DISPLAY BUFFERPOOL (DB2) (DB2 Commands)

Related information:

DSNB401I (DB2 Messages)

Controlling user-defined functions

User-defined functions are extensions to the SQL language, which you can invoke in an SQL statement wherever you can use expressions or built-in functions.

About this task

Introductory concepts:

User-defined functions (Introduction to DB2 for z/OS)

User-defined functions, like stored procedures, run in WLM-established address spaces. Because new functions are sometimes added to DB2, it is best to fully qualify references to user-defined functions, or set the value of the CURRENT PATH special register, so that the references resolve to the correct function in the event of a name conflict.

Procedure

GUPI

To control user-defined functions:

Issue the appropriate command for the action that you want to take.

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.

GUPI

Related concepts:

Sample user-defined functions (DB2 SQL)

Function resolution (DB2 SQL)

Related tasks:

"Monitoring and controlling stored procedures" on page 320

Defining a user-defined function (Managing Security)

Related reference:

-START FUNCTION SPECIFIC (DB2) (DB2 Commands)

-DISPLAY FUNCTION SPECIFIC (DB2) (DB2 Commands)

-STOP FUNCTION SPECIFIC (DB2) (DB2 Commands)

SET PATH (DB2 SQL)

CURRENT PATH (DB2 SQL)

Starting user-defined functions

You can activate external functions that are stopped by using the DB2 START FUNCTION SPECIFIC command.

About this task

You cannot start built-in functions or user-defined functions that are sourced on another function.

Procedure

GUPI

To activate all or a specific set of stopped external functions:

Issue the START FUNCTION SPECIFIC command.

Example

For example, assume that you want to start functions USERFN1 and USERFN2 in the PAYROLL schema. Issue the following command:

```
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
```

GUPI

Related reference:

-START FUNCTION SPECIFIC (DB2) (DB2 Commands)

Related information:

DSNX973I (DB2 Messages)

Monitoring user-defined functions

You can monitor external user-defined functions by using the DISPLAY FUNCTION SPECIFIC command. This command displays statistics about the functions and lists the range of functions that are stopped because of a STOP FUNCTION SPECIFIC command.

About this task

This DB2 command displays statistics about external user-defined functions that are 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 reflects 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.

Procedure

GUPI

To monitor user-defined functions:

Issue the DISPLAY FUNCTION SPECIFIC command.

Example

For example, to display information about functions in the PAYROLL schema and the HRPROD schema, issue this command:

```
-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
```

GUPI

Related reference:

-DISPLAY FUNCTION SPECIFIC (DB2) (DB2 Commands)

-STOP FUNCTION SPECIFIC (DB2) (DB2 Commands)

Related information:

DSNX975I (DB2 Messages)

Stopping user-defined functions

You can prevent DB2 from accepting SQL statements with invocations of specific functions by using the STOP FUNCTION SPECIFIC command. This command does not prevent SQL statements with invocations of the functions from running if they have already been queued or scheduled by DB2.

About this task

You cannot stop built-in functions or user-defined functions that are sourced on another function.

Procedure

GUPI

To stop access to all or a specific set of external functions:

Issue the STOP FUNCTION SPECIFIC command.

Example

For example, issue a command like the following one, 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
```

While the STOP FUNCTION SPECIFIC command is in effect, attempts to execute the stopped functions are queued.

GUPI

Related reference:

-STOP FUNCTION SPECIFIC (DB2) (DB2 Commands)

Related information:

DSNX974I (DB2 Messages)

Controlling DB2 utilities

DB2 utilities are classified into two groups: online and stand-alone.

About this task

The online utilities require DB2 to be running and can be controlled in several different ways. The stand-alone utilities do not require DB2 to be running, and they can be controlled only by means of JCL.

Related concepts:

DB2 online utilities (DB2 Utilities)

DB2 stand-alone utilities (DB2 Utilities)

Starting online utilities

DB2 utilities can dynamically create object lists from a pattern-matching expression and can dynamically allocate the data sets that are required to process those objects.

Procedure

To start a DB2 utility:

Prepare an appropriate set of JCL statements for a utility job. The input stream for that job must include DB2 utility control statements.

Monitoring and changing online utilities

You can monitor the status of online utilities, change parameter values of some utilities, and terminate a utility job prior to completion.

About this task

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. The following table shows the compatibility of utility types and object status.

Table 29. Compatibility of utility types and object status

Utility type	Object access
Read-only	RO
All	RW (default access type for an object)
DB2	UT

Procedure

GUPI

To monitor and change an online utility:

1. Issue the appropriate command for the action that you want to take.

   ALTER UTILITY
   Alters parameter values of an active REORG or REBUILD utility.

   DISPLAY UTILITY
   Displays the status of utility jobs.

   TERM UTILITY
   Terminates a utility job before its normal completion.

2. To change the status of an object, use the START DATABASE command with the ACCESS option to start the object with a new status. For example:

   ```
   -START DATABASE (DSN8D61A) ACCESS(RO)
   ```

GUPI

Related concepts:

DB2 online utilities (DB2 Utilities)

Related reference:

-START DATABASE (DB2) (DB2 Commands)

Controlling DB2 stand-alone utilities

You use JCL to run the DB2 stand-alone utilities, most of which can run while DB2 is running.

Procedure

GUPI

To run a DB2 stand-alone utility:

1. Stop the table spaces and index spaces that are the object of the utility job. If you do not do this, you might receive inconsistent output.
2. If the utility is one that requires that DB2 be stopped during utility execution, use this command:

   ```
   -STOP DB2 MODE (FORCE)
   ```
3. If the utility is one that requires that DB2 be running during utility execution and if DB2 is not running, issue this command:

   ```
   -START DB2
   ```
4. Create a JCL job that includes the utility control statement with code specific data set names and associated parameters for your utility.

GUPI

Stand-alone utilities

Some stand-alone utilities can be run only by means of JCL.

These stand-alone utilities are:

- 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.

GUPI

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.

GUPI

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 the STOP DB2 MODE(QUIESCE) command 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 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.

Related concepts:

DB2 stand-alone utilities (DB2 Utilities)

Controlling the IRLM

The internal resource lock manager (IRLM) subsystem manages DB2 locks.

About this task

The particular IRLM to which DB2 is connected is specified in the DB2 load module for subsystem parameters. The particular IRLM 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.

Each IMS and DB2 subsystem must use a separate instance of IRLM.

In a data sharing environment, the IRLM handles global locking, and each DB2 member has its own corresponding IRLM.

Procedure

GUPI

To control the IRLM:

Issue the appropriate z/OS command for the action that you want to take.
In each command description, *irlmproc* is the IRLM procedure name and *irlmnm* is the IRLM subsystem name.

MODIFY *irlmproc*,**ABEND,DUMP**
Abnormally terminates the IRLM and generates a dump.

MODIFY *irlmproc*,**ABEND,NODUMP**
Abnormally terminates the IRLM but does not generate a dump.

MODIFY *irlmproc*,**DIAG**
Initiates diagnostic dumps for IRLM subsystems in a data sharing group when a delay occurs.

MODIFY *irlmproc*,**SET**
Dynamically sets the maximum amount of private virtual (PVT) storage or the number of trace buffers that are 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, and XCF).

TRACE CT,ON,COMP=*irlmnm*,**SUB=(***subname***)**
Starts IRLM tracing for a single subtype.

GUPI

Related concepts:

IRLM names (DB2 Installation and Migration)

Related reference:

z/OS IRLM commands (DB2 Commands)

z/OS commands that operate on IRLM

You can use several z/OS commands to modify and monitor the IRLM connection.

GUPI

MODIFY *irlmproc*,**SET,PVT=***nnn*
Sets the maximum amount of private virtual (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 this command 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 that are used for this IRLM.

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 private virtual (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.

Each IMS and DB2 subsystem must use a separate instance of IRLM.

GUPI

Related reference:

z/OS IRLM commands (DB2 Commands)

Starting the IRLM

The internal resource lock manager (IRLM) must be available when you start DB2.

About this task

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.

Consider starting the IRLM manually if you are having problems starting DB2 for either of the following 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.

Procedure

GUPI

If an automatic IRLM start has not been specified, to start IRLM:

Issue the z/OS START *irlmproc* command.
When started, the IRLM issues this message to the z/OS console:

```
DXR117I irlmnm INITIALIZATION COMPLETE
```

GUPI

Stopping the IRLM

If the internal resource lock manager (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.

Procedure

To stop IRLM:

Issue the z/OS STOP *irlmproc* command.
If you try to stop the IRLM while DB2 or IMS is still using it, you get the following message:

GUPI

```
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  -  HI-PVT   5402K
```

Results

Your DB2 subsystem will abend. An IMS subsystem that uses 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 routine. The event exit routine must be in the link list. It is part of the IRLM DXRRL183 load module. The event exit routine ensures that the IRLM name is defined to z/OS when ARM restarts IRLM on a target z/OS system that is different from the failing z/OS system. The IRLM element name that is 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, this command does not prevent ARM from restarting DB2, and, if you set the automatic restart manager to restart IRLM, DB2 automatically starts IRLM.

GUPI

Monitoring threads

You monitor threads by using the DB2 DISPLAY THREAD command, which displays current information about the status of threads.

About this task

GUPI

The DISPLAY THREAD command displays information about:
- Threads that are processing locally
- Threads that are processing distributed requests
- Stored procedures or user-defined functions that the thread is executing
- Parallel tasks

GUPI

Related tasks:

Managing DB2 threads (DB2 Performance)

Related reference:

-DISPLAY THREAD (DB2) (DB2 Commands)

Related information:

DSNV401I (DB2 Messages)

Types of threads

Threads are an important resource within a DB2 subsystem. A *thread* is a structure that describes a connection made by an application and traces its progress in the DB2 subsystem.

Allied threads

Allied threads are threads that are connected to DB2 from TSO, batch, IMS, CICS, CAF, or RRSAF.

Database Access Threads

Distributed *database access threads* (sometimes called DBATs) are threads that are connected through a network to access data at a DB2 server on behalf distributed requesting systems. Database access threads are created in the following situations:

- When new connections are accepted from remote requesters
- When DB2 is configured in INACTIVE mode, and a new request is received from a remote requester and no pooled database access thread is available to service the new request

Database access threads can operate in ACTIVE or INACTIVE mode. The mode used for database access threads is controlled by the value of the CMTSTAT subsystem parameter.

INACTIVE mode

When the value of the CMTSTAT subsystem parameter is INACTIVE, a database access thread can be active or pooled. When a database access thread 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.

INACTIVE mode database access threads are terminated under any of the following conditions:

- After processing 200 units of work.
- After being idle in the pool for the amount of time specified by the value of the IDTHTOIN subsystem parameter.

However, the termination of an INACTIVE mode thread does not prevent another database access thread from being created to meet processing demand, as long as the value of the MAXDBAT subsystem parameter has not been reached.

ACTIVE mode

When the value CMTSTAT subsystem parameter is ACTIVE, a database access thread is always active from initial creation to termination.

Related reference:

DDF THREADS field (CMTSTAT subsystem parameter) (DB2 Installation and Migration)

MAX REMOTE ACTIVE field (MAXDBAT subsystem parameter) (DB2 Installation and Migration)

IDLE THREAD TIMEOUT field (IDTHTOIN subsystem parameter) (DB2 Installation and Migration)

Output of the DISPLAY THREAD command

The output of the DISPLAY THREAD command can also indicate that a system quiesce is in effect as a result of the ARCHIVE LOG command.

The DISPLAY THREAD command allows you to select which type of information you want to include in the display by using one or more of the following:

- Active, indoubt, postponed-abort, or pooled threads
- Allied threads that are associated with the address spaces whose connection-names are specified
- Allied threads
- Distributed threads
- Distributed threads that are associated with a specific remote location
- Detailed information about connections with remote locations
- A specific logical unit of work ID (LUWID)

The information that is returned by the DISPLAY THREAD command reflects a dynamic status. By the time the information is displayed, the status might 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.

More information about how to interpret this output can be found in the topics describing the individual connections and in the description of message DSNV408I, which is part of message DSNV401I.

Related tasks:

"Archiving the log" on page 348

Related reference:

-DISPLAY THREAD (DB2) (DB2 Commands)

Related information:

DSNV401I (DB2 Messages)

Displaying information about threads

Use the DISPLAY THREAD TYPE(INDOUBT) command to find information about allied and database access indoubt threads. This command provides information about threads where DB2 is a participant, a coordinator, or both.

About this task

GUPI

The TYPE(INDOUBT) option tells you which systems still need indoubt resolution and provides the LUWIDs that 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 that are 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 rollback decision from the coordinator, and the connection is still active, DB2 considers the thread active.

GUPI

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, the latter system backed out the thread, and the first system must therefore do the same.

Related concepts:

"Output of the DISPLAY THREAD command" on page 279

Related reference:

-DISPLAY THREAD (DB2) (DB2 Commands)

Related information:

DSNV401I (DB2 Messages)

Displaying information by location

You can use the DISPLAY THREAD command to display thread information for particular locations.

Procedure

GUPI

To display information by location:

Issue the DISPLAY THREAD command with the LOCATION option, followed by a list of location names.

Example

For example, you can specify an asterisk (*) after the THREAD and LOCATION options:

```
-DISPLAY THREAD(*) LOCATION(*) DETAIL
```

When you issue this command, DB2 returns messages like the following:

```
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
     (  1)2002:91E:610:1::5 4
```

```
    V447-INDEX SESSID           A ST TIME
    V448-(  1) 446:1300 5       W S2 9802812045091
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I - DSNVDT '-DISPLAY THREAD' 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 that involves DCE services (D). Currently, DB2 supports the optional use of DCE services to authenticate remote users.
2	The A (active) column contains an asterisk that indicates 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 partner. If the RDBNAME is not known, the location column contains either an SNA LUNAME or IP address.
5	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.

GUPI

Related reference:

-DISPLAY THREAD (DB2) (DB2 Commands)

Related information:

DSNV401I (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.

About this task

The LUNAME is enclosed by the less-than (<) and greater-than (>) symbols. The IP address can be in the dotted decimal or colon hexadecimal format.

DB2 uses the one of the following formats in messages that display information about non-DB2 requesters:

- LUNAME notation
- Dotted decimal format
- Colon hexadecimal format

Procedure

GUPI To display information for non-DB2 locations:

Issue the DISPLAY THREAD command with the LOCATION option.

Example

For example, if you want to display information about a non-DB2 database management system (DBMS) with the LUNAME of LUSFOS2, enter the following command:

```
-DISPLAY THREAD (*) LOCATION (<LUSFOS2>)
```

GUPI

Related reference:

-DISPLAY THREAD (DB2) (DB2 Commands)

Related information:

DSNV401I (DB2 Messages)

Displaying conversation-level information about threads

Use the DISPLAY THREAD command with the LOCATION and DETAIL options to display information about conversation activity when distribution information is displayed for active threads.

About this task

The DETAIL option has no effect on the display of indoubt threads.

Procedure

GUPI

To display conversation-level information about threads:

Issue the DISPLAY THREAD command with the LOCATION and DETAIL options:

```
-DISPLAY THREAD(*) LOCATION(*) DETAIL
```

DB2 returns the following messages, which indicate 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
     (  1)USIBMSTODB22-LUND1
    V447--INDEX SESSID           A ST TIME
    V448--(  1) 0000000000000000 N 1  A1 2  9015816504776
   TSO      RA *    11 SYSADM       SYSADM   DSNESPRR 001A      15
    V445-STLDRIV.SSLU.A23555366A29=15 ACCESSING DATA FOR
     (  1)123.34.101.98
    V447--INDEX SESSID           A ST TIME
```

```
    V448--(  1) 446:3171 3        S2    9015611253108
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I - DSNVDT '-DISPLAY THREAD' 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 the network stack (VTAM or TCP/IP) or in DB2. A value of W indicates that the thread is suspended in DB2 and is waiting for notification from the network that the event has completed. A value of N indicates that control of the conversation is in the network stack.
2	The information on this line is part of message DSNV448I. The A in the conversation ST (status) column for a serving site indicates that a conversation is being allocated in DB2. 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. 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.

GUPI

Related reference:

-DISPLAY THREAD (DB2) (DB2 Commands)

Related information:

DSNV401I (DB2 Messages)

Displaying threads by LUWID

Use the optional LUWID keyword, which is valid only when DDF is started, to display threads by logical unit of work identifiers. The LUWIDs are assigned to the thread by the site that originated the thread.

About this task

GUPI You can use an asterisk (*) in an LUWID keyword, like you can use an asterisk in a LOCATION name. For example, issue the command `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."

Use the DETAIL keyword with the DISPLAY THREAD LUWID command to show the status of every conversation that is connected to each displayed thread, and to indicate whether a conversation is using DRDA access.

Procedure

To display threads by LUWIDs:

Issue the DISPLAY THREAD command with the following options:

```
-DISPLAY THREAD(*) LUWID (luwid) DETAIL
```

DB2 returns the following message:

```
-DISPLAY THREAD(*) 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
    (  1)USIBMSTODB22-LUNSITE1
   V447--INDEX SESSID           A ST TIME
   V448--(  1) 00C3F4228C5A244C   S2 2  8929612225354
  DISPLAY ACTIVE REPORT COMPLETE
  DSN9022I - DSNVDT '-DISPLAY THREAD' 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: `-DISPLAY THREAD(*) 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 a DRDA access conversation.

GUPI

Displaying threads by type

Use the DISPLAY THREAD command to display threads by type. For example, you can display only the active threads that are executing a stored procedure or user-defined function.

Procedure

GUPI

To display threads by type:

Issue the DISPLAY THREAD command with the TYPE keyword. For example:

```
-DISPLAY THREAD(*) TYPE(PROC)
```

GUPI

Monitoring all DBMSs in a transaction

Use the DETAIL keyword of the DISPLAY THREAD command to monitor all of the requesting and serving database management systems (DBMSs) that are involved in a transaction.

About this task

GUPI

For example, you could monitor an application that runs at USIBMSTODB21 requesting information from USIBMSTODB22, which must establish conversations with secondary servers USIBMSTODB23 and USIBMSTODB24 to provide the requested information. The following figure 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.

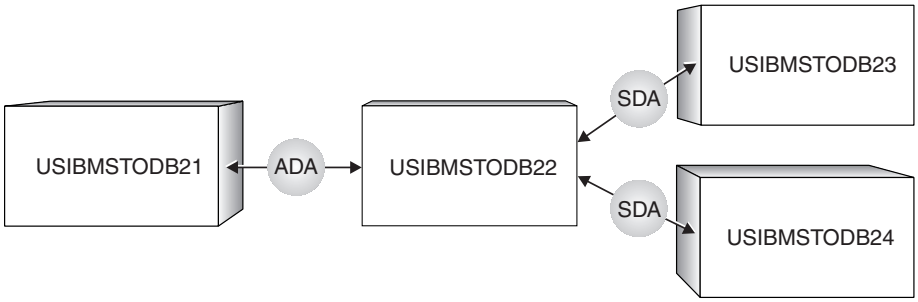


Figure 20. Example of a DB2 transaction that involves four sites

The application that runs at USIBMSTODB21 is connected to a server at USIBMSTODB22 by using DRDA access. If you enter the DISPLAY THREAD command with the DETAIL keyword from USIBMSTODB21, you receive the following output:

```
-DISPLAY THREAD(*) 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
     (  1)USIBMSTODB22-SSURLU
    V447--INDEX SESSID           A ST TIME
    V448--(  1) 0000000300000004 N R2 9015611253116
   DISPLAY ACTIVE REPORT COMPLETE
  11:26:23 DSN9022I - DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
```

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 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 following output:

```
-DISPLAY THREAD(*) 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
     (  1)USIBMSTODB21-SSLU
```

```
   V444-STLDRIV.SSLU.A23555366A29=2 ACCESSING DATA AT
    (  2)USIBMSTODB23-OSSLU
    (  3)USIBMSTODB24-OSSURLU
   V447--INDEX SESSID           A ST TIME
   V448--(  1) 0000000300000004   S2 9015611253108
   V448--(  2) 0000000600000002   S1 9015611253077
   V448--(  3) 0000000900000005 N R1 9015611253907
  DISPLAY ACTIVE REPORT COMPLETE
 11:26:34 DSN9022I - DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
```

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 following output:

```
-DISPLAY THREAD(*) 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
         (  1)USIBMSTODB22-SSURLU
    V447--INDEX SESSID           A ST TIME
    V448--(  1) 0000000600000002 W R1 9015611252369
   DISPLAY ACTIVE REPORT COMPLETE
  11:27:25 DSN9022I - DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
```

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 following output:

```
-DISPLAY THREAD(*) 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
     (  1)USIBMSTODB22-SSURLU
    V447--INDEX SESSID           A ST TIME
    V448--(  1) 0000000900000005   S1 9015611253075
   DISPLAY ACTIVE REPORT COMPLETE
  11:27:32 DSN9022I - DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
```

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 be taking a long time. To determine whether the conversation is hung, issue the DISPLAY THREAD command again and compare the new timestamp to the timestamps from previous output messages. If the timestamp is changing, but the status is not changing, 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 time, you can use the CANCEL DDF THREAD command if the thread is in DB2 (whether active or suspended). If the thread is in VTAM, you can use the VARY NET TERM command.

GUPI

Related tasks:

"Canceling threads" on page 319

Related reference:

-CANCEL THREAD (DB2) (DB2 Commands)

-DISPLAY THREAD (DB2) (DB2 Commands)

Related information:

DSNV401I (DB2 Messages)

Controlling connections

The method that you use to control connections between DB2 and another subsystem or environment depends on the subsystem or environment that is involved.

Controlling TSO connections

z/OS does not provide commands for controlling or monitoring a TSO connection to DB2.

About this task

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 a z/OS console or an administrator's TSO session.

Related tasks:

"Issuing commands from TSO terminals" on page 205

"Monitoring threads" on page 277

Related reference:

-DISPLAY THREAD (DB2) (DB2 Commands)

Monitoring TSO and CAF connections

To display information about connections that use the TSO attachment facility and the call attachment facility (CAF), issue the DISPLAY THREAD command.

About this task

GUPI

The following table summarizes how the output for the DISPLAY THREAD command differs for a TSO online application, a TSO batch application, a QMF™ session, and a call attachment facility application.

Table 30. Differences in DISPLAY THREAD information for different environments

Connection	Name	AUTHID	Corr-ID[1]	Plan[1]
DSN (TSO Online)	TSO	Logon ID	Logon ID	RUN .. Plan(x)

Table 30. Differences in DISPLAY THREAD information for different environments (continued)

Connection	Name	AUTHID	Corr-ID[1]	Plan[1]
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

Notes:

1. After the application connects to DB2 but before a plan is allocated, this field is blank.

The name of the connection can have one of the following values:

Name Connection to
TSO Program that runs in TSO foreground
BATCH
Program that runs in TSO background
DB2CALL
Program that uses the call attachment facility and that runs in the same address space as a program that uses the TSO attachment facility

The correlation ID, *corr-id*, is either the foreground authorization ID or the background job name.

The following command displays information about TSO and CAF threads, including those threads that process 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	Description
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 21. DISPLAY THREAD output that shows TSO and CAF connections

GUPI

Related tasks:

"Monitoring threads" on page 277

Related reference:

-DISPLAY THREAD (DB2) (DB2 Commands)

Related information:

DSNV401I (DB2 Messages)

Disconnecting from DB2 while under TSO

Several conditions can cause the TSO connection to DB2 to end.

About this task

GUPI

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, or you enter END or RETURN when using any of them:

- SQL statements using SPUFI
- DCLGEN
- BIND/REBIND/FREE
- RUN

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
```

GUPI

Controlling CICS connections

Certain CICS attachment facility commands can be entered from a CICS terminal to control and monitor connections between CICS and DB2.

About this task

GUPI

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.

GUPI

Related reference:

CICS attachment facility commands (DB2 Commands)

Related information:

Overview of the CICS DB2 interface (CICS DB2 Guide)

Connecting from CICS

You can start a connection to DB2 at any time after CICS initialization by using the CICS attachment facility. The CICS attachment facility is a set of DB2-provided modules that are loaded into the CICS address space.

Procedure

GUPI To connect to DB2, use one of the following approaches:

- Issue the following command to start the attachment facility:

  ```
  DSNC STRT ssid
  ```

  For *ssid*, specify a DB2 subsystem ID to override the value that is specified in the CICS INITPARM macro.
- Start the attachment facility automatically at CICS initialization by using a program list table (PLT). GUPI

Restarting CICS

One function of the CICS attachment facility is to keep data synchronized between the two systems.

About this task

If DB2 completes phase 1 but does not start phase 2 of the commit process, the units of recovery that are 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 knowing about it.

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.

Procedure

To restart CICS:

You must auto-start CICS (START=AUTO in the DFHSIT table) to obtain all necessary information for indoubt thread resolution that is available from its log. Do not perform a cold start. You specify the START option in the DFHSIT table. If CICS has requests active in DB2 when a DB2 connection terminates, the corresponding CICS tasks might remain suspended even after CICS is reconnected to DB2. Purge those tasks from CICS by using a CICS-supplied transaction such as:

GUPI

```
CEMT SET TASK(nn) FORCE
```

GUPI

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 have to manually recover these indoubt units of recovery.

Related concepts:

CICS Transaction Server for z/OS Supplied Transactions

Related tasks:

"Recovering CICS-DB2 indoubt units of recovery" on page 293

Related information:

Resource definition (CICS Transaction Server for z/OS)

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.

About this task

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 one is not already available for the specific DB2 plan.

Related concepts:

CICS Transaction Server for z/OS DB2 Guide

Monitoring CICS threads

No operator intervention is required for connecting applications; CICS handles the threads dynamically. You can monitor threads by using CICS attachment facility commands or DB2 commands.

About this task

GUPI

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 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 that is associated with the transaction (8 characters).
- PLAN/TRAN name (8 characters).
- A or I (one 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 destination
```

GUPI

Displaying CICS-DB2 indoubt units of recovery:

You can display a list of CICS-DB2 indoubt units of recovery.

About this task

GUPI

Procedure

To display a list of indoubt units of recovery:

Issue the DISPLAY THREAD command. For example:

```
-DISPLAY THREAD (connection-name) TYPE (INDOUBT)
```

Results

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 '-DISPLAY THREAD' NORMAL COMPLETION
```

GUPI

This output is part of message DSNV401I.

Related reference:

-DISPLAY THREAD (DB2) (DB2 Commands)

Related information:

DSNV401I (DB2 Messages)

Recovering CICS-DB2 indoubt units of recovery:

You can recover CICS-DB2 indoubt units of recovery.

Procedure

GUPI To recover an indoubt unit of recovery

Issue 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. The *correlation-id* is the correlation ID of the thread to be

recovered. You can determine the correlation ID by issuing the command DISPLAY THREAD. Your choice for the ACTION parameter indicates whether to commit or roll back the associated unit of recovery.
One of the following messages might be issued after you use the RECOVER command:

```
DSNV414I - THREAD correlation-id COMMIT SCHEDULED
DSNV415I - THREAD correlation-id ABORT SCHEDULED
```

GUPI

Related concepts:

"Multiple system consistency" on page 379

Displaying CICS postponed units of recovery:

You can display a list of CICS postponed units of recovery.

Procedure

GUPI

To display a list of postponed units of recovery:

Issue the DISPLAY THREAD command. For example:

```
-DISPLAY THREAD (connection-name) TYPE (POSTPONED)
```

Results

The command produces messages similar to the following:

```
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 '-DISPLAY THREAD' NORMAL COMPLETION
```

GUPI

This output is part of message DSNV401I.

Related reference:

-DISPLAY THREAD (DB2) (DB2 Commands)

Related information:

DSNV401I (DB2 Messages)

Disconnecting CICS applications

To disconnect a particular CICS transaction from DB2, you must abnormally terminate the transaction.

Procedure

GUPI

To disconnect a CICS application from DB2, use one of the following methods:

- The DB2 command CANCEL THREAD can be used to cancel a particular thread. CANCEL THREAD requires that you know the *token* for any thread that you want to cancel. Enter the following command to cancel the thread that is identified by the token as indicated in the display output.

  ```
  -CANCEL THREAD(46)
  ```

  When you issue the CANCEL THREAD command 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 that are 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.

GUPI

Related concepts:

CICS Transaction Server for z/OS DB2 Guide

Disconnecting from CICS

To disconnect the DB2 attachment to CICS, you can do an orderly disconnection or a forced disconnection.

About this task

Performing an orderly termination from CICS:

Orderly termination is recommended whenever possible. An orderly termination of the connection allows each CICS transaction to terminate before thread subtasks are detached. Therefore, no indoubt units of recovery should exist when you reconnect.

Procedure

GUPI To perform an orderly termination, use one of the following methods:

- Enter the DSNC STOP QUIESCE command. CICS and DB2 remain active.

  For example, the following command 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)
  ```

  The following 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, the following message is added to the output from the CICS job:

  ```
  DSNC025I THE ATTACHMENT FACILITY IS INACTIVE
  ```
- Enter the CICS command CEMT PERFORM SHUTDOWN. During program list table (PLT) processing, the CICS attachment facility is also named to shut down. DB2 remains active. For information about this command, see CICS shutdown (CICS Transaction Server for z/OS).

- Enter the DB2 command CANCEL THREAD. The thread terminates abnormally.

GUPI

Performing a forced termination from CICS:

Although forced termination is not recommended, at times you might need to force the connection to end.

About this task

GUPI A forced termination of the connection can abnormally terminate CICS transactions that are connected to DB2. Therefore, indoubt units of recovery can exist at reconnect.

Procedure

A forced termination occurs under the following circumstances:

- 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.

  This message appears when the stop process starts and frees the entering terminal (option FORCE):

  ```
  DSNC022I THE ATTACHMENT FACILITY STOP FORCE IS PROCEEDING
  ```

  DB2 and CICS remain active.
- You enter the CICS command CEMT PERFORM SHUTDOWN IMMEDIATE. For information about this command, see CICS shutdown (CICS Transaction Server for z/OS). 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. The CICS transaction overflows to the pool. The transaction issues an intermediate commit. The thread is terminated at commit time, and further DB2 access is not allowed.

Results

When the stop process ends and the connection is terminated, the following message is added to the output from the CICS job:

```
DSNC025I THE ATTACHMENT FACILITY IS INACTIVE
```

GUPI

Controlling IMS connections

You use IMS operator commands to control and monitor the connection to DB2.

About this task

/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.

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.

Related reference:

IMS commands (DB2 Commands)

Related information:

IMS commands

Connections to the IMS control region

IMS makes one connection to its control region from each DB2 subsystem. IMS can make the connection either automatically or in response to a command.

Connections are made in the following ways:

- 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 /START SUBSYS *ssid* command, 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, when DB2 comes up, DB2 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, IMS cannot reconnect automatically. You must make the connection by using the /START SUBSYS command.

The following messages can be produced when IMS attempts to connect a DB2 subsystem. In each message, *imsid* is the IMS connection name.

- 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 IMS master terminal:

  ```
  DSNM003I IMS/TM imsid FAILED TO CONNECT TO SUBSYSTEM ssnm
           RC=00   imsid
  ```

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.

IMS thread attachment

IMS connection *threads* describe the IMS application connection, traces progress, processes resource functions, and delimits accessibility to DB2 resources and services.

Execution of the first SQL statement of the program 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.

Duplicate IMS correlation IDs

Under certain circumstances, two threads can have the same correlation ID.

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#.psbname*) 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 that are 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, which is 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 4 bytes contain the scheduling number, and the last 4 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.

Displaying IMS attachment facility threads

You use the DISPLAY THREAD command to display IMS attachment facility threads.

About this task

GUPI

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 non-message-driven program. The following table summarizes these differences.

Table 31. 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 connects to DB2 but before sign-on processing completes, this field is blank.
2. After sign-on processing completes but before a plan is 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 '-DISPLAY THREAD' NORMAL COMPLETION
```

Key	Description
1	This is a message-driven BMP.
2	This thread has completed sign-on processing, but a DB2 plan has not been allocated.

Figure 22. DISPLAY THREAD output showing IMS connections

GUPI

Terminating IMS attachment facility threads

When an application terminates, IMS invokes an exit routine to disconnect the application from DB2. You cannot terminate a thread without causing an abend in the IMS application with which it is associated.

Procedure

To terminate an IMS application, use one of these methods:

- Terminate the application.

  The IMS commands /STOP REGION *reg*# ABDUMP or /STOP REGION *reg*# CANCEL can be used to terminate an application that runs in an online environment. For an application that runs in the DL/I batch environment, the z/OS command CANCEL can be used.
- Use 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 that you want to cancel. Enter the following command to cancel the thread that is identified by a token in the display output:

  ```
  -CANCEL THREAD(46)
  ```

  When you issue the CANCEL THREAD command, that thread is scheduled to be terminated in DB2.

Related information:

IMS commands

Displaying IMS-DB2 indoubt units of recovery

You can display information about threads whose status is indoubt by using the DB2 DISPLAY THREAD command.

About this task

GUPI One function of the thread that connects 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, IMS might 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.

Procedure

To display a list of indoubt units of recovery:

Issue the DISPLAY THREAD command. For example:

```
-DISPLAY THREAD (imsid) TYPE (INDOUBT)
```

This command produces messages similar to the following:

```
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 '-DISPLAY THREAD' NORMAL COMPLETION
```

GUPI

Related tasks:

Chapter 10, "Restarting DB2 after termination," on page 363

Related reference:

-DISPLAY THREAD (DB2) (DB2 Commands)

Related information:

DSNV401I (DB2 Messages)

Recovering IMS-DB2 indoubt units of recovery

When you determine that indoubt units of recovery exist, you recover from this situation by using the DB2 RECOVER INDOUBT command.

Procedure

To recover an indoubt unit:

Issue one of the following commands.

GUPI In each command, *imsid* is the connection name, and *pst#.psbname* is the correlation ID that is listed by the command DISPLAY THREAD. Your choice of the ACTION parameter tells whether to commit or roll back the associated unit of recovery.

- `-RECOVER INDOUBT (imsid) ACTION (COMMIT) ID (pst#.psbname)`
- `-RECOVER INDOUBT (imsid) ACTION (ABORT) ID (pst#.psbname)`

Results

One of the following messages might be issued after you issue the RECOVER command:

```
DSNV414I - THREAD pst#.psbname COMMIT SCHEDULED
DSNV415I - THREAD pst#.psbname ABORT SCHEDULED
```

GUPI

Related tasks:

"Resolving indoubt units of recovery" on page 386

Displaying postponed IMS-DB2 units of recovery

You can display a list of postponed IMS-DB2 units of recovery.

About this task

GUPI

Procedure

To display a list of postponed units of recovery:

Issue the DISPLAY THREAD command. For example:

```
-DISPLAY THREAD (imsid) TYPE (POSTPONED)
```

In this command, *imsid* is the connection name. 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 '-DISPLAY THREAD' NORMAL COMPLETION
```

GUPI

Related tasks:

Chapter 10, "Restarting DB2 after termination," on page 363

Related reference:

-DISPLAY THREAD (DB2) (DB2 Commands)

Related information:

DSNV401I (DB2 Messages)

Resolving IMS residual recovery entries

You can resolve IMS residual recovery entries (RREs).

Procedure

To resolve IMS RREs:

1. To display the residual recovery entry (RRE) information, issue the following command:

   `/DISPLAY OASN SUBSYS subsystem-name`

2. 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 that is listed in the display. The originating application sequence number is the schedule number of the program instance, indicating 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.

Results

These commands reset the status of IMS; they do not result in any communication with DB2.

IMS residual recovery entries:

At certain times, IMS builds a list of *residual recovery entries* (RREs). RREs are units of recovery about which DB2 might be in doubt.

RREs occur 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.
- 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.

Related information:

"Recovering from IMS indoubt units of recovery" on page 476

Controlling IMS dependent region connections

Controlling IMS dependent region connections involves connecting from dependent regions, monitoring connection activity, and disconnecting from dependent regions.

How IMS dependent region connections work:

The IMS attachment facility that is 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 is never to 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 subsystem establishes the following connections to DB2:
   - A control region coordinator connection
   - An application connection from each dependent region

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 abends. This is a PSTOP transaction type; the input transaction is re-queued for processing, and new transactions are queued.

A The application abends. 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 by using the resource translation table (RTT).

Related concepts:

IMS attachment facility macro (DSNMAPN) (DB2 Installation and Migration)

Disconnecting from IMS dependent regions:

Usually, IMS master terminal operators avoid disconnecting a dependent region explicitly.

About this task

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**.

Monitoring activity on connections from DB2

A thread is established from a dependent region when an application makes its first successful DB2 request. You can issue IMS or DB2 commands to see information about connections and the applications that currently use them.

Procedure

To monitor activity on connections, issue the following commands:

- **From DB2:**

  -DISPLAY THREAD (*imsid*)
- **From IMS:**

  /SSR -DISPLAY THREAD (*imsid*)

Results

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
```

Related tasks:

"Displaying information by location" on page 280

Related reference:

-DISPLAY THREAD (DB2) (DB2 Commands)

Related information:

DSNV401I (DB2 Messages)

Monitoring activity of connections from IMS

You can monitor the connection to DB2 from IMS by using the **/DISPLAY SUBSYS** command.

About this task

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.

Procedure

To monitor the connection to DB2 from IMS:

Issue the **/DISPLAY SUBSYS** command with the following syntax:
`/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 shown as one of the following states:

- CONN
- CONN, ACTIVE (includes LTERM of user)

Example

The following four examples show the output that might be generated when you issue the IMS **/DISPLAY SUBSYS** command.

The following figure 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 23. Example of output from IMS /DISPLAY SUBSYS processing

The following figure 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 24. Example of output from IMS /DISPLAY SUBSYS processing

The following figure 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 25. Example of output from the IMS /DISPLAY SUBSYS command

The following figure 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 26. 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 IMS

The connection between IMS and DB2 ends when either IMS or DB2 terminates. Alternatively, the IMS master terminal operator can explicitly break the connection.

About this task

To break the connection, enter 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 re-establish the connection.

In an implicit or explicit disconnect, the following 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 is terminating normally (for example, /CHE FREEZE|DUMPQ|PURGE). Connected threads complete.
B	IMS is terminating abnormally. Connected threads are rolled back. DB2 data is backed out now; DL/I data is backed out at 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 terminating abnormally. Connected threads are rolled back. DL/I data is backed out now. DB2 data is backed out now if DB2 terminated normally; otherwise, it is backed out 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

You can start or restart a Resource Recovery Services attachment facility (RRSAF) connection at any time after Resource Recovery Services (RRS) is started.

About this task

GUPI

If RRS is not started, an IDENTIFY request fails with reason code X'00F30091'.

Application programs can use the following 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 that are to be associated with the connection. Invokes the sign-on exit routine. Optionally, lets a thread join a global transaction.

AUTH SIGNON
Provides a user ID, an access control environment element (ACEE), and, optionally, one or more secondary authorization IDs that are to be associated with the connection. Invokes the sign-on exit routine.

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, the TERMINATE IDENTIFY command 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.

GUPI

Related tasks:

Invoking the Resource Recovery Services attachment facility (DB2 Application programming and SQL)

Programming for concurrency (DB2 Performance)

Abnormal termination involving DB2 and RRS

If DB2 abnormally terminates but RRS remains active, RRS might commit or roll back work without DB2 knowledge. In a similar manner, if RRS abnormally terminates after DB2 has completed phase 1 of commit processing for an application, 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 RRS indoubt units of recovery

You can display a list of the Resource Recovery Services (RRS) indoubt units of recovery.

Procedure

GUPI

To display a list of indoubt units of recovery:

Issue the following DISPLAY THREAD command:

```
-DISPLAY THREAD (RRSAF) TYPE (INDOUBT)
```

The command produces output that is similar to the following example:

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV406I - INDOUBT THREADS -
COORDINATOR              STATUS      RESET URID           AUTHID
RRSAF                    INDOUBT           00019B8ADE9E   ADMF001
```

```
 V449-HAS NID= AD64101C7EED900000000000101010000 AND ID= ST47653RRS
DISPLAY INDOUBT REPORT COMPLETE
DSN9022I - DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
```

For RRSAF connections, a network ID is the z/OS RRS unit of recovery ID (URID), which uniquely identifies a unit of work. A z/OS RRS URID is a 32-character number.

GUPI

Related reference:

-DISPLAY THREAD (DB2) (DB2 Commands)

Related information:

DSNV401I (DB2 Messages)

Recovering RRS indoubt units of recovery manually

You might need to manually recover an indoubt unit of recovery if the RRS log is lost. When that happens, message DSN3011I is displayed on the z/OS console.

Procedure

GUPI To recover an indoubt unit of recovery:

1. Determine the correlation ID of the thread to be recovered by issuing the DISPLAY THREAD command.
2. Issue one of the following commands to recover the indoubt unit:
   - `-RECOVER INDOUBT (RRSAF) ACTION (COMMIT) ID (`*correlation-id*`)`
   - `-RECOVER INDOUBT (RRSAF) ACTION (ABORT) ID (`*correlation-id*`)`

   The ACTION parameter of the RECOVER command indicates whether to commit or roll back the associated unit of recovery.

Results

If you recover a thread that is part of a global transaction, all threads in the global transaction are recovered.

The following messages might be issued when you issue the RECOVER INDOUBT command:

`DSNV414I - THREAD `*correlation-id*` COMMIT SCHEDULED`

`DSNV415I - THREAD `*correlation-id*` ABORT SCHEDULED`

The following DSNV418I message might also be issued:

`DSNV418I - RECOVER INDOUBT REJECTED FOR ID=`*correlation-id*

If this message is issued, use the following NID option of RECOVER INDOUBT:

`-RECOVER INDOUBT(RRSAF) ACTION(`*action*`) NID(`*nid*`)`

where *nid* is the 32-character field that is displayed in the DSNV449I message.

GUPI

Related concepts:

“Multiple system consistency” on page 379

Related tasks:

“Resolving indoubt units of recovery” on page 386

Related reference:

-DISPLAY THREAD (DB2) (DB2 Commands)

-RECOVER INDOUBT (DB2) (DB2 Commands)

Displaying RRS postponed units of recovery

You can display a list of the Resource Recovery Services (RRS) postponed units of recovery.

Procedure

GUPI

To display a list of postponed units of recovery:

Issue the following DISPLAY THREAD command:

```
-DISPLAY THREAD (RRSAF) TYPE (POSTPONED)
```

The command produces output that is similar to the following example:

```
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 '-DISPLAY THREAD' NORMAL COMPLETION
```

For RRSAF connections, a network ID is the z/OS RRS unit of recovery ID (URID), which uniquely identifies a unit of work. A z/OS RRS URID is a 32-character number.

GUPI

Related reference:

-DISPLAY THREAD (DB2) (DB2 Commands)

Related information:

DSNV401I (DB2 Messages)

Monitoring and displaying RRSAF connections

The Resource Recovery Services attachment facility (RRSAF) allows an application or application monitor to disassociate a DB2 thread from a TCB. Later the thread can be associated with the same or a different TCB within the same address space.

About this task

GUPI

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 disassociate the CONTEXT from the current TCB and then associate the CONTEXT with the same TCB or a different TCB.

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 the output in the following figure:

```
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	Description
1	This is an application that used CREATE THREAD to allocate the special plan that is 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 27. DISPLAY THREAD output showing RRSAF connections

GUPI

Disconnecting RRSAF applications from DB2

To disconnect a Resource Recovery Services attachment facility (RRSAF) transaction from DB2, you must abnormally terminate the transaction.

Procedure

GUPI

To cancel a particular thread:

Issue the CANCEL THREAD command.
The CANCEL THREAD command requires that you know the *token* for any thread that you want to cancel. Issue the DISPLAY THREAD command to obtain the token number, and then enter the following command to cancel the thread:

```
-CANCEL THREAD(token)
```

When you issue the CANCEL THREAD command, DB2 schedules the thread for termination.

GUPI

Controlling connections to remote systems

You can control connections to remote systems that use distributed data by controlling the threads. The types of threads that are involved in connecting to other systems are allied threads and database access threads.

About this task

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 that is initiated by a remote DBMS to your DB2 subsystem.

Related concepts:

"DB2 commands for monitoring connections to other systems" on page 315

Related tasks:

"Resolving indoubt units of recovery" on page 386

Related information:

"Recovering from database access thread failure" on page 550

Starting DDF

You can start the distributed data facility (DDF) if you have at least SYSOPR authority.

About this task

When you install DB2, you can request that the DDF start automatically when DB2 starts.

Procedure

GUPI

To start the DDF, if it has not already been started:

Issue the START DDF command.
When DDF is started and is responsible for indoubt thread resolution with remote partners, message DSNL432I, message DSNL433I, or both, is generated. These messages summarize the responsibility DDF has for indoubt thread resolution with remote partners.
The following messages are associated with the START DDF command:

```
DSNL003I - DDF IS STARTING
DSNL004I - DDF START COMPLETE LOCATION locname
           LU netname.luname
           GENERICLU netname.gluname
           DOMAIN domain
           TCPPORT tcpport
           SECPORT secport
           RESPORT resport
```

If the DDF is not properly installed, the START DDF command fails, and message `,DSN9032I, - REQUESTED FUNCTION IS  NOT AVAILABLE,` is issued. If the DDF has 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.

GUPI

Related concepts:

Chapter 11, "Maintaining consistency across multiple systems," on page 379

Related reference:

-START DDF (DB2) (DB2 Commands)

-DISPLAY DDF (DB2) (DB2 Commands)

DSNTIPR: Distributed data facility panel 1 (DB2 Installation and Migration)

Suspending DDF server activity

You can use the STOP DDF MODE(SUSPEND) command to suspend distributed data facility (DDF) server threads temporarily.

About this task

GUPI

Suspending DDF server threads frees all resources that are held by the server threads and lets the following operations complete:

- CREATE
- ALTER
- DROP
- GRANT
- REVOKE

When you issue the STOP DDF MODE(SUSPEND) command, 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.

GUPI

Related reference:

-STOP DDF (DB2) (DB2 Commands)

Resuming DDF server activity

You can resume suspended distributed data facility (DDF) server activity.

Procedure

GUPI

To resume suspended DDF server threads:

Issue the START DDF command.

GUPI

Related reference:

-START DDF (DB2) (DB2 Commands)

Displaying information about DDF work

The DISPLAY DDF command displays information regarding the status of the distributed data facility (DDF). This command also displays information that is related to the start of DDF, such as the location name, the LU name, the IP address, and domain names.

Before you begin

GUPI

To issue the DISPLAY DDF command, you must have SYSOPR authority or higher.

About this task

Tip: You can use the optional DETAIL keyword to receive additional configuration and statistical information.

The DISPLAY DDF DETAIL command is especially useful, because the command displays new inbound connections that are not indicated by other commands. For example, sometimes new inbound connections are not yet reflected in the DISPLAY THREAD report. Cases of when a new inbound connection is not displayed include if DDF is in INACTIVE MODE, as denoted by a DT value of **I** in the message DSNL090I, and DDF is stopped with mode SUSPEND, or the maximum number of active database access threads has been reached. These new connections are displayed in the DISPLAY DDF DETAIL report. However, 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 associated with a database access thread.

Procedure

To display information about DDF work:

Enter one of the following commands:

- To show only the basic information, enter the DISPLAY DDF command.

  ```
  -DISPLAY DDF
  ```

  DB2 returns output similar to the following sample.

  ```
  DSNL080I - DSNLTDDF DISPLAY DDF REPORT FOLLOWS-
  DSNL081I STATUS=STARTD
  DSNL082I LOCATION           LUNAME            GENERICLU
  DSNL083I SVL650A               USIBMSY.SYEC650A     -NONE
  DSNL084I TCPPORT=446   SECPORT=0 RESPORT=5001 IPNAME=-NONE
  DSNL085I IPADDR=::9.110.115.106
  DSNL085I IPADDR=2002:91E:610:1::5
  DSNL086I SQL    DOMAIN=v8ec103.svl.ibm.com
  DSNL086I RESYNC DOMAIN=v8ec103.svl.ibm.com
  DSNL087I ALIAS              PORT  SECPORT  STATUS
  DSNL088I ALIASLOC1          551   0        STATIC
  DSNL088I ALIASLOC2          552   0        STATIC
  DSNL088I ALIASLOC3          553   0        STATIC
  DSNL088I ALIASLOC4          554   0        STATIC
  DSNL088I ALIASLOC5          555   0        STATIC
  DSNL088I ALIASLOC6          556   0        STATIC
  DSNL088I ALIASLOC7          557   0        STATIC
  DSNL088I ALIASLOC8          558   0        STATIC
  DSNL089I MEMBER IPADDR=::9.110.115.112
  DSNL089I MEMBER IPADDR=2002:91E:610:1::112
  ```

```
DSNL105I CURRENT DDF OPTIONS ARE:
DSNL106I PKGREL = COMMIT
DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE
```

- To show additional information, enter the DISPLAY DDF command with the DETAIL option.

  ```
  -DISPLAY DDF DETAIL
  ```

  With the DETAIL option, the following additional information is included in the output:

  ```
  DSNL090I DT=      A CONDBAT=  64 MDBAT=   64
  DSNL092I ADBAT=   1 QUEDBAT=   0 INADBAT=  0 CONQUED=      0
  DSNL093I DSCDBAT= 0 INACONN=   0
  DSNL100I     LOCATION SERVER LIST:
  DSNL101I     WT IPADDR            IPADDR
  DSNL102I     64 ::9.110.115.111   2002:91E:610:1::111
  DSNL102I        ::9.110.115.112   2002:91E:610:1::112
  DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE
  ```

GUPI

Related reference:

-DISPLAY DDF (DB2) (DB2 Commands)

Related information:

DSNL080I (DB2 Messages)

DB2 commands for monitoring connections to other systems:

By issuing certain DB2 commands, you can generate information about the status of distributed threads.

GUPI

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 LOCATION
Displays statistics about threads and conversations between a remote DB2 subsystem and the local subsystem.

DISPLAY THREAD
Displays information about DB2, distributed subsystem connections, and parallel tasks.

GUPI

Displaying information about connections with other locations:

The DISPLAY LOCATION command displays summary information about connections with other locations. You can also use this command to display detailed information about the conversations with other locations.

Before you begin

Prerequisite: To issue the DISPLAY LOCATION command, you must have SYSOPR authority or higher.

About this task

You can specify location names, SNA LU names, or IP addresses, and the DETAIL keyword is supported.

GUPI To display information about connections with other locations:

Procedure

Issue the DISPLAY LOCATION command. For example:

```
-DISPLAY LOCATION(*)
```

DB2 returns output that is similar to the following example:

```
DSNL200I  @ DISPLAY LOCATION REPORT FOLLOWS-
LOCATION                                        PRDID    T ATT CONNS
LUND0                                           DSN10011 R          1
LUND1                                           DSN10011 S          0
::FFFF:124.63.51.17..50000                      SQL09073 R          3
::FFFF:124.63.51.17                             SQL09073 S         15
DISPLAY LOCATION REPORT COMPLETE
```

You can use an asterisk (*) in place of the end characters of a location name. For example, you can use `DISPLAY LOCATION(SAN*)` to display information about all active connections between your DB2 subsystem and a remote location that begins with "SAN". The results include the number of conversations and conversation role, requester, or server.
When DB2 connects with a remote location, information about that location, persists in the report even if no active connections exist, including:

- The physical address of the remote location (LOCATION)
- The product identifier (PRDID)

Example

The DISPLAY LOCATION command displays the following types of information for each DBMS that has active connections, except for the local subsystem:

- The network address for the location:
  - For remote locations accessed through SNA connections, the location name the SNA LU name.
  - For remote locations accessed through TCP/IP connections, the location name is the dotted decimal IPv4 or colon hexadecimal IPv6 address. If the T column contains R, the IP address is concatenated with the port value of the remote location.
- The PRDID, which identifies the database product at the location.

  The format of product identifier values is *pppvvrrm*, where *ppp* is a 3-letter product code (such as DSN for DB2), *vv* is the version, *rr* is the release, and *m* is the modification level. For example, DSN10015 identifies DB2 10 in new-function mode, the value is 'DSN10015'. The product code (*ppp*) is one of the following values:

  AQT for IBM DB2 Analytics Accelerator for z/OS

ARI for DB2 Server for VSE & VM

DSN for DB2 for z/OS

JCC for IBM Data Server Driver for JDBC and SQLJ

QSQ for DB2 for i

SQL for DB2 for Linux, UNIX, and Windows

Modification (*m*) values have the following meanings:

- Values '0' and '1' identify modification levels in conversion and enabling-new-function mode from Version 8 (CM8, CM8*, ENFM8, and ENFM8*)
- Values '2' and '3' identify modification levels in conversion and enabling-new-function mode from Version 9 (CM9, CM9*, ENFM9, and ENFM9*)
- Value 4 is reserved.
- Values '5', '6', '7', '8', and '9' identify modification levels in new-function mode.

- Whether the local system is requesting data from the remote system, or acting as a server to the remote system. 'R' indicates a requester connection from local subsystem accessing the remote system. 'S' indicates a server connection from remote system accessing the local subsystem.
- The number of connections that have a particular attribute from or to the location. The attribute value is blank for the message line that contains the total number of connections for the location. Additional lines for connections with particular attributes are shown only when a detailed report is requested.
- The total number of conversations that are in use between the local system and the remote system

For example, suppose two threads are at location USIBMSTODB21. One thread is a database access thread that is related to a non-DB2 requester system, and the other is an allied thread that goes from USIBMSTODB21 to the non-DB2 server system. Both threads use SNA connections. The DISPLAY LOCATION command that is issued at USIBMSTODB21 displays the following output:

```
DSNL200I  @ DISPLAY LOCATION REPORT FOLLOWS-
LOCATION                                        PRDID    T ATT CONNS
LUND1                                                    R          1
LULA                                            DSN09010 S          1
DISPLAY LOCATION REPORT COMPLETE
```

The following output shows the result of the `DISPLAY LOCATION(*)` command when DB2 is connected to the following DRDA partners:

- TCP/IP for DRDA connections to ::FFFF:124.38.54.16
- SNA connections to LULA.

```
DSNL200I  @ DISPLAY LOCATION REPORT FOLLOWS-
LOCATION                                        PRDID    T ATT CONNS
::FFFF:124.38.54.16..446                        DSN10015 R          2
::FFFF:124.38.54.16                             DSN10015 S          1
LULA                                            DSN09010 R          1
LULA                                            DSN09010 S          2
LULA
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 exists:

- Sessions currently exist with the partner location, but no active conversations are allocated to any of the sessions.
- Sessions no longer exist with the partner, because contact with the partner has been lost.

When you specify the DETAIL option in the DISPLAY LOCATION command, the report might include additional messages lines for each remote location, including:

- Information about the number of conversations in the CONNS column with each remote location that have particular attributes. that are identified by a value in the ATT column. For example, the number of connections that use trusted context are identified by TRS.
- Information about conversations that are owned by DB2 system threads, such as those that are used for resynchronization of indoubt units of work.

GUPI

Canceling SQL from an IBM data server driver

You can use the CLI and ODBC function SQLCancel() or the JDBC cancel method to cancel a remote SQL request from a client application.

Procedure

GUPI To cancel SQL statements that are running on a remote DB2 for z/OS server:

1. Establish an additional remote connection from your application to the remote DB2 server.
2. From that connection, issue SQLCancel() or invoke the JDBC cancel method.

Results

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 codes that excludes transaction-level statements (CONNECT, COMMIT, ROLLBACK) and bind statements from a client application.

GUPI

Related reference:

SQLCancel function (CLI) - Cancel statement

Configuration of Sysplex workload balancing for non-Java clients

InterruptProcessingMode IBM Data Server Driver configuration keyword

Driver support for JDBC APIs

Common IBM Data Server Driver for JDBC and SQLJ properties

Configuration of Sysplex workload balancing for Java clients

Canceling threads

You can use the CANCEL THREAD command to terminate threads that are active or suspended in DB2.

Before you begin

Using the CANCEL THREAD command requires SYSOPR authority or higher.

About this task

The command has no effect if the thread is not active or suspended in DB2.

GUPI If the thread is processing in VTAM or TCP/IP, you can use VTAM or TCP/IP commands to terminate the conversations.

You can use the DISPLAY THREAD command to determine if a thread is hung in DB2 or VTAM. In VTAM, there is no reason to use the CANCEL command.

Procedure

To terminate a thread, enter one of the following commands:

- To cancel a thread with a token, 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)`

Results

The *token* is a 1-character to 5-character number that identifies the thread result. When DB2 schedules the thread for termination, the following message for a distributed thread is issued:

```
DSNL010I - DDF THREAD token or luwid HAS BEEN CANCELED
```

For a non-distributed thread, you see the following message:

```
DSNV426I - csect THREAD token HAS BEEN CANCELED
```

As a result of entering CANCEL THREAD, the following messages can be displayed:

DSNL009I
DSNL010I
DSNL022I

CANCEL THREAD allows you to specify that a diagnostic dump be taken. GUPI

Related concepts:

Obtaining DB2 Diagnosis Guide and Reference ()

Related reference:

-CANCEL THREAD (DB2) (DB2 Commands)

Effects of the CANCEL THREAD command:

A database access thread can be in the prepared state, waiting for the commit decision from the coordinator. When you issue the CANCEL THREAD command for a database access thread that is in the prepared state, the thread is converted from active to indoubt status.

GUPI

The conversation with the coordinator and all conversations with downstream participants are terminated, and message DSNL450I is returned. The resources that are 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.

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.

GUPI

Related tasks:

"Resolving indoubt units of recovery" on page 386

Monitoring and controlling stored procedures

Stored procedures, such as native SQL procedures, external SQL procedures, and external stored procedures, are user-written programs that run at a DB2 server.

About this task

External SQL procedures and external stored procedures run in 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 a z/OS system that is part of a Sysplex, the scope of that command is the Sysplex.

Related tasks:

Creating a stored procedure (DB2 Application programming and SQL)

Displaying information about stored procedures with DB2 commands:

Use the DISPLAY PROCEDURE command and the DISPLAY THREAD command to obtain information about a stored procedure while it is running.

About this task

Because native SQL procedures do not run in WLM-established address spaces, the best way to monitor native SQL procedures is by using the START TRACE command and specifying accounting class 10, which activates IFCID 239.

Related concepts:

DB2 trace output (DB2 Performance)

Related reference:

-START TRACE (DB2) (DB2 Commands)

-DISPLAY THREAD (DB2) (DB2 Commands)

-DISPLAY PROCEDURE (DB2) (DB2 Commands)

Displaying statistics about stored procedures:

Issue the DISPLAY PROCEDURE command to display statistics about stored procedures that are accessed by DB2 applications.

About this task

This command can display the following information about stored procedures:

- Status (started, stop-queue, stop-reject, or stop-abend)
- Number of requests that are currently running and queued
- Maximum number of threads that are running a stored procedure load module and queued
- Count of timed-out SQL CALLs

Procedure

To display information about all stored procedures in all schemas that have been accessed by DB2 applications:

Issue the DISPLAY PROCEDURE command.

GUPI

For example:

```
-DISPLAY PROCEDURE
```

GUPI

Note: To display information about a native SQL procedure, you must run the procedure in DEBUG mode. If you do not run the native SQL procedure in DEBUG mode (for example, in a production environment), the DISPLAY PROCEDURE command will not return output for the procedure.

If you do run the procedure in DEBUG mode the WLM environment column in the output contains the WLM ENVIRONMENT FOR DEBUG that you specified when you created the native SQL procedure. The DISPLAY PROCEDURE output shows the statistics of native SQL procedures as '0' if the native SQL procedures are under the effect of a STOP PROCEDURE command.

Example

GUPI

The following example shows two schemas (PAYROLL and HRPROD) that have been accessed by DB2 applications. You can also display information about specific stored procedures.

```
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
```

GUPI

Related reference:

-DISPLAY PROCEDURE (DB2) (DB2 Commands)

Displaying thread information about stored procedures:

Issue the DISPLAY THREAD command to display thread information about stored procedures.

About this task

This command tells whether:
- A thread is waiting for a stored procedure to be scheduled.
- A thread is executing within a stored procedure.

Procedure

GUPI To display active threads that are running stored procedures and user-defined functions:

Issue the DISPLAY THREAD command. For example:

```
-DISPLAY THREAD(*) TYPE(PROC)
```

GUPI

Example

Example 1: The following example of output from the DISPLAY THREAD command shows a thread that is executing an external SQL procedure or an external stored procedure.

```
-display thread(*) type(proc) detail
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 '-DISPLAY THREAD' 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.

Example 2: This example of output from the DISPLAY THREAD command shows a thread that is executing a native SQL procedure. If you do not specify the DETAIL option, the output will not include information that is specific to the stored procedure.

Issuing the command `-display thread(*) type(proc) detail` results in the following output:

```
SERVER RA * 22325 driver.exe USRT010 DISTSERV 0067 8
V437-WORKSTATION=CALADAN, SERID=USRT010,
APPLICATION NAME=driver.exe
V436-PGM=USRT010.MARKETWATCH_F1, SEC=3, STMNT=39
V442-CRTKN=9.30.129.213.15369.090129190416
V445-G91E81D8.C3B9.C3AB43861A26=8 ACCESSING DATA FOR
( 1)::FFFF:9.30.129.213
V447--INDEX SESSID A ST TIME V448--( 1) 50105:2364 W S2 902911191075
```

Example 3: The following example of output from the DISPLAY THREAD command shows a thread that is executing a user-defined function.

```
-display thread(*) type(proc) detail
DSNV401I ! DISPLAY THREAD REPORT FOLLOWS -
DSNV402I ! ACTIVE THREADS -
NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
BATCH    SP      27 LI33FN1      SYSADM   MYPLAN   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
```

Related reference:

-DISPLAY THREAD (DB2) (DB2 Commands)

Determining the status of an application environment:

Use the z/OS command DISPLAY WLM to determine the status of an application environment in which a stored procedure runs.

About this task

The output from the DISPLAY WLM command 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
```

Results

You might get results like the following:

```
IWM029I  15.22.22  WLM DISPLAY
APPLICATION ENVIRONMENT NAME     STATE     STATE DATA
WLMENV1                          AVAILABLE
ATTRIBUTES: PROC=DSNWLM1 SUBSYSTEM TYPE: DB2
```

The output indicates that WLMENV1 is available, so WLM can schedule stored procedures for execution in that environment.

Refreshing WLM application environments for stored procedures:

When you make certain changes to a stored procedure or to the JCL startup procedure for a WLM application environment, you need to refresh the WLM application environment.

Before you begin

Before you refresh a WLM application environment, ensure that WLM is operating in goal mode.

About this task

Refreshing the WLM environment starts a new instance of each address space that is active for the WLM environment. Existing address spaces stop when the current requests that are executing in those address spaces complete.

Refresh the WLM application environment if any of the following situations are true:

- For external procedures (including external SQL procedures), you changed the stored procedure logic, the load module, or the CREATE PROCEDURE definition.
- For a Java™ stored procedure, you changed the properties that are pointed to by the JAVAENV data set.
- You changed the JCL startup procedure for the WLM application environment.

  Restriction: In some cases, refreshing the WLM environment might not be sufficient to incorporate your change. For example, assume that you changed the NUMTCB value in the JCL. The refresh fails because WLM cannot start a new WLM address space that has a different NUMTCB value as the existing one. In this case, you need to quiesce the WLM environment while you change the JCL startup procedure, and resume the environment when your changes are complete.

If you create a procedure that uses an existing WLM environment, you do not need to refresh the WLM environment.

Procedure

To refresh a WLM application environment for stored procedures:

Perform one of the following actions:

- Call the WLM_REFRESH stored procedure.
- Issue the following z/OS command:

  ```
  VARY WLM,APPLENV=environment-name,REFRESH
  ```

  In this command, *environment-name* is the name of a WLM application environment that is associated with one or more stored procedures. The application environment is refreshed to incorporate the changed load modules for all stored procedures and user-defined functions in the particular environment.

Alternatively, when you make certain changes to the JCL startup procedure, you must quiesce and then resume the WLM application environment rather than refresh it. For these types of changes, use the following z/OS commands:

1. To stop all stored procedures address spaces that are associated with the WLM application environment *name*, use the following z/OS command:

   ```
   VARY WLM,APPLENV=name,QUIESCE
   ```

   The address spaces stop when the current requests that are executing in those address spaces complete.

   This command puts the WLM application environment in QUIESCED state. When the WLM application environment is in QUIESCED state, the stored procedure requests are queued. If the WLM application environment is restarted within a certain time, the stored procedures are executed. If a stored procedure cannot be executed, the CALL statement returns SQL code -471 with reason code 00E79002.
2. To restart all stored procedures address spaces that are associated with WLM application environment *name*, use the following z/OS command:

   ```
   VARY WLM,APPLENV=name,RESUME
   ```

   New address spaces start when all JCL changes are established. Until that time, work requests that use the new address spaces are queued.

   Also, you can use the VARY WLM command with the RESUME option when the WLM application environment is in the STOPPED state due to a failure. This state might be the result of a failure when starting the address space, or because WLM detected five abnormal terminations within 10 minutes. When an application environment is in the STOPPED state, WLM does not schedule stored procedures for execution in it. If you try to call a stored procedure when the WLM application environment is in the STOPPED state, the CALL statement returns SQL code -471 with reason code 00E7900C. After correcting the condition that caused the failure, you need to restart the application environment.

Related concepts:

WLM management of stored procedures (DB2 Installation and Migration)

Related tasks:

Setting up a WLM application environment for stored procedures during installation (DB2 Installation and Migration)

Related reference:

WLM_REFRESH stored procedure (DB2 SQL)

CREATE PROCEDURE (SQL - native) (DB2 SQL)

Related information:

Using Operator Commands for Application Environments (z/OS MVS Planning: Workload Management)

Refreshing WLM environments for stored procedures automatically:

You can enable automatic refreshes for WLM environments for stored procedures.

Before you begin

If a security product is configured to authorize MVS console commands, you must grant authorization for the DB2 ssnmMSTR address space to issue MVS commands.

Procedure

To enable automatic refreshes of WLM environments for stored procedures:

Add the following DD statement to the startup procedure for the WLM-established stored procedure address space:

```
//AUTOREFR DD DUMMY
```

DB2 saves the environment name in a list of environments to refresh automatically. If the z/OS Resource Recovery Services (RSS) environment is recycled, but DB2 and the its associated WLM-established stored procedures address space are not restarted, DB2 issues the following z/OS command to refresh each environment that is named in the list:

```
VARY WLM,APPLENV=environment-name,REFRESH
```

Recommendation: To prevent the automatic issuing of duplicate and unnecessary `VARY WLM,APPLENV=applenvname` commands, limit the number of startup procedures that contain the //AUTOREFR DD statement . The `WLM,APPLENV=applenvname` command has a Sysplex scope, which means that it affects all servers of an application environment on all systems in the Sysplex. The result is that every DB2 subsystem issues its own `WLM,APPLENV=applenvname` command for each application environment that is nominated by the //AUTOREFR DD statement in its startup procedure.

Related concepts:

WLM management of stored procedures (DB2 Installation and Migration)

WLM address space startup procedure for Java routines (DB2 Application Programming for Java)

Related tasks:

Assigning stored procedures and functions to WLM application environments (DB2 Performance)

"Refreshing WLM application environments for stored procedures" on page 324

Defining application environments (MVS Planning: Workload Management)

Obtaining diagnostic information and debugging stored procedures:

You have several options for obtaining diagnostic information and debugging stored procedures, depending on the type of stored procedure.

Procedure

To obtain diagnostic information and debug stored procedures:

Take the appropriate action, depending on the type of stored procedure that you use.

Type of stored procedure	Actions
All types of stored procedures	• Look at the diagnostic information in CEEDUMP. 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 is printed after 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. • Debug the stored procedure as a stand-alone program on a workstation. • Record stored procedure debugging messages to a disk file or JES spool file by using the Language Environment MSGFILE run time option. • Store debugging information in a table. This option works well for remote stored procedures.
C stored procedures	Use the Debug Tool for z/OS.
C++ stored procedures	Use the Debug Tool for z/OS.
COBOL stored procedures	Use the Debug Tool for z/OS.
External applications	Use a driver application.
External SQL procedures	• Use the Unified Debugger. • Use the DB2 stored procedure debugger, which is part of IBM Data Studio.
Java stored procedures	• Use the Unified Debugger. • Use the DB2 stored procedure debugger, which is part of IBM Data Studio.
Native SQL procedures	Use the GET DIAGNOSTICS statement. The **DB2_LINE_NUMBER** parameter returns: • The line number where an error is encountered in parsing, binding, or executing a CREATE or ALTER statement for a native SQL procedure. • The line number when a CALL statement invokes a native SQL procedure and the procedure returns with an error. This information is not returned for an external SQL procedure, and this value is meaningful only if the statement source contains new line control characters.

What to do next

Remember: After you finish debugging stored procedures, remember to disable the debugging option that you used, so that you do not run debugging tools in a production system.

Related tasks:

"Refreshing WLM application environments for stored procedures" on page 324

Related reference:

IBM Data Studio product documentation (IBM Data Studio, IBM Optim Database Administrator, IBM infoSphere Data Architect)

Migrating stored procedures from test to production:

After developing and testing a stored procedure, you can migrate it from a test environment to production.

About this task

The process that you follow to migrate a stored procedure from a test environment to production depends on the type of stored procedure that you want to migrate. The process that you follow also depends on the change management policy of your site. You can migrate a native SQL procedure, an external SQL procedure, or an external stored procedure. You also can choose to recompile to create new object code and a new package on the production server, or you can choose not to recompile.

Related tasks:

"Implementing DB2 stored procedures" on page 84

Migrating native SQL procedures from test to production:

Migrating native SQL procedures from a test environment to production is a straightforward process.

About this task

Because native SQL procedures do not contain load modules that need to be link-edited or WLM address spaces that need to be refreshed, you do not need to determine whether you want to recompile to create new object code and a new package on the production server.

Procedure

To migrate a native SQL procedure from a test environment to production:

Deploy the native SQL procedure to the new server.

Migrating external SQL procedures from test to production:

Use IBM Data Studio to migrate external SQL procedures from a test environment to production.

About this task

External SQL procedures are usually built by using IBM Data Studio.

Procedure

GUPI

To migrate an external SQL procedure from a test environment to production:

Use IBM Data Studio to deploy the stored procedure. The binary deploy capability in IBM Data Studio promotes external SQL procedures without recompiling. The binary deploy capability copies all necessary components from one environment to another and performs the bind in the target environment.

GUPI

Migrating external stored procedures from test to production:

Use the CREATE PROCEDURE statement to migrate external stored procedures from a test environment to a production environment.

About this task

For Java stored procedures, you also can use IBM Data Studio to do a binary deploy of the stored procedure. The binary deploy capability promotes Java stored procedures without recompiling. The binary deploy capability copies all necessary components from one environment to another and performs the bind in the target environment.

Procedure

GUPI To migrate an external stored procedure from a test environment to production:

1. Determine the change management policy of your site. You can choose to recompile to create new object code and a new package on the production server, or you can choose not to recompile.
2. Depending on your change management policy, complete the appropriate task.
   - To migrate the stored procedure without recompiling:
     a. Copy the CREATE PROCEDURE statement.
     b. Modify the CREATE PROCEDURE statement to reflect the new schema, new collection ID and new WLM application environment.
     c. Define the stored procedure with the new CREATE PROCEDURE statement. You can use the IBM-supplied programs DSNTIAD or DSNTEP2.

        Note: Make sure that the schema, collection ID and WLM application environment correspond to the new environment or code level.
     d. Copy the DBRM and bind the DBRM to produce a DB2 package.

        Note: Make sure that the collection ID of the BIND statement and collection ID of the CREATE PROCEDURE statement are the same.
     e. Copy the load module and refresh the WLM application environment.
   - To migrate the stored procedure and recompile to create new object code and a new package on the production server:
     a. Copy the CREATE PROCEDURE statement.
     b. Modify the CREATE PROCEDURE statement to reflect the new schema, new collection ID and new WLM application environment.

c. Define the stored procedure with the new CREATE PROCEDURE statement. You can use the IBM-supplied programs DSNTIAD or DSNTEP2.

   Note: Make sure that the schema, collection ID and WLM application environment correspond to the new environment or code level.

d. Copy the source code.

e. Precompile, compile, and link-edit. This step produces a DBRM and a load module.

f. Bind the DBRM to produce a DB2 package.

   Note: Make sure that the collection ID of the BIND statement and collection ID of the CREATE PROCEDURE statement are the same.

g. Refresh the WLM application environment.

GUPI

Monitoring DDF problems by using NetView

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.

About this task

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 the following figure.

```
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       RQST  ---  SRVR
            +--------+   +--------+
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 28. Recommended action for selected event panel in NetView. In this example, the AR (USIBMSTODB21) reports the problem, which affects the AS (USIBMSTODB22).

Key	Description
1	The system that is reporting the error. The system that is reporting the error is always on the left side of the panel. That system 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 that is affected by the error. The system that is affected by the error is always displayed to the right of the system that is reporting the error. The affected system 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, this system does not appear on the panel.

3 DB2 reason code.

Related concepts:

Obtaining DB2 Diagnosis Guide and Reference ()

Related reference:

IBM Tivoli NetView for z/OS User's Guide

DDF alerts:

Several 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 Hardware Monitor panels and are logged in the hardware monitor database. The following figure 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 29. 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 are RS (server) and RQ (requester).

Stopping DDF

You can stop the distributed data facility (DDF) if you have SYSOPR authority or higher.

Procedure

GUPI To stop the DDF:

Use one of the following commands:

```
-STOP DDF MODE (QUIESCE)
-STOP DDF MODE (FORCE)
```

Results

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. GUPI

Stopping DDF using the QUIESCE option:

Stop DDF by using the QUIESCE option of the STOP DDF command whenever possible. This option is the default.

About this task

GUPI With the QUIESCE option, 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 any indoubt units of work 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 the FORCE option.

Procedure

To stop DDF with the QUIESCE option:

Issue the following command:

```
-STOP DDF MODE (QUIESCE)
```

GUPI

Stopping DDF using the FORCE option:

Stop DDF by using the FORCE option of the STOP DDF command only when you must stop DDF quickly.

About this task

GUPI When DDF is stopped with the FORCE option, and DDF has indoubt thread responsibilities with remote partners, message DSNL432I, DSNL433I, or both are issued.

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 that are 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.

Procedure

To stop DDF with the FORCE option:

Issue the following command:

```
-STOP DDF MODE (FORCE)
```

GUPI

Stopping DDF using VTAM commands:

One 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.

Procedure

To stop force DDF to stop:

Enter the following command:

```
VARY NET,INACT,ID=db2lu,FORCE
```

where *db2lu* is the VTAM LU name for the local DB2 system.
When DDF has stopped, you must issue the following command before you can issue the START DDF command:

```
VARY NET,ACT,ID=db2lu
```

Controlling traces

Several traces are available for problem determination.

About this task

- 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

Related concepts:

Types of DB2 traces (DB2 Performance)

Diagnostic traces for attachment facilities

Several trace facilities provide diagnostic information.

- IMS provides a trace facility that shows the flow of requests across the connections from the IMS control and IMS 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.

  In addition, the IMS attachment facility of DB2 provides an internal wraparound 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 provides a series of menus that you can use to set CICS trace options to trace the CICS attachment facility. For CICS 4.1 and later, 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.
- 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 is included in any ABEND dump that is produced by RRSAF. This tracing facility provides a history of RRSAF usage that can aid in diagnosing errors in RRSAF.

Related concepts:

CICS Transaction Server for z/OS Supplied Transactions

Controlling the DB2 trace

DB2 provides commands for controlling the collection of trace data.

Before you begin

GUPI

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)

Procedure

To issue trace commands:

Select the appropriate command for the action that you want to take. 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 as a result of the parameters that were specified during installation or migration.

MODIFY TRACE
Changes the trace events (IFCIDs) that are being traced for a specified active trace.

You can specify several parameters to further qualify the scope of a trace. You can trace specific events within a trace type as well as events within specific DB2 plans, authorization IDs, resource manager IDs, and locations. You can also control where trace data is sent.
When you install DB2, you can request that any trace type and class start automatically when DB2 starts.

GUPI

Related concepts:

Types of DB2 traces (DB2 Performance)

Related tasks:

Minimizing the volume of DB2 trace data (DB2 Performance)

Related reference:

DSNTIPN: Tracing parameters panel (DB2 Installation and Migration)

-START TRACE (DB2) (DB2 Commands)

-DISPLAY TRACE (DB2) (DB2 Commands)

-STOP TRACE (DB2) (DB2 Commands)

-MODIFY TRACE (DB2) (DB2 Commands)

Diagnostic trace for the IRLM

You can control diagnostic traces for the IRLM using z/OS commands.

MODIFY *irlmproc*,**SET,TRACE**
Dynamically sets the maximum number of trace buffers for each trace type. IRLM uses this value 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 that are used for each trace type. Also displays indication of whether 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 acts independently of 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.

Related reference:

z/OS IRLM commands (DB2 Commands)

Setting the priority of stored procedures

Stored procedure priority is inherited from the caller. The stored procedure always runs at the dispatching priority of whatever called it.

About this task

For example, if you call a stored procedure from CICS, it runs at CICS priority. If you call a stored procedure from batch, it runs at batch priority. If you call a stored procedure from DDF, it runs at DDF priority.

Procedure

To set stored procedure priority:

1. When you set up WLM, ensure that the WLM address spaces are set up with the default started task priority. WLM address spaces that use the default started task priority perform system administrative work more efficiently.
2. Set up your service classes for the regular DB2 threads according to the priority that you want to give to the stored procedure callers. The stored procedure has the same priority as its caller.

Related tasks:

Setting up a WLM application environment for stored procedures during installation (DB2 Installation and Migration)

Setting performance objectives for distributed workloads by using z/OS Workload Manager (DB2 Performance)

Chapter 9. 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) contains information about the data sets that contain the log. You can perform a variety of tasks to ensure that DB2 logging satisfies the needs of your environment.

About this task

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*.

Related information:

Chapter 14, "Reading log records," on page 599

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 and points of consistency

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*. The unit of recovery ends at a later point of consistency.

A unit of recovery is the work that changes DB2 data from one point of consistency to another. This work is done by a single DB2 DBMS for an application. The point of consistency (also referred to as *sync point* or *commit point*) is a time when all recoverable data that an application program accesses is consistent with other data.

The following figure shows an example of units of recovery within an application program.

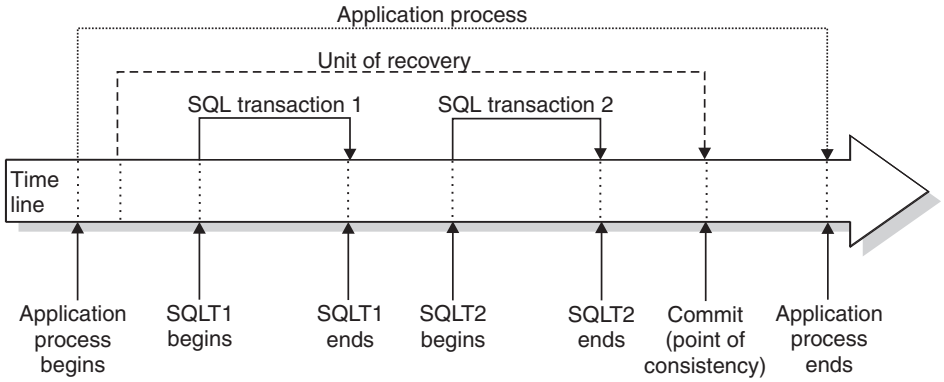


Figure 30. A unit of recovery within an application process

© Copyright IBM Corp. 1982, 2016

In this example, the application process makes changes to databases at SQL transactions 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.

Related concepts:

"Multiple system consistency" on page 379

How DB2 rolls back work

If failure occurs within a unit of recovery, DB2 rolls back (backs out) any changes to data. *Rolling back* returns the data to its state at the start of the unit of recovery; that is, DB2 undoes the work.

For a partition-by-growth table space, if a new partition was added in the unit of recovery, any uncommitted updates can be backed out, but the physical partition is not deleted.

The events are shown in the following figure.

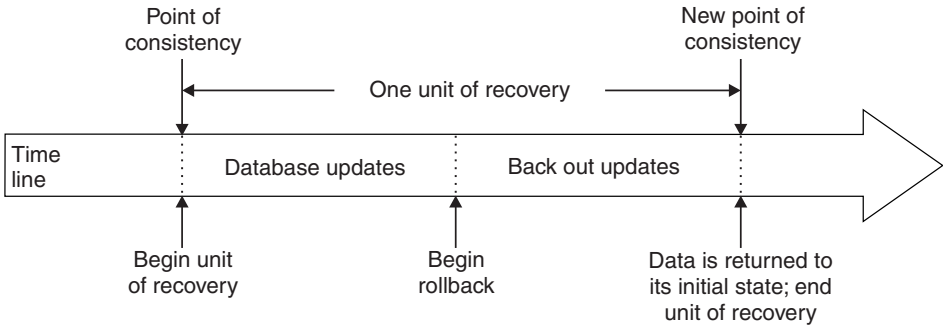


Figure 31. Unit of recovery (rolling back)

GUPI

The possible events that trigger "Begin rollback" in this figure include:

- SQL ROLLBACK statement
- Deadlock (reported as SQLCODE -911)
- Timeout (reported as SQLSTATE 40001)

The effects of inserts, updates, and deletes to large object (LOB) values are backed out along with all the other changes that were made during the unit of work that is being rolled back, even if the LOB values that were changed reside in a LOB table space that has 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 that are associated with the thread as refresh-pending (REFP) and puts the objects in a logical page list (LPL).

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).

GUPI

Related reference:

REFRESH-pending status (DB2 Utilities)

How the initial DB2 logging environment is established

The initial DB2 logging environment is established during installation of DB2.

Installation panels enable you 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.

Related reference:

DSNTIPH: System resource data set names panel (DB2 Installation and Migration)

How DB2 creates log records

Log records typically go through a standard life 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 also used to identify log records.)
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 that result from inserts, updates, and deletes are also logged as compressed data. Updates to compressed indexes are not logged as compressed data.

How DB2 writes the active log

DB2 writes the log buffers to an active log data set in response to several conditions. The most common condition is that the DB2 subsystem forces the log buffer to be written.

DB2 also writes the log buffers to an active log data set when they become full, or when the write threshold is reached.

When DB2 forces the log buffer to be written (such as at commit time), 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 that are 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.

How DB2 writes (offloads) the archive log

The process of copying active logs to archive logs is called *offloading*.

The relationship of offloading to other logging events is shown schematically in the following figure.

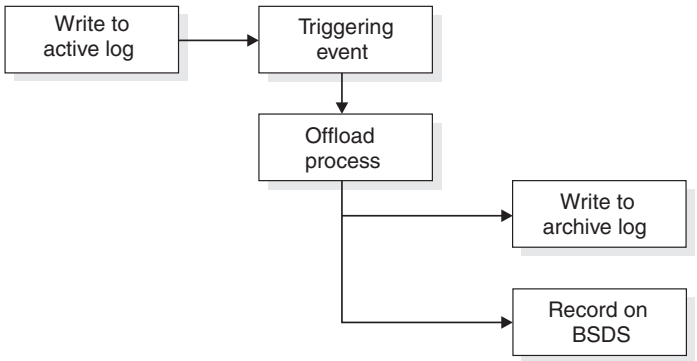


Figure 32. The offloading process

During the process, DB2 determines which data set to offload. Using the last log relative byte address (RBA) that was 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, DB2 cannot continue doing any work that requires writing to the log.

Related information:

"Recovering from active log failures" on page 488

What triggers an offload

An offload of an active log to an archive log can be triggered by several events.

The most common situations that trigger an offload include:

- An active log data set is full.
- DB2 starts, but an active log data set is full.
- The ARCHIVE LOG command is issued.

An offload can be also triggered by two uncommon events:

- An error occurs 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 a normal end-of-file condition. With dual active logs, both copies are truncated so the two copies remain synchronized.
- The last unarchived active log data set becomes full. 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 issues the following message and stops processing until offloading occurs.

  ```
  DSNJ111E - OUT OF SPACE IN ACTIVE LOG DATA SETS
  ```

Role of the operator in the offload process

When it is time to offload an active log, you can send a request to the z/OS console operator to mount a tape or prepare a disk unit.

The value of the WRITE TO OPER field 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 uses all the 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.

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 of a possible 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. The operator can cancel and then restart the offload.

Messages that are returned during offloading

During the offload process, DB2 sends a series of messages to the z/OS console. Most of these messages include information about 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 is to 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=...
  ```

- One of the following message appears when offload reaches end-of-volume or end-of-data-set in an archive log data set:

  The non-data sharing version of this message is:

  ```
  DSNJ003I - FULL ARCHIVE LOG VOLUME DSNAME=..., STARTRBA=..., ENDRBA=...,
             STARTTIME=..., ENDTIME=..., UNIT=..., COPYnVOL=...,
             VOLSPAN=..., CATLG=...
  ```

  The data sharing version of this message 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 performed on 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
  ```

Effects of interruptions and errors on the offload process

DB2 can handle some types of interruptions during the offloading process.

DB2 handles interruptions to the offloading process in the following ways:

- The **STOP DB2** command 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 "Recovering from active log failures" on page 488, or write I/O errors on the archive log, under "Recovering from archive log failures" on page 492.

- 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.

Archive log data sets

Archive log data sets can be placed on standard tapes or disks and can be managed by DFSMShsm (Data Facility Hierarchical Storage Manager). Archive logs are always written by QSAM.

Prior to DB2 9, archive logs on tape are read by BSAM. Those on disk are read by BDAM. In DB2 9 and later, they are always read by using BSAM. The block size of an archive log data set is a multiple of 4 KB.

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 that are 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.

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.

Related reference:

DSNTIPA: Archive log data set parameters panel (DB2 Installation and Migration)

DSNTIPH: System resource data set names panel (DB2 Installation and Migration)

How dual archive logging works

Each log control interval (CI) that is 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 ends-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, on installation panel DSNTIPA enables you to specify that the primary copy of the archive log go to disk and the secondary copy go to tape.

Dual archive logging increases recovery speed without using as much disk. The second tape is intended as a backup, or it 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 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.

Tips for archiving

You can archive to tape or disk.

Tips for archiving to tape:

If you choose to archive to tape, following certain tips can help you avoid problems.

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. Although a file sequence number of 1 might appear to be an error in the integrated catalog facility catalog, be aware that this situation causes no problems in DB2 processing.

If you choose to offload to tape, consider adjusting the size of your active log data sets so that each data 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 it 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 that is required for the BSDS.

Tips for archiving to disk:

If you choose to archive to disk, following certain tips can help you avoid problems.

All archive log data sets that are allocated on disk must be cataloged. If you choose to archive to disk, 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, ensure 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. The possibility of unwanted z/OS B37 or E37 abends during the offload process is thereby minimized. Primary space allocation is set with the PRIMARY QUANTITY field of the DSNTIPA installation panel. If the archive log is not allocated with the DSNTYPE=LARGE attribute, then the primary space quantity must be less than 64KB tracks because of the DFSMS Direct Access Device Space Management limit of 64KB tracks on a single volume when allocating a sequential disk data set without the DSNTYPE=LARGE attribute.

Tips for archiving with DFSMS:

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, override this candidate volume count by specifying YES for the field SINGLE VOLUME on installation panel DSNTIPA. This enables 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, ensure that the value of the CATALOG subsystem parameter is YES. Even if it does not, message DSNJ072E is returned, and DB2 forces the data set to be cataloged.

DB2 uses the basic sequential access method (BSAM) to read archive logs from disk and BSAM supports the use of Extended Format (EF) 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, not when the data is written.

Related concepts:

DB2 and DFSMS (Introduction to DB2 for z/OS)

Related reference:

CATALOG DATA field (CATALOG subsystem parameter) (DB2 Installation and Migration)

Automatic archive log deletion

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.

The default for the retention period keeps archive logs forever. Any other retention period 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, you need an archive log retention period of at least 120 days. For more than two cycles, you need a correspondingly longer retention period.

Tip: In case of data loss or corruption, retain all DB2 recovery log and image copy data sets until you can identify the source of the data loss or corruption. You might need these data sets to aid in nonstandard recovery procedures. On at least a weekly basis, take an image copy backup of every DB2 table space object that was updated and retain the DB2 recovery log data sets for a minimum of seven days. If you detect data loss or corruption, prevent older DB2 recovery log and image copy data sets from being discarded until the issue is assessed and resolved.

If archive log data sets or tapes are deleted automatically, the operation does not update the archive log data set inventory in the BSDS. You can update the BSDS with the change log inventory utility. This update is not required and recording old archive logs in the BSDS wastes space. However, it does no harm because the archive log data set inventory wraps and automatically deletes the oldest entries.

Related concepts:

"Management of the bootstrap data set" on page 360

"Recommendations for changing the BSDS log inventory" on page 362

Related reference:

DSNTIPA: Archive log data set parameters panel (DB2 Installation and Migration)

How DB2 retrieves log records

Normal DB2 operation and recovery tasks rely on the availability of log records. DB2 retrieves log records from different sources, depending on the situation.

About this task

Log records are retrieved by DB2 through the following events:

- A log record is requested using its RBA.
- DB2 searches for the log record in the following locations in the order in which they are presented:
  1. The log buffers.
  2. 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.
  3. The archive logs. DB2 determines which archive volume contains the CIs, dynamically allocates the archive volume, acquires a buffer, and reads the CIs.

Managing the log

You can control and monitor log activity by using several DB2 commands and a utility.

Quiescing activity before offloading

You can use the MODE(QUIESCE) option of the ARCHIVE LOG command to ensure that activity has stopped before the log is archived.

About this task

GUPI

With this option, DB2 work is 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. Using MODE(QUIESCE) 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.

The MODE(QUIESCE) option suspends all new update activity on DB2 up to the maximum period of time that is specified on the installation panel DSNTIPA. If the time needed to quiesce is less than the time that is specified, 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 they 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 that are 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   MYPLAN   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.

Regardless of whether the quiesce is successful, 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.
```

GUPI

Related reference:

DSNTIPA: Archive log data set parameters panel (DB2 Installation and Migration)

-ARCHIVE LOG (DB2) (DB2 Commands)

Archiving the log

If you are a properly authorized operator, you can archive the current DB2 active log data sets when necessary by issuing the ARCHIVE LOG command. Using the ARCHIVE LOG command can help with diagnosis by enabling you to quickly offload the active log to the archive log, where you can use DSN1LOGP to further analyze the problem.

Before you begin

You must have either SYSADM authority or have been granted the ARCHIVE privilege.

Procedure

GUPI

To archive the log:

Enter the following command:

```
-ARCHIVE LOG
```

When you issue the preceding command, DB2 truncates the current active log data sets, 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 that is written in the active log data set that is being truncated.

Example

You can 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.

Canceling log offloads

About this task

In some cases, the offload of an active log might 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, the DB2 active log data sets become full and DB2 stops logging.

To cancel (and retry) an offload, issue this command:

```
-ARCHIVE LOG CANCEL OFFLOAD
```

When you enter the command, DB2 restarts the offload, 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.

GUPI

Adding an active log data set to the active log inventory with the SET LOG command

You can use the SET LOG command to add a new active log data set to the active log inventory without stopping and starting DB2.

Before you begin

Before you issue the SET LOG command, define and format the new active log data sets.

1. Use the Access Method Services DEFINE command to define new active log data sets.
2. Use the DSNJLOGF utility to preformat the new active log data sets.

   If you do not preformat the active logs with the DSNJLOGF utility, the DB2 database manager needs to preformat them the first time that they are used, which might have a performance impact.

Procedure

To add an active log data set to the active log inventory:

Issue the SET LOG command with the NEWLOG and COPY keywords. If the DB2 database manager can open the newly defined log data set, the log data set is added to the active log inventory in the bootstrap data set (BSDS). The new active log data set is immediately available for use without stopping and starting the database manager.
Currently, the database manager uses active log data sets in the reverse order from the order in which you add them to the active log inventory with the SET LOG command. For example, suppose that the active log inventory contains data sets DS01, DS02, and DS03, and you add data set DS04 and then data set DS05. If data

set DS03 is active, and you issue the ARCHIVE LOG command, the new active log becomes DS05. This behavior might change in the future, so schemes for adding and switching active logs should not depend on this order.

Related reference:

-SET LOG (DB2) (DB2 Commands)

DSNJLOGF (preformat active log) (DB2 Utilities)

Dynamically changing the checkpoint frequency

You can use the LOGLOAD option, the CHKTIME option, or a combination of both of these options of the SET LOG command to dynamically change the checkpoint frequency without recycling DB2.

About this task

GUPI

The LOGLOAD value specifies the number of log records that DB2 writes between checkpoints. The CHKTIME value specifies the number of minutes between checkpoints.

You specify the initial LOGLOAD and CHKTIME parameter values at installation time. Also at installation time, you specify which of these values controls when a checkpoint occurs, or whether both do. If you specify both the LOGLOAD and CHKTIME options together, when the threshold for one is reached, a system checkpoint is taken, and the counters for both thresholds are reset. In the following example, the SET LOG command changes checkpoint scheduling to use both log records and time:

```
-SET LOG BOTH CHKTIME(10) LOGLOAD(500000)
```

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 also can use either the LOGLOAD option or the CHKTIME option to initiate an immediate system checkpoint. For example:

```
-SET LOG LOGLOAD(0)
-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.

GUPI

Related reference:

-SET LOG (DB2) (DB2 Commands)

Setting limits for archive log tape units

Use the DB2 SET ARCHIVE command to set the upper limit for the number of, and the deallocation time of, tape units for the archive log.

About this task

GUPI

This command overrules the values that are specified during installation, or in a previous invocation of the SET ARCHIVE command. The changes that are initiated by the SET ARCHIVE command are temporary. At restart, DB2 uses the values that are set during installation.

GUPI

Related reference:

-SET ARCHIVE (DB2) (DB2 Commands)

Monitoring the system checkpoint

DB2 schedules a system checkpoint every time it switches active log data sets, regardless of the currently defined checkpoint frequency.

About this task

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, quiesce activity and terminate DB2 to minimize the restart time.

Procedure

To display the most recent checkpoint, use one of the following approaches:

- Issue the DISPLAY LOG command.
- Run the print log map utility (DSNJU004).

Related tasks:

"Displaying log information"

Related reference:

DSNJU004 (print log map) (DB2 Utilities)

-DISPLAY LOG (DB2) (DB2 Commands)

Displaying log information

You can use the DISPLAY LOG command to display the current checkpoint frequency. You can obtain additional information about log data sets and checkpoints from the print log map utility (DSNJU004).

About this task

The checkpoint frequency can be either the number of log records or the minutes between checkpoints.

Procedure

To display log information:

Issue the DISPLAY LOG command, or use the print log map utility (DSNJU004).

Related reference:

-DISPLAY LOG (DB2) (DB2 Commands)

-SET LOG (DB2) (DB2 Commands)

DSNJU004 (print log map) (DB2 Utilities)

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.

About this task

Procedure

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'F00000000000', 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 hexadecimal 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.

Related tasks:

"Resetting the log RBA value in a non-data sharing environment" on page 354

"Resetting the log RBA value in a 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 must reset the log RBA value for that member.

Procedure

To reset the log RBA value in a data sharing environment:

1. Issue the **STOP DB2** command to quiesce the member that is approaching the end of the log RBA range.
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 using another member is an acceptable solution, you can leave the original member stopped indefinitely.
7. Optional: Before you cold start the member, complete this step to avoid a potential performance issue that is caused by log formatting. When the residual RBA range is greater than the log truncation point, DB2 resets the high used RBA (HURBA) for the active logs after the cold start of a member. This action avoids log read errors that can result from reading residual log data with higher RBA values from peer members of the data sharing group. The first time the logs become current, DB2 must format the active logs ahead of the log writes until the log is full.

   To preformat the active logs before a cold start:

   a. Delete and redefine the active logs with IDCAMS.
   b. Format the empty logs by using the DSNJLOGF utility.
   c. Use the DSNJU003 utility to delete the active logs from the BSDS and add the logs back in with no RBA range. Adding the logs back in with no RBA range shows that the logs are empty.

During the subsequent cold start, DB2 detects these changes to the active logs and does not reset the HURBA. Therefore, log formatting is not required ahead of the log writes.

8. 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. Make a full image copy of all data by using the COPY utility. For example, in a data sharing group with two members, where you are resetting the log RBA of the first member, make a full image copy by using the second member. The first member must remain quiesced for awhile. The length of time that the member is quiesced depends on how long it takes to establish a new recovery base and the time required for the logs from the quiesced member to no longer be required for any form of recovery. After you are satisfied that the logs are not required for recovery purposes, you can reset the log RBA for the quiesced member and restart the member.
   b. Cold start the first 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
      ```

Related concepts:

"Log RBA range" on page 353

Related reference:

COPY (DB2 Utilities)

DSNJU003 (change log inventory) (DB2 Utilities)

DSNJLOGF (preformat active log) (DB2 Utilities)

Resetting the log RBA value in a non-data sharing environment

Before a 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.

Before you begin

Having full backups of all DB2 subsystem volumes is recommended, including volumes with active and archive log data sets and the BSDSs.

Procedure

To reset the log RBA value in a non-data sharing environment by using the COPY utility:

1. Drop any user-created indexes on the SYSIBM.SYSCOPY catalog table.
2. Alter all of the indexes on the catalog tables so that they have the COPY YES attribute by issuing the ALTER INDEX COPY YES statement, and commit the changes after running every 30 ALTER statements.

   Tip: You do not need to alter the following objects:
   - DB2 directory indexes. By default, these indexes already have the COPY YES attribute. Issue a DISPLAY DATABASE(DSNDB01) SP(*) command to have a list of all the directory table spaces and indexes for later steps.
   - The DB2-defined indexes for the SYSIBM.SYSCOPY table. You will be resetting the log RBA value for these indexes later in this procedure.

3. Issue the **-STOP DB2** command to quiesce the subsystem that is approaching the end of the log RBA range.
4. Restart DB2 in ACCESS(MAINT) mode.
5. Issue the **-DISPLAY THREAD** command. Ensure that there are no INDOUBT or POSTPONED ABORT units of recovery.
6. Issue the **-DISPLAY UTILITY** command. Ensure there are no active or stopped utilities.
7. Issue the **-DISPLAY DATABASE(*) SPACENAM(*) RESTRICT** command. Ensure that all restricted states are removed.
8. Quiesce the DB2 subsystem again by issuing the **-STOP DB2** command.
9. 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 members DSNTIJDE, DSNTIJIN, DSNTIJID, and DSNTIJIE in library SDSNSAMP. For DB2 10 subsystems at CM8, you must use the SDSNSAMP library for DB2 for z/OS Version 8. Also, for DB2 10 subsystems at CM9, you must use the SDSNSAMP library for DB2 for z/OS Version 9.

   CAUTION:
   Edit these members so that only the pertinent page sets are processed.
10. Complete the following steps to enable the COPY utility to reset the log RBA values in data pages and index pages as they are copied:
    a. Edit member DSN6SPRC of the prefix.SDSNMACS library and locate the entry SPRMRRBA.
    b. Change the SPRMRRBA setting to '1' and save the change.
    c. Run the first two steps of your customized copy of job DSNTIJUZ to rebuild your DB2 subsystem parameter module (DSNZPxxx).
11. Cold start this subsystem back to the RBA value of 0 (zero). This step removes all log data from the BSDS. This step requires utility DSNJU003 with the following options:

    ```
    CRESTART CREATE,STARTRBA=0,ENDRBA=0
    ```
12. Start the DB2 subsystem in ACCESS(MAINT) mode.
13. Take new, full image copies of all table spaces and indexes by using the COPY utility with the SHRLEVEL REFERENCE option to automatically reset the log RBA values.
    a. COPY the DSNDB06.SYSRTSTS catalog table space.
    b. Copy the rest of the catalog and directory table spaces and indexes, including any user-created DB2 catalog indexes.
    c. Alter indexes on user table spaces so that they have the COPY YES attribute by issuing the ALTER INDEX COPY YES statement, and commit the changes after running every 30 ALTER statements.
    d. Copy the table spaces and indexes in user-created databases. You can do this step and the next step in parallel.
    e. Copy the tables spaces and indexes in default database DSNDB04.

    Do not copy the table spaces in workfile database DSNDB07. DB2 automatically resets the log RBA values in these table spaces when they are used.

    Restriction: Because FlashCopy technology does not reset the log RBA, you cannot use the COPY FLASHCOPY option in this situation.

14. Re-create any user-created indexes on the SYSIBM.SYSCOPY table that were dropped in step 1 of this procedure.
15. Verify that the log RBA values were reset:
    a. Run a query against the tables SYSIBM.SYSCOPY, SYSIBM.SYSTABLEPART, and SYSIBM.SYSINDEXPART to verify that all objects were copied.
    b. Use the DSN1PRNT utility with the FORMAT option to print several pages from some of the objects so that you can verify that the PGLOGRBA field in the pages are reset to zero. The COPY utility updates the PGLOGRBA field and other RBA fields in header pages (page zero) so these fields will contain non-zero values.
16. Stop DB2, and disable the reset RBA function in the COPY utility by following the instructions in step 10 and setting SPRMRRBA to '0'.
17. Restart DB2 for normal access.
18. Alter the DB2 catalog indexes and user-created indexes to have the COPY NO attribute by issuing the ALTER INDEX COPY NO statement. Commit the changes after every thirty ALTER statements. However, you should issue these ALTER statements over several days, because during this process SYSCOPY and SYSLGRNX records are deleted and contention might occur.

    Note: If the RBA fields for an object are not reset, abend04E RC00C200C1 is returned during SQL update, delete, and insert operations. The object also is placed in STOPE status. You can use the DSN1COPY utility with the RESET option to reset the log RBA values. This two-step process requires copying the data out and then back into the specified data sets. Before using DSN1COPY with the RESET option, make sure that the object is stopped by issuing the command **-STOP DB(...) SPACENAM(...)**.

Related concepts:

"Log RBA range" on page 353

Related reference:

COPY (DB2 Utilities)

Canceling and restarting an offload

If the offload task remains stalled, the active logs eventually become full and DB2 stops database update activity.

Procedure

GUPI

To cancel and restart an offload task:

Issue the ARCHIVE LOG CANCEL OFFLOAD command.

GUPI

Displaying the status of an offload

You can use the DISPLAY LOG command to view information about the current active log data sets and status of the offload task.

Procedure

GUPI

To view the status of an offload task:

Issue the DISPLAY LOG command.

GUPI

Related reference:

-DISPLAY LOG (DB2) (DB2 Commands)

Discarding archive log records

You must keep enough log records to recover units of work and databases.

About this task

To recover units of recovery, you need log records at least until all current actions are completed. If DB2 terminates abnormally, 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. If all counts are zero, no unit-of-recovery actions are pending. If indoubt units of recovery remain, identify and recover them by the methods described in Chapter 8, “Monitoring and controlling DB2 and its connections,” on page 253.

To recover databases, you need log records and image copies of table spaces. How long you keep log records depends, on how often you make those image copies. If you do not already know what records you want to keep, see Chapter 12, “Backing up and recovering your data,” on page 395 for suggestions about recovery cycles.

Locating archive log data sets

To ensure that you can recover your data in the event of a failure, you need to locate archive log data sets.

Before you begin

In preparation, you must:

- Keep all the logs that have been written since the most recent checkpoint of DB2, so that DB2 can restart.
- Keep all the logs for two or more complete image copy cycles of your least-frequently copied table space.

About this task

You can discard active data sets, based on their log RBA ranges. The earliest log record that you need to retain is identified by a log RBA. You can discard any archive log data sets that contain only records with log RBAs that are lower than that RBA.

Procedure

GUPI To locate archive log data sets:

1. Resolve indoubt units of recovery. If DB2 is running with TSO, continue with step 2. If DB2 is running with IMS, CICS, or distributed data, the following substeps apply:
   a. Ensure that the period between one startup and the next startup is free of any indoubt units of recovery. Ensure that no DB2 activity is going on when you are performing this set of substeps. (To minimize impact on users, consider planning this work for a non-prime shift.) To determine whether indoubt units of recovery exist, issue the following DB2 command:

      ```
      -DISPLAY THREAD TYPE(INDOUBT)
      ```

      If you find no indoubt units of recovery, skip to step 2.
   b. If one or more indoubt units of recovery exist, take one of the following actions:
      - If IMS or CICS is involved with the indoubt units of work, start IMS or CICS. Starting IMS or CICS causes 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, issue the following command to resolve the unit of work:

        ```
        -RECOVER INDOUBT
        ```
      - Issue the following DB2 command:

        ```
        -RECOVER INDOUBT
        ```
   c. Reissue the **DISPLAY THREAD TYPE(INDOUBT)** command to ensure that the indoubt units have been recovered. When no indoubt units of recovery remain, continue with step 2.
2. Find the startup log RBA. Keep at least all log records with log RBAs greater than the one that is given in this message, which is issued at restart:

   ```
   DSNR003I RESTART...PRIOR CHECKPOINT RBA=xxxxxxxxxxxx
   ```

   If you suspended DB2 activity while performing step 1, restart DB2 now.
3. Find the minimum log RBA that is 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 that you want to keep.
   a. If you have any table spaces that were created so recently 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.

      GUPI The following SQL statement generates a list of the table spaces for which no full image copy is available:

      ```
      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;
      ```

      GUPI
   b. Issue the following SQL statement to find START_RBA values:

GUPI

```
SELECT DBNAME, TSNAME, DSNUM, ICTYPE, TIMESTAMP, HEX(START_RBA)
  FROM SYSIBM.SYSCOPY
  ORDER BY DBNAME, TSNAME, DSNUM, TIMESTAMP;
```

GUPI

The statement generates a list of all databases and the table spaces within them, in ascending order by date.

c. Find the START_RBA value 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 plan to discard records from SYSIBM.SYSCOPY and SYSIBM.SYSLGRNX, note the date of the earliest image copy that you want to keep.

4. Use job DSNTIJIC to copy all catalog and directory table spaces. Doing so ensures that copies of these table spaces are included in the range of log records that you plan to keep.

5. Locate and discard archive log volumes. Now that you know the minimum log RBA, from step 3, suppose that you want to find archive log volumes that contain only log records earlier than that. Proceed as follows:

   a. Execute the print log map utility (DSNJU004) to print the contents of the BSDS.

   b. Find the sections of the output titled "ARCHIVE LOG COPY *n* DATA SETS". (If you use dual logging, two sections exist.) The STARTRBA and ENDRBA columns in the output show the range of log RBAs that are contained in each volume. Find the volumes (two, for dual logging) whose ranges include the minimum log RBA that you found in step 3. These volumes are the earliest volumes that you need to keep.

      If no volumes have an appropriate range, one of the following cases applies:

      - The minimum log RBA 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 maximum number that is allowed. The maximum number is specified in the RECORDING MAX field (MAXARCH subsystem parameter) of installation panel DSNTIPA. If the BSDS does not register an archive log volume, it can never be used for recovery. Therefore, consider adding information about existing volumes to the BSDS.

        Also, consider increasing the value of the MAXARCH subsystem parameter to change the maximum number of archive log volumes that are to be recorded in the BSDS.

   c. 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 that 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 that you can discard all entries above the entry for the archive log that contains the minimum log RBA.

   d. 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 (DSNJU003) to delete entries for the discarded volumes. GUPI

Related tasks:

"Resolving indoubt units of recovery" on page 386

Related reference:

DSNTIPA: Archive log data set parameters panel (DB2 Installation and Migration)

DSNJU003 (change log inventory) (DB2 Utilities)

DSNJU004 (print log map) (DB2 Utilities)

Management of the bootstrap data set

The *bootstrap data set (BSDS)* is a VSAM key-sequenced data set that contains information about the log data sets and the records that those data sets include. The BSDS 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.

The active logs are first registered in the BSDS by job DSNTIJID, during DB2 installation. They cannot be replaced, nor can new ones be 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 the list wraps around when a user-determined number of entries is reached. The maximum number of archive log data sets that DB2 keeps in the BSDS depends on the value of the MAXARCH subsystem parameter. The allowable values for the MAXARCH subsystem parameter are between 10 and 10,000 (inclusive). If two copies of the archive log are being created, the BSDS will contain records for both copies, resulting in between 20 and 20,000 entries.

You can manage the inventory of archive log data sets with the change log inventory utility (DSNJU003).

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 is scratched by a tape management system following the expiration of the retention period of the data set.

Conversely, the maximum number of archive log data sets might be exceeded, and the data from the BSDS might be dropped long before the data set reaches its expiration date.

If you specified at installation that archive log data sets are to be cataloged when allocated, the BSDS points to the integrated catalog facility catalog for the information that is 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.

Related concepts:

Automatic archive log deletion

Related reference:

DSNJCNVB (DB2 Utilities)

Restoring dual-BSDS mode

In dual-BSDS mode, 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-BSDS mode.

Procedure

To restore dual-BSDS mode:

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 RECOVER BSDS command to make a copy of the good BSDS in the newly allocated data set.

BSDS copies with archive log data sets

Every time that a new archive log data set is created, a copy of the BSDS is also created either on tape or on disk.

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.

Recommendation: For better offload performance and space utilization, use a block size of 28672 for tape or 24576 for disk. (The default value is 24576.) If required, you can change this value in the BLOCK SIZE field on installation panel DSNTIPA. Also adjust the PRIMARY QUANTITY and SECONDARY QUANTITY fields 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 occurs 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.

Related reference:

DSNJU004 (print log map) (DB2 Utilities)

Recommendations for changing the BSDS log inventory

You do not need 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 or change the DDF communication record.

You can change the BSDS by running the DB2 batch change log inventory (DSNJU003) utility. You can only run this utility when DB2 is inactive.

You can copy an active log data set using the access method services IDCAMS REPRO statement. The copy can be performed when only DB2 is down, because DB2 allocates the active log data sets as exclusive (DISP=OLD) at DB2 startup. .

Related reference:

DSNJU003 (change log inventory) (DB2 Utilities)

Related information:

REPRO command (DFSMS Access Method Services for Catalogs)

Chapter 10. Restarting DB2 after termination

When you need to restart DB2 after DB2 terminates normally or abnormally, keep in mind these considerations, which are important for backup and recovery, and for maintaining consistency.

About this task

The term *object*, used in any discussion of restarting DB2 after termination, refers to any database, table space, or index space.

Related concepts:

Chapter 11, "Maintaining consistency across multiple systems," on page 379

Related tasks:

Chapter 12, "Backing up and recovering your data," on page 395

Methods of restarting

DB2 can restart in several different ways. Some options are based on how DB2 terminated or what your environment is.

Types of termination

DB2 terminates normally in response to the **STOP DB2** command . 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.

GUPI

You can use either STOP DB2 MODE (QUIESCE) or STOP DB2 MODE (FORCE). The effects of each command are compared in the following table.

Table 32. 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 STOP DB2 MODE(QUIESCE) command, 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:

- Open, held cursors exist.
- Special registers are not in their original state.

© Copyright IBM Corp. 1982, 2016

Before DB2 can stop, 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 stops the CICS attachment facility, so an active task might not necessarily run to completion.

For example, assume that a CICS transaction opens no cursors that are 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.

A data object might 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 outstanding units of recovery become inflight units of recovery when DB2 is restarted; then they are returned to their original states.

GUPI

Abnormal terminations (abends)

An abnormal termination, or *abend*, is said to happen when DB2 does not terminate in an orderly way.

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" on page 365
"Phase 2: Current status rebuild" on page 366
"Phase 3: Forward log recovery" on page 367
"Phase 4: Backward log recovery" on page 368

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 "Consistency after termination or failure" on page 384.

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. Installation options LIMIT BACKOUT and BACKOUT DURATION 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.

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. The UR log threshold option is established at installation time, through field UR LOG WRITE CHECK on panel DSNTIPL.

Restart of large object (LOB) table spaces is like restart of other table spaces. LOB table spaces that are 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.

After DB2 has gone through a group or normal restart that involves group buffer pool (GBP) failure, group buffer pool recovery pending (GRECP) can be automatically initiated for all objects except the object that is explicitly deferred during restart (ZPARM defer), or the object that is associated with the indoubt or postponed-abort UR.

Related reference:

DSNTIPL: Active log data set parameters (DB2 Installation and Migration)

Phase 1: Log initialization

During the first restart phase, DB2 attempts to locate the last log RBA that was written before termination. Logging continues at the subsequent RBA.

In phase 1:

1. DB2 compares the high-level qualifier of the integrated catalog facility catalog name that is 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 366.
   - 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 probably have the wrong parameter module. Start DB2 with the command START DB2 PARM(*module-name*), and name the correct module.

2. DB2 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.
   - 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 that are not noted in the BSDS would be unknown to the system.
3. DB2 finds in the BSDS the log RBA of the last log record that was 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.
4. DB2 scans the log forward, beginning at the log RBA of the most recent log record, up to the last control interval (CI) that was written before termination.
5. DB2 prepares to continue writing log records at the next CI on the log.
6. DB2 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.

Related reference:

DSNTIPL: Active log data set parameters (DB2 Installation and Migration)

Related information:

"Recovering from BSDS failures" on page 495

Phase 2: Current status rebuild

During the second restart phase, 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:

1. DB2 checks the BSDS to find the log RBA of the last complete checkpoint before termination.

2. DB2 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. DB2 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 that exist for each database and all image copy information that is related to the DSNDB01.SYSUTILX, DSNDB01.DBD01, DSNDB06.SYSCOPY, and DSNDB01.SYSDBDXA table spaces.
   - All objects that are open at the time of termination.
   - How far back in the log to go to reconstruct data pages that were not written to disk.

   The number of log records that are written between one checkpoint and the next is set when DB2 is installed.

   You can temporarily modify the checkpoint frequency by using the command SET LOG. The value that you specify persists while DB2 is active; on restart, DB2 uses the checkpoint type and frequency that is specified on installation panel DSNTIPL1.
4. DB2 issues message DSNR004I, which summarizes the activity that is required at restart for outstanding units of recovery.
5. DB2 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.

Related reference:

-SET LOG (DB2) (DB2 Commands)

DSNTIPL: Active log data set parameters (DB2 Installation and Migration)

Phase 3: Forward log recovery

During the third restart phase, DB2 completes the processing for all committed changes and database write operations.

DB2 processing during phase 3 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
- In the case of group restarts in which locks have been lost, acquiring locks to prevent access to data that is in use by those units of recovery

Restart time is longer when lock information needs to be recovered during a group restart, because DB2 needs to go back to the earliest begin_UR for an inflight UR belonging to that subsystem. This is necessary to rebuild the locks that member has obtained during the inflight UR. (A normal restart goes back only as far as the earliest RBA that is needed for database writes or is associated with the begin_UR of indoubt units of recovery.)

DB2 executes these steps:

1. DB2 detects whether a page set that is 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 by using one of the methods described in "Recovering from a down-level page set problem" on page 537.

2. DB2 scans the log forward, beginning at the lowest (earliest) log RBA that is either required for completion of database writes or that 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. DB2 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. DB2 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. DB2 writes pages to disk as the need for buffers demands it.
6. DB2 marks the completion of each unit of recovery that is processed. If restart processing terminates later, those units of recovery do not reappear in status lists.
7. DB2 stops scanning at the current end of the log.
8. DB2 writes to disk all modified buffers that are not yet written.
9. DB2 issues message DSNR005I, which summarizes the number of remaining in-commit or indoubt units of recovery. No in-commit units of recovery should exist because all processing for these units should have completed. The number of indoubt units of recovery should be equal to the number that is specified in the previous DSNR004I restart message.
10. DB2 issues message DSNR007I, 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.

Related information:

Recovering from a down-level page set problem

Phase 4: Backward log recovery

During the fourth restart phase, DB2 changes that were performed for inflight or in-abort units of recovery are reversed.

In phase 4:

1. DB2 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 verifies 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. DB2 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, so 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 and DB2 ignores it.
3. DB2 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.
   - The page set is deferred at restart using the DEFER subsystem parameter.
4. DB2 writes pages to disk as the need for buffers demands it.
5. DB2 writes to disk all modified buffers that have not yet been written.
6. DB2 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. DB2 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, DB2 reacquires write claims for the objects on behalf of the indoubt and postponed-abort units of recovery.
9. DB2 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.

Automatic restart

If you run 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.

Restart in a data sharing environment

In a data sharing environment, restart processing coordinates data recovery across more than one DB2 subsystem.

When certain critical resources are lost, restart includes additional processing to recover and rebuild those resources. This process is called *group restart*.

Related concepts:

Group restart phases (DB2 Data Sharing Planning and Administration)

Restart implications for table spaces that are not logged

Even if all tables that are modified by a transaction reside in table spaces that are not logged, a unit of recovery is established before any of those updates are performed. Undo processing continues to read the log in the backward direction, looking for undo log records that need to be applied. Undo processing stops when it detects the beginning of this unit of recovery as recorded on the log.

Therefore you still need to perform frequent commits to limit the distance that undo processing might need to go backward on the log to find the beginning of the unit of recovery. If a transaction that is not logged does not do frequent commits, it is subject to being reported as a long-running unit of recovery in message DSNR035I. This means that if such a unit of recovery persists for a duration that qualifies as a long-running unit of recovery, message DSNR035I is issued to identify it.

If, during restart, you need to undo work that has not been logged because of the NOT LOGGED attribute, the table space loses its data integrity, and therefore must be recovered. Recovery can be accomplished by using the RECOVER utility or by reinserting the appropriate data. For example, a summary table can be re-created by one or more INSERT statements; a materialized query table can be rebuilt by using a REFRESH TABLE SQL statement.

To mark the need for recovery, the table space or partition is marked with RECOVER-pending status. To prevent any access to the corrupt data, the table space or partition is placed in the LPL. When undo processing places a table space or partition in the logical page list (LPL) and marks it with RECOVER-pending status, it also places all of the updated indexes on all tables in the table space in the LPL. The corresponding partitions of data-partitioned secondary indexes (DPSIs) are placed in the LPL, which prevents other processes that use index-only access from seeing data whose integrity is in doubt. These indexes are also marked with REBUILD-pending status.

After restart, when DB2 is operational, if undo processing is needed for a unit of recovery in which modifications were made to the table space that was not logged, the entire table space or partition is placed in the LPL, and the table space is marked with RECOVER-pending status. This can happen, for example, as a result of a rollback, abort, trigger error, or duplicate key or referential constraint violation. The LPL ensures that no concurrently running agent can see any of the data whose integrity is in doubt.

Avoid duplicate key or referential constraint violations in table spaces that are not logged because the result is an unavailable table space that requires manual action.

When a table space or partition is placed in the LPL because undo processing is needed for a table space that is not logged, either at restart time or during rollback

processing, automatic LPL recovery is not initiated, and a START DATABASE command identifying this table space has no effect on its LPL status.

Conditional restart

A *conditional restart* is a DB2 restart that is directed by a user-defined conditional restart control record (CRCR).

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 that are 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.

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.

Related concepts:

Restarting a member with conditions (DB2 Data Sharing Planning and Administration)

Related reference:

DSNJU003 (change log inventory) (DB2 Utilities)

Terminating DB2 normally

Whenever possible, ensure that DB2 terminates normally.

Procedure

GUPI

To terminate DB2, issue either of the following commands:

- `-STOP DB2 MODE (QUIESCE)`
- `-STOP DB2 MODE (FORCE)`

During shutdown, use the command DISPLAY THREAD to check the shutdown progress. If shutdown is taking too long, you can issue STOP DB2 MODE (FORCE), but rolling back work can take as long as or longer than the completion of QUIESCE.

Results

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.

GUPI

Restarting automatically

You control how automatic restart works by using automatic restart policies.

About this task

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 that defines the action that you want taken.

Procedure

To create a policy:

Identify 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, the IRLM subsystem name, and the IRLM ID.

Related reference:

Adding MVS systems to a sysplex

Deferring restart processing

You have several options for deferring restart processing on an object.

Procedure

To defer restart processing, use one of the following approaches:

- To vary the device (or volume) on which the objects reside offline:

  If the data sets that contain 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.
- To delay the backout of a long-running UR:

  On installation panel DSNTIPL, you can use the following options:

  – LIMIT BACKOUT defined as YES, AUTO, LIGHT, or LIGHTAUTO indicates that some backout processing is to postponed when restarting DB2. 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, that are to be read during the backward log scan phase of restart.

  The amount of backout processing that is 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 DB2 stopped

  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 until the requested number of log records have been processed.

- To 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.

Deferral of restart

Usually, restarting DB2 activates restart processing for objects that were available when DB2 terminated (in other words, not stopped with the STOP DATABASE command). Restart processing applies or backs out log records for objects that have unresolved work.

Restart processing is controlled by specifications on installation panel DSNTIPS, and the default is RESTART ALL.

If some specific object is causing problems, one option is to defer 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 that are necessary for restart of the object 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 DEFER ALL is specified at a site that is designated as RECOVERYSITE in DSNZP*xxx*, all pages for an object that is deferred are placed in the LPL (as a page range, not as a large list of individual pages). 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 issued. Instead either DSNT500I or DSNT501I 'resource unavailable' is issued.
- For a deferred restart scenario that needs to recover all DB2 objects after DB2 is up, the recommendation is to set the DEFER and ALL subsystem parameters on panel DSNTIPS 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. When an error is encountered while accessing an index object during restart, the entire index is put in the LPL, not just the individual pages.

Performing conditional restart

Normal recovery operations can be partially or fully bypassed by using conditional restart control records in the BSDS.

Procedure

To perform a conditional restart:

1. Optional: 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.
2. While DB2 is stopped, run the change log inventory utility by using the CRESTART control statement to create a new conditional restart control record.
3. Restart DB2. The recovery operations that take place are governed by the current conditional restart control record.
4. Optional: For data sharing environments, use the LIGHT(YES) or LIGHT(NOINDOUBTS) parameter on the START DB2 command to quickly recover retained locks on a DB2 member.

Related concepts:

"Messages at start" on page 211

Conditional restart with system-level backups

You can recover a DB2 subsystem to a point in time (system point-in-time recovery) where the log copy pool is restored from a system-level backup. In this case, perform a conditional restart with a log truncation point that is less than or equal to the data complete log record sequence number (LRSN) of the system-level backup.

For example, create the conditional restart record by using the data complete LRSN as the CRESTART ENDRBA, ENDLRSN, or SYSPITR log truncation point for the change log inventory utility, DSNJU003. The data complete LRSN is externalized in the DSNU1614I message that is issued by the BACKUP SYSTEM utility when a system-level backup completes.

You can use the print log map utility, DSNJU004, to print the information for a system-level backup. This information includes the data complete LRSN. Using the DSNJU004 utility is beneficial if you did not preserve the job output or the log that contains the DSNU1614I message.

Options for recovery operations after 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
- Read the log forward to recover indoubt and uncommitted units of recovery
- Read the log backward to back out uncommitted and in-abort units of recovery
- 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 fails.

Conditional restart records

In addition to information describing the active and archive logs, the BSDS contains two queues of records that are associated with conditional restart.

The two queues are:

- A wrap-around queue of conditional restart control records. Each element in the queue records the choices that you made when you created the record and the progress of the restart operation that it controls. When the operation is complete, the use count is set at 1 for the record, and the record 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 that checkpoint queue is then used for that purpose.

You can use the utility DSN1LOGP to read information about checkpoints and conditional restart control records.

Related reference:

DSN1LOGP (DB2 Utilities)

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.

About this task

If you specify LIMIT BACKOUT = YES or LIGHT, you must use the RECOVER POSTPONED command to resolve postponed units of recovery.

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.

Related reference:

DSNTIPL: Active log data set parameters (DB2 Installation and Migration)

DROP (DB2 SQL)

-RECOVER POSTPONED (DB2) (DB2 Commands)

RECOVER POSTPONED command

Output from the RECOVER POSTPONED command consists of informational messages.

In the following figure, 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 33. Example of output from 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 that are encountered during recovery of 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.

Recovering from an error during RECOVER POSTPONED processing

When an error prevents logging of a compensation log record, DB2 terminates abnormally.

Procedure

If DB2 terminates abnormally:

1. Fix the error.
2. Restart DB2.
3. Re-issue the RECOVER POSTPONED command if automatic backout processing has not been specified.

   If the RECOVER POSTPONED processing lasts for an extended period, the output includes DSNR047I messages to help you monitor backout processing. These messages show the current RBA that is being processed and the target RBA.

Chapter 11. Maintaining consistency across multiple systems

Data consistency issues arise when DB2 acts in conjunction with other systems, such as IMS, CICS, or remote database management systems (DBMSs).

Multiple system consistency

DB2 can work with other DBMSs, including IMS, and other types of remote DBMSs through the distributed data facility (DDF). DB2 can also work with other DB2 subsystems through the DDF.

If data in more than one subsystem is to be consistent, all update operations at all subsystems for a single logical unit of work must either be committed or backed out.

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 ensures that either all the effects of the logical unit of work persist, or none of the effects persist. The commit process ensures this outcome despite component, system, or communications failures.

DB2 uses a two-phase commit process to communicate between subsystems. The two-phase commit process is controlled by one of the subsystems, called the *coordinator*. The other systems that are involved are called the *participants*. DB2, CICS, or WebSphere® Application Server are always the coordinator when they interact with DB2. In transactions that include those systems, 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.

The following figure 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.

© Copyright IBM Corp. 1982, 2016

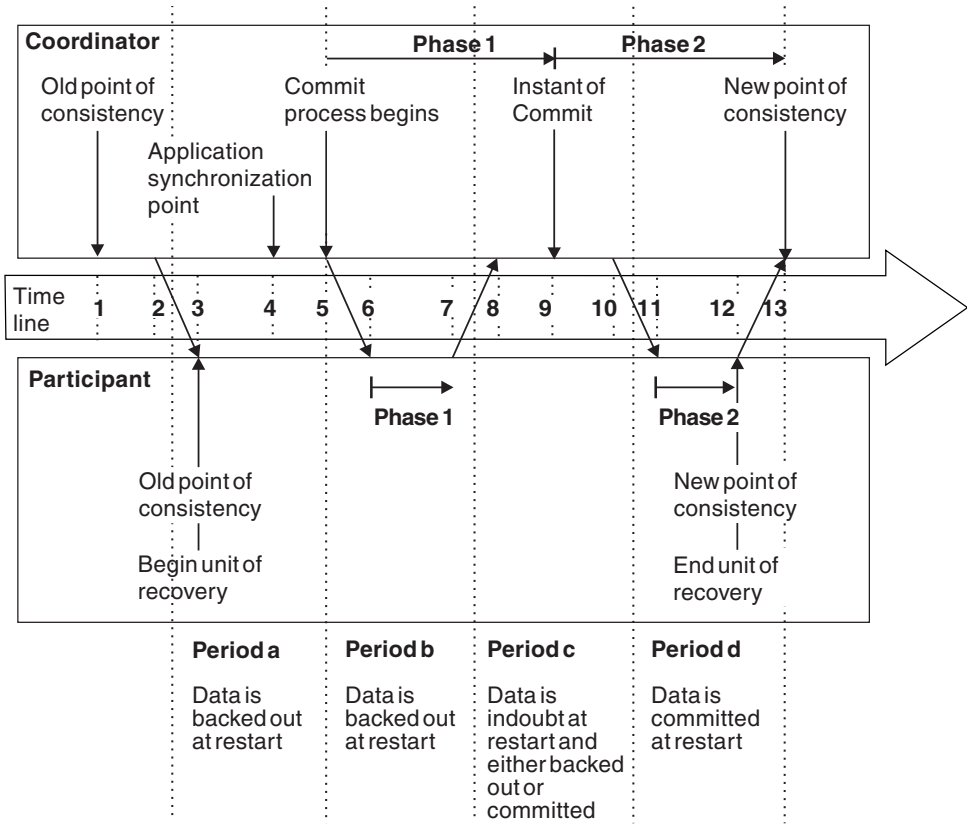


Figure 34. Time line illustrating a commit that is coordinated with another subsystem

The numbers below are keyed to the timeline 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. The coordinator notifies the participant that it can begin 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 that is controlled by both subsystems is now consistent and available to other applications.

On some occasions, 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 a SYNCPOINT is issued after performance of a series of SELECT statements or when end-of-task is reached immediately after a SYNCPOINT is 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.

Commit coordinator and multiple participants

The principles and methods for maintaining consistency across more than two systems are similar to those that are used to ensure consistency across two systems. The main difference involves the role of a system as coordinator or participant when a unit of work spans multiple systems.

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, the DB2 coordinator communicates 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, CICS, or WebSphere Application Server systems. However, DB2 can also serve as the coordinator for other DBMSs or for other 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 the following figure, DB2A is the participant for an IMS transaction, but DB2 becomes the coordinator for the two database servers (AS1 and AS2), for DB2B, and for its respective DB2 servers (DB2C, DB2D, and DB2E).

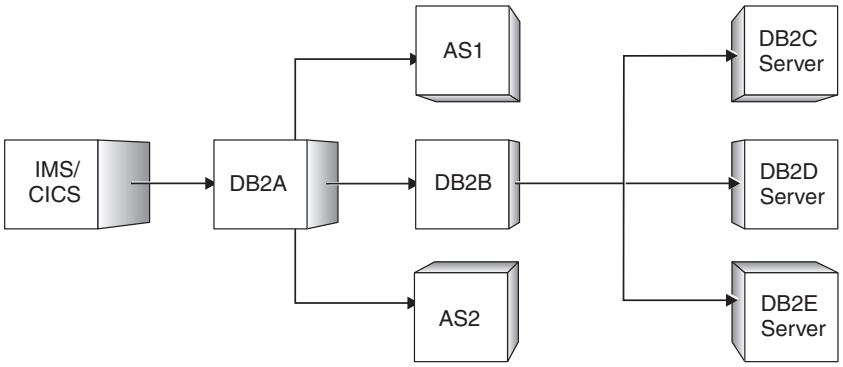


Figure 35. Illustration of multi-site unit of work

If the connection between DB2A and the coordinating IMS system fails, the connection becomes an indoubt thread. However, DB2A connections to the other systems are still waiting and are not considered indoubt. Automatic recovery occurs 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 that are involved in the unit of work.

Illustration of multi-site update

You can set up a multi-site update that involves one coordinator and two participants.

The following figure shows an example of a multi-site update with one coordinator and two participants.

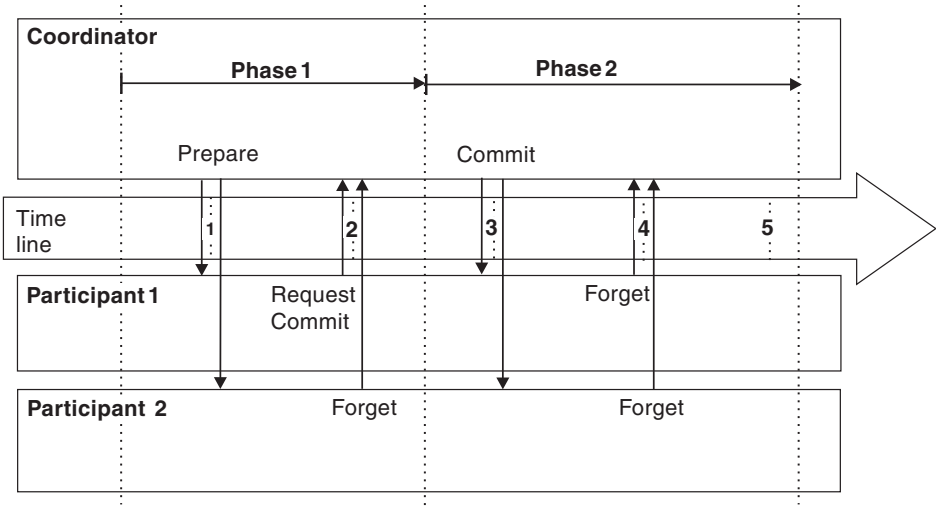


Figure 36. Illustration of multi-site update

The following process describes each action that the figure 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. The disposition (commit or rollback) of the logical unit of work is irrelevant to the participant.

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 is to 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

1. 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, no second phase occurs, and no log records are written by the coordinator.
2. 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.)
3. 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.

Termination for multiple systems

Termination for multiple systems is like termination for single systems, but with some 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 that is 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 it could be indoubt from an earlier termination that is 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.

Consistency after termination or failure

If a DB2 failure occurs while DB2 acts as a coordinator, DB2 has the information to determine whether to commit or roll back after restart. However, if a DB2 failure occurs while DB2 acts as the participant, DB2 must determine after restart whether to commit or 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, DB2 must get information from the coordinator after the connection is re-established.

The status of a unit of recovery after a termination or failure depends on the moment when the incident occurred. The following figure shows possible statuses.

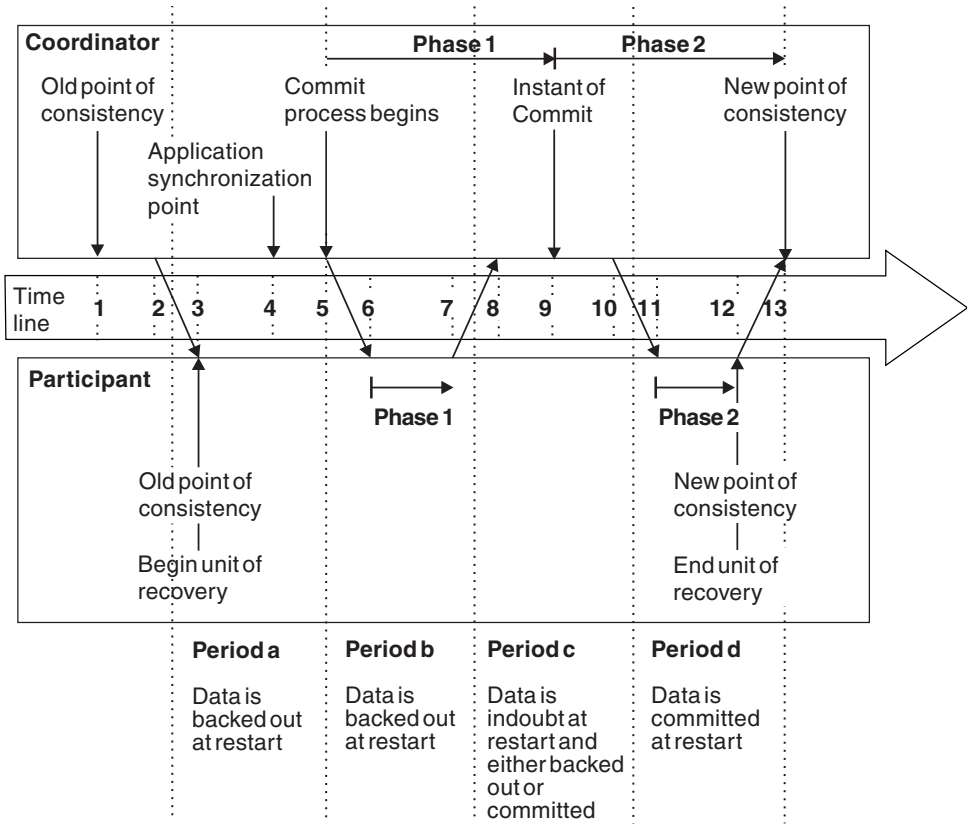


Figure 37. Time line illustrating a commit that is coordinated with another subsystem

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 the 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.

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 365.

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 "Consistency after termination or failure" on page 384).

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 is not yet committed. The data is locked and cannot be used until DB2 recognizes and acts on the indoubt decision. (For a description of indoubt units of recovery, see "Resolving indoubt units of recovery" on page 386.)

Phase 4: Backward log recovery
This phase reverses changes that are performed for inflight or in-abort units of recovery. At the end of this phase, interrupted inflight and in-abort changes are removed from the database (the data is consistent and can be used) or removal of the changes is postponed (the data is inconsistent and unavailable).

If removal of the changes is 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 at completion of backout work or at cold start 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 the RECOVER POSTPONED command is issued.

Multiple-system restart with conditions

In some circumstances, you might need to perform a multiple-system conditional restart.

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 that are 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 subsystem 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.

Related tasks:

"Resolving inconsistencies resulting from a conditional restart" on page 529

Heuristic decisions about whether to commit or abort an indoubt thread

From the perspective of DB2, 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 than 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 that is 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 that the coordinator takes. Validate your decision with the administrator of the other systems that are involved with the logical unit of work.

Resolving indoubt units of recovery

If DB2 loses its connection to another system, it attempts to recover all inconsistent objects after restart. The information that is needed to resolve indoubt units of recovery must come from the coordinating system.

About this task

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.

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.

Related information:

DSNR036I (DB2 Messages)

Resolution of IMS indoubt units of recovery

The resolution of indoubt units of recovery 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 the commit decision was recorded on the IMS log. The existence of indoubt units of recovery 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 of recovery 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 is 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 of recovery, the attachment facility issues message DSNM004I.

The IMS attachment facility writes all the records that are involved in indoubt processing to the IMS log tape as type X'5501FE'.

For all resolved units of recovery, 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. You can 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 that you can clean up the IMS RREs. You can then recover the indoubt threads.

Resolve all indoubt units of work unless software or operating problems occur, 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 of recovery are resolved.

Related tasks:

"Controlling IMS connections" on page 296

Resolution of CICS indoubt units of recovery

The resolution of indoubt units of recovery 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 of recovery, depending on whether an end-of-unit-of-work log record exists. The existence of indoubt work does not lock CICS resources until DB2 connects.

A process to resolve indoubt units of recovery is initiated during startup 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 list. 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 for 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 unresolved units of recovery from previous connections, if any exist.

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 of recovery, 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; you can then resolve them.

Related information:

"Recovering CICS indoubt units of recovery" on page 482

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 re-establish communication with each other. If something prevents this, 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, 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, DB2 knows the commit or 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, inconsistent DB2 managed data might exist.
- In a Sysplex, if DB2 is restarted on a z/OS system where RRS is not installed, 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, you can use the RRS unit of recovery ID that is displayed 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.

Resolving 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.

About this task

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:

```
DSNL405I  = THREAD
G91E1E35.GFA7.00F962CC4611.0001=217
PLACED IN INDOUBT STATE BECAUSE OF
COMMUNICATION FAILURE WITH COORDINATOR ::FFFF: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 handling 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 start DB2 and to release locks.

Procedure

GUPI

To manually resolve indoubt transactions:

1. Issue the command `-DISPLAY THREAD(*) T(I) DETAIL` to display indoubt threads from the resource manager console.

   This command produces output like this example:

```
=dis thd(*) t(i) detail
DSNV401I  = DISPLAY THREAD REPORT FOLLOWS -
DSNV406I  = INDOUBT THREADS -
COORDINATOR              STATUS      RESET URID         AUTHID
1 ::FFFF:9.30.30.53..4007   INDOUBT     0000111F049A       ADMF002
  V437-WORKSTATION=jaijeetsvl, USERID=admf002,
       APPLICATION NAME=db2jccmain
2 V440-XID=7C7146CE 00000014 00000021 F6EF6F8B
         F36873BE A37AC6BC 256F295D 04BE7EE0
         8DFEF680 D1A6EFD5 8C0E6343 67679239
         CC15A350 65BFB8EA 670CEBF4 E85938E0
         06
2 V467-HAS LUWID G91E1E35.GFA7.00F962CC4611.0001=217
  V466-THREAD HAS BEEN INDOUBT FOR    00:00:17
DISPLAY INDOUBT REPORT COMPLETE
```

Key	Description
1	Note that in this example, only one indoubt thread exists.
2	A transaction is identified by a transaction identifier, called an XID. The first 4 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 that is 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 in step 1. Use either of the following commands:
   - `-RECOVER INDOUBT ACTION(COMMIT) LUWID(217)`
   - `-RECOVER INDOUBT ACTION(ABORT) LUWID(217)`

   Either command produces output like this example:

```
=RECOVER INDOUBT ACTION(COMMIT) LUWID(217)
DSNV414I  = THREAD
LUWID=G91E1E35.GFA7.00F962CC4611.0001=217 COMMIT
SCHEDULED
DSN9022I  = DSNVRI '-RECOVER INDOUBT' NORMAL COMPLETION
```

4. Display indoubt threads again from the resource manager console by issuing the `-DISPLAY THREAD(*) T(I) DETAIL` command.

   This command produces output like this example:

```
   =dis thd(*) t(i) detail
   DSNV401I  = DISPLAY THREAD REPORT FOLLOWS -
   DSNV406I  = INDOUBT THREADS -
   COORDINATOR               STATUS      RESET URID         AUTHID
1  ::FFFF: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
```

Key	Description
1	Notice that the transaction now appears as a heuristically committed transaction (COMMITTED=H).

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 in step 1 by issuing the `-RESET INDOUBT IPADDR(::FFFF:9.30.30.53..4007)FORCE` command.

   This command produces output like this example:

```
=RESET INDOUBT IPADDR(::FFFF:9.30.30.53..4007)FORCE
DSNL455I  = DB2 HAS NO INFORMATION RELATED TO
IPADDR 9.30.30.53:4007
DSNL454I  = QUALIFYING INDOUBT INFORMATION FOR
IPADDR 9.30.30.53:4007 HAS BEEN PURGED
```

GUPI

Resolving 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. Failure also can result with the communication link between the participant and coordinator, even if the systems themselves have not failed.

About this task

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 that is required for that unit of recovery 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 releasing locked resources and bypassing the normal recovery process is imperative, 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 recovery action of the coordinator is, data is inconsistent at the coordinator and participant.

Procedure

To resolve units of recovery manually, you must use the following approaches:

- Commit changes that were made by logical units of work that were committed by the other system.
- Roll back changes that were made by logical units of work that were rolled back by the other system.

Determining the coordinator's commit or abort decision

You can use several methods to ascertain the status of indoubt units of work at other systems.

Procedure

To ascertain the status of indoubt units of work, use one of the following approaches:

- 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.
- GUPI Use the command DISPLAY THREAD TYPE(INDOUBT) LUWID(*luwid*). If the coordinator DB2 system is started and no DB2 cold start was performed, you can issue a DISPLAY THREAD TYPE(INDOUBT) command. 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. GUPI
- Read the recovery log by using DSN1LOGP.

If the coordinator DB2 cannot be started, DSN1LOGP can determine the commit decision. If the coordinator DB2 performed a cold start (or any type of conditional restart), the system log should contain messages DSNL438I or DSNL439I, which describe the status of the unit of recovery (LUWID).

Recovering indoubt threads

After you determine whether you need to commit or roll back an indoubt thread, you can recover the thread by using the RECOVER INDOUBT command.

About this task

This command does not erase the indoubt status of the thread. The status 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 token with the command.

Procedure

To recover an indoubt thread:

Issue the RECOVER INDOUBT command.
Use the ACTION(*ABORT|COMMIT*) option of the RECOVER INDOUBT command to commit or roll back a logical unit of work (LUW). If your system is the coordinator of one or more other systems that are involved with the logical unit of work, your action propagates to the other system that are associated with the LUW.

Example

GUPI

Assume that you need to recover two indoubt threads. The first has LUWID=DB2NET.LUNSITE0.A11A7D7B2057.0002, and the second has a token of 442. To commit the LUWs, enter the following command:

```
-RECOVER INDOUBT ACTION(COMMIT) LUWID(DB2NET.LUNSITE0.A11A7D7B2057.0002,442)
```

GUPI

Related concepts:

"Scenarios for resolving problems with indoubt threads" on page 584

Resetting the status of an indoubt thread

After manual recovery of an indoubt thread, allow the systems to resynchronize automatically. Automatic resynchronization resets the status of the indoubt thread. However, you might need to take additional steps if heuristic damage or a protocol error occurs.

Procedure

To delete indoubt thread information for a thread whose reset status is set to YES:

Issue the RESET INDOUBT command.
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 that is connected to IMS or CICS, the command applies only to coordinator information about downstream participants. Information that is purged does not appear in the next display thread report and is erased from the DB2 logs.

Example

GUPI

Assume that you need to purge information about two indoubt threads. The first has an LUWID=DB2NET.LUNSITE0.A11A7D7B2057.0002 and a resync port number of 123, and the second has a token of 442. Use the following command to purge the information:

```
-RESET INDOUBT LUWID(DB2NET.LUNSITE0.A11A7D7B2057.0002:123,442)
```

GUPI

You can also use a LUNAME or IP address with the RESET INDOUBT command. You can use the keyword IPADDR in place of LUNAME or LUW keywords, when the partner uses TCP/IP instead of SNA. The resync port number of the parameter 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 that are associated with that location by using the (*) option.

Resolving an indoubt unit of recovery during DB2 restart

When DB2 logs are no longer available, you can resolve one, long-running indoubt unit of recovery (UR) during DB2 restart by using the CRESTART control statement of the DSNJU003 utility.

About this task

Use this method of resolving an indoubt UR only when the DB2 logs are not available. All database updates for the indoubt UR are committed after DB2 restarts.

Procedure

To resolve an indoubt UR during DB2 restart:

Run the DSNJU003 utility and specify the CRESTART control statement with the following options:

- STARTRBA, where the value is the first log RBA that is available after the indoubt UR
- FORWARD=YES to allow forward-log recovery
- BACKOUT=YES to allow backward-log recovery

Related reference:

Syntax and options of the DSNJU003 control statement (DB2 Utilities)

Chapter 12. Backing up and recovering your data

DB2 supports recovering data to its current state or to an earlier state. You can recover table spaces, indexes, index spaces, partitions, data sets, and the entire system. Developing backup and recovery procedures at your site is critical in order to avoid costly and time-consuming loss of data.

About this task

Develop procedures for the following recovery actions:
- Create a point of consistency
- Recovering 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 a z/OS component failure
- Recover from a disaster off-site

Recommendation: To improve recovery capability in the event of a disk failure, use dual active logging, and place the copies of the active log data sets and the bootstrap data sets on different disk volumes.

Tip: For best results, check the consistency of the DB2 catalog and directory regularly, even outside of the migration process. For detailed instructions, see Migration step 2: Verify the integrity of DB2 table spaces (DB2 Installation and Migration) and Migration step 4: Check for consistency between catalog tables (optional) (DB2 Installation and Migration)

The following utilities are the principal tools for DB2 recovery:
- QUIESCE
- REPORT
- COPY
- COPYTOCOPY
- RECOVER
- MERGECOPY
- BACKUP SYSTEM
- RESTORE SYSTEM

This information provides an overview of these utilities to help you with your backup and recovery planning. Note that the term *page set* in this information can refer to a table space, index space, or any combination of these.

Related concepts:

How the initial DB2 logging environment is established

Related reference:

Implications of moving data sets after a system-level backup

© Copyright IBM Corp. 1982, 2016

Plans for recovery of distributed data

In a distributed data environment, each unit of work is processed as a whole at the target system, regardless of where a unit of work originates. Therefore, a unit of work is recovered as a whole at the target system.

At a DB2 server, the entire unit is either committed or rolled back. It is not processed if it violates referential constraints that are 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, applications should be written so that all related updates are done within one unit of work.

Point-in-time recovery (to the last image copy or to a relative byte address (RBA)) presents other challenges. 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 two different logs are involved, and a RBA does not mean the same thing for both of them.

In planning, 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.

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.

Plans for extended recovery facility toleration

DB2 can be used in an extended recovery facility (XRF) recovery environment with CICS or IMS.

All DB2-owned data sets (executable code, the DB2 catalog, and user databases) must be on a disk that is 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.

Plan carefully 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.

Plans for recovery of indexes

You can use the REBUILD INDEX utility or the RECOVER utility to recover DB2 indexes to currency.

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 or system-level backup and then applying the log. It can also recover indexes to a prior point in time.

You can use the REBUILD INDEX utility to recover any index, and you do not need to prepare image copies or system-level backups 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 by using the SQL statement ALTER INDEX with the option COPY YES before you backup and recover them using image copies or system-level backups.
- Create image copies of all indexes that you plan to recover or take system-level backups by using the BACKUP SYSTEM utility.

The COPY utility makes full image copies or concurrent copies of indexes. Incremental copies of indexes are 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.

Actions to take when you back up data

When backing up data, you can take certain actions to improve the results associated with data recovery.

During the installation of, or migration to, DB2 for z/OS DB2 10, make a full image copy of the DB2 directory and catalog with installation job DSNTIJIC .
If this task was not done during installation or migration, use the COPY utility. to make a full image copy of the DB2 catalog and directory. If this task is not done and there are later problems with inconsistent data in the DB2 catalog or directory, the RECOVER utility cannot be used to resolve the problem.

To speed recovery of the catalog and directory indexes, take a full image copy of all the index spaces when you copy the table spaces. This task enables you to recover the table spaces and index spaces at the same time.

Periodically make image copies of the catalog, directory, and user databases
This task minimizes the time the RECOVER utility requires to perform recovery. In addition, this task increases the probability that the necessary

archive log data sets are still available. Make two copies of each level of image copy data sets to reduce the risk of loss or damage.

To speed recovery of the indexes, take a full image copy of all the index spaces when you copy the table spaces. This task enables you to recover the table spaces and index spaces at the same time.

Use dual logging for active log, archive log, and bootstrap data sets (BSDSs).

This task increases the chance of recovering from all problems, and is especially useful for data inconsistency problems.

Before you use RECOVER, rename the data sets

If the image copy or log data sets are damaged, use the RECOVER utility to compound the problem. Therefore, before you use RECOVER copy the data sets by using DSN1COPY.

The RECOVER utility applies log records to the new data set with the old name. Then, if a problem occurs during RECOVER utility processing, there is a copy (under a different name) of the data set being recovered.

Keep back-level image copy data sets

If an image copy of a table space or index space that contains inconsistent data is made, the RECOVER utility cannot resolve the data inconsistency problem. However, RECOVER can resolve the inconsistency if there is an older image copy of that table space or index space that is taken before the problem occurred. The MODIFY utility deletes the SYSCOPY record that describes an image copy. To retain a particular image copy for use by RECOVER, use the MODIFY utility with the TIMESTAMP of that image copy. This task also retains any later image copies.

Maintain consistent referential structures

A *referential structure* is a set of tables and relationships that are designed with referential constraints, such that each table in the set is a parent or dependent of itself or another table in the set. To facilitate maintaining 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. REPORT TABLESPACESET reports all members of a table space set defined by referential constraints.

A referential structure must be kept consistent regarding 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.

Related reference:

COPY (DB2 Utilities)

Actions to avoid when you back up data

When backing up data, avoid certain actions to reduce problems that occur during data recovery.

Do not discard archive logs that might be needed

The RECOVER utility might need an archive log to recover from an inconsistent data problem. If discarded, the problem must be resolved manually without using the RECOVER utility.

Do not make an image copy of a table space or index space that contains inconsistent data

If the COPY utility is used to make an image copy of a table space or index space that contains inconsistent data, the RECOVER utility cannot recover a problem that involves that space (unless an older image copy is available that was taken before the problem occurred). Run DSN1COPY with the CHECK option to determine whether intra-page data inconsistency problems exist on table or index spaces before you make image copies of them. If a COPY of a catalog or directory table space is taken, run DSN1COPY CHECK (before image copies are taken) to verify the integrity of DB2 catalog and directory table spaces.

Do not use the -TERM UTILITY command on utility jobs you want to restart

If an error occurs while a utility is being run, the data on which the utility was operating remains at a commit point. If the utility stops while it has exclusive access to data, other applications cannot access that data. In this case, you can 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 the utility job cannot be restarted or does not need to be restarted.

Issuing the -TERM UTILITY command can cause two different situations:
- If the utility is active, it terminates at its next commit point.
- If the utility is stopped, it terminates immediately.

By using the -TERM UTILITY command to terminate the REORG or LOAD utilities, or COPY utility for incremental image copies, 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.

Related tasks:

Discarding archive log records

Related reference:

-TERM UTILITY (DB2) (DB2 Commands)

COPY (DB2 Utilities)

Preparation for recovery: a scenario

You can use the RECOVER utility to recover table spaces or index spaces.

DB2 can recover a page set by using an image copy or system-level backup, the recovery log, or both. The DB2 recovery log contains a record of all changes that are made to the page set. If DB2 fails, it can recover the page set by restoring the image copy or system-level backup and applying the log changes to it from the point of the image copy or system-level backup.

The DB2 catalog and directory page sets must be copied at least as frequently as the most critical user page sets. Moreover, you are responsible for periodically copying the tables in the communications database (CDB), the application registration table, the object registration table, and the resource limit facility (governor), or for maintaining the information that is necessary to re-create them. Plan your backup strategy accordingly.

The following backup scenario suggests how DB2 utilities might be used when taking object level backups with the COPY utility:

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.

Tuesday morning

You run the COPY utility again. This time you make an incremental image copy to record only the changes that have been 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. You run the RECOVER utility. The utility restores the table space from the full image copy that was made by MERGECOPY on Wednesday and the incremental image copies that were made on Thursday and Friday, and includes all changes that are 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 the current point in time.

This 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.

Related reference:

COPY (DB2 Utilities)

RECOVER (DB2 Utilities)

Events that occur during recovery

During recovery, several events occur, such as reading the log and running utilities that rely on image copies of the data.

Figure 39 on page 402 shows an overview of the recovery process. To recover a page set, the RECOVER utility typically uses these items:

- A backup that is a full image copy or a system-level backup.
- For table spaces only, when the COPY utility is used, any later incremental image copies; each incremental image copy summarizes all the changes that were made to the table space from the time that the previous image copy was made.
- All log records for the page set that were created since the image copy. 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 that are to be applied by the RECOVER utility.

In the example shown in Figure 39 on page 402:

- Where the COPY utility is used, the RECOVER utility uses information in the SYSIBM.SYSCOPY catalog table to:
  - Restore the page set with the data in the full image copy (appearing, in Figure 39 on page 402, at X'38000').
  - For table spaces only, apply all the changes that are summarized in any later incremental image copies. (Two copies appear in Figure 39 on page 402: 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 39 on page 402, X'C0040', that address is also stored in the SYSIBM.SYSCOPY catalog table.)

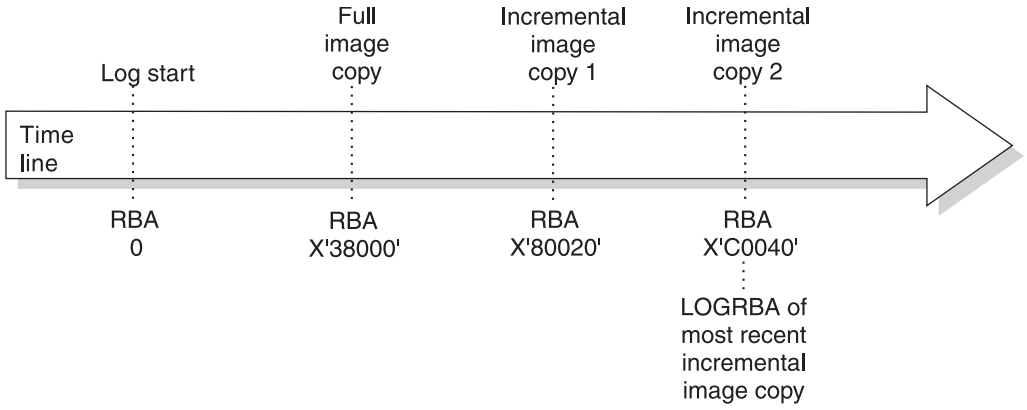


Figure 38. Overview of DB2 recovery from COPY utility. The figure shows one complete cycle of image copies. The SYSIBM.SYSCOPY catalog table can record many complete cycles.

- Where the BACKUP SYSTEM utility is used, the RECOVER utility uses information in the BSDS and in the SYSIBM.SYSCOPY catalog table to:
  - Restore the page set from the most recent backup, in this case, it is a system-level backup (appearing, in Figure 39 on page 402 at X'80000').
  - Apply all changes to the page set that are registered in the log, beginning at the log RBA of the most recent system-level backup.

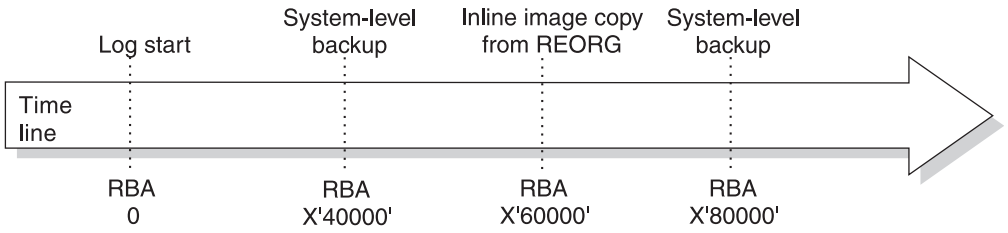


Figure 39. Overview of DB2 recovery from BACKUP SYSTEM utility. The figure shows one complete cycle of image copies. The SYSIBM.SYSCOPY catalog table can record many complete cycles.

Complete recovery cycles

In planning for recovery, you determine how often to take image copies or system-level backups and how many complete cycles to keep. Those values tell how long you must keep log records and image copies or system-level backups for database recovery.

In deciding how often to take image copies or system-level backups, consider the time needed to recover a table space. The time is determined by all of the following factors:

- The amount of log to traverse
- The time that it takes an operator to mount and remove archive tape volumes
- The time that it takes to read the part of the log that is needed for recovery
- The time that is needed to reprocess changed pages

In general, the more often you make image copies or system-level backups, the less time recovery takes, but the more time is spent making copies. If you use LOG NO without the COPYDDN keyword or the FCCOPYDDN 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 or the FCCOPYDDN 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, 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 arise unexpectedly. The MERGECOPY operation does not apply to indexes.

Even if the BACKUP SYSTEM is used, it is important to take full image copies (inline copies) during REORG and LOAD to avoid the COPY-pending status on the table space.

Recommendation: Copy your indexes after the associated utility has run. Optionally, you can take a FlashCopy image copy when running the REBUILD INDEX or REORG INDEX utilities. Indexes are placed in informational COPY-pending (ICOPY) status after running the LOAD TABLESPACE, REORG TABLESPACE, REBUILD INDEX, or REORG INDEX utilities. Only structural modifications of the index are logged when these utilities are run, so the log does not have enough information 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 that contains an important image copy or a log data set. A retention period of at least two full cycles is recommended. For enhanced security, keep records for three or more copy cycles.

A recovery cycle example when using image copies

Log management for a user group involves using image copies. You need to determine how often to make image copies.

Table 33 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, which contains employee salary data. If the group chooses to keep two complete image copy cycles on hand, each time EMPSALS is copied, records prior to its previous copy or copies, made two months ago, can be deleted. They will always have on hand between two months and four months of log records.

In the example, the most critical tables are copied daily. The DB2 catalog and directory are also copied daily.

Table 33. 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 are using the BACKUP SYSTEM utility, you should schedule the frequency of system-level backups based on your most critical data.

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 "Plans for recovery of indexes" on page 397 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 DFSMShsm, you need to know that it automatically moves data to and from the DB2 databases.

Restriction: Because DFSMShsm can migrate data sets to different disk volumes, you cannot use DFSMShsm in conjunction with the BACKUP SYSTEM utility. The RECOVER utility requires that the data sets reside on the volumes where they had been at the time of the system-level backup.

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 they allow for keeping a data set for a predetermined period before deleting or moving it.

All DFSMShsm operations can also be performed manually.

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 be recalled only to like devices of the same VOLSER as 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 that is specified by the RECALL DELAY parameter. If the recall is not completed within that time, the program receives an error message indicating that 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.

Related tasks:

"Discarding archive log records" on page 357

Related information:

DFSMShsm Managing Your Own Data

Tips for maximizing data availability during backup and recovery

You need to develop a plan for backup and recovery. Then, 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.

Consider the following factors when you develop and implement your plan:

- "Decide on the level of availability you need"
- "Practice for recovery"
- "Minimize preventable outages"
- "Determine the required backup frequency"
- "Minimize the elapsed time of RECOVER jobs" on page 406
- "Minimize the elapsed time for COPY jobs" on page 406
- "Determine the right characteristics for your logs" on page 406
- "Minimize DB2 restart time" on page 407

Decide on the level of availability you need

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.

The availability of data is affected by the availability of related objects. For example, if there is an availability issue with one object in a related set, the availability of the others may be impacted as well. The related object set includes base table spaces and indexes, objects related by referential constraints, LOB table space and indexes, and XML table spaces and indexes.

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 you use image copies and 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, you should make copies after every 4 GB of data that is 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 take system-level backups using the BACKUP SYSTEM utility. Because the FlashCopy technology is used, the entire system is backed up very quickly with virtually no data unavailability.

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

When recovering system-level backups from disk, the RECOVER utility restores data sets serially by the main task. When recovering system-level backups from tape, the RECOVER utility creates multiple subtasks to restore the image copies and system-level backups for the objects.

If you are using system-level backups, be sure to have recent system-level backups on disk to reduce the recovery time.

Restoring FlashCopy image copies happens very quickly. However, creating a FlashCopy image copy with consistency (FLASHCOPY CONSISTENT) uses more system resources and might take longer than creating an image copy by specifying FLASHCOPY YES. Specifying FLASHCOPY CONSISTENT might take longer, because backing out uncommitted work requires reading the logs and updating the image copy.

For point-in-time recoveries, recovering to quiesce points and SHRLEVEL REFERENCE copies can be faster than recovering to other points in time.

If you are recovering to a non-quiesce point, the following factors can have an impact on performance:

- The duration of URs that were active at the point of recovery.
- The number of DB2 members that have active URs to be rolled back.

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.

Also, you can take FlashCopy image copies with the COPY utility. FlashCopy can reduce both the unavailability of data during the copy operation and the amount of time that is required for recovery operations.

Determine the right characteristics for your logs

Consider the following criteria when determining 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 that you have enough tape drives so 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.

You can limit the backout activity during DB2 system restart. You can postpone the backout of long-running units of recovery until after the DB2 system is operational. Use the installation options LIMIT BACKOUT and BACKOUT DURATION to determine what backout work will be delayed during restart processing.

The following major factors 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 IBM Software Support 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.

  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.
- Active threads

  DB2 cannot shut down until all threads have terminated. Issue the DB2 command DISPLAY THREAD to determine if any threads are active 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 that were opened since DB2 startup. You can reduce the time that this processing takes by setting the z/OS parameter DDCONS(NO).

The following major factors influence the speed of DB2 startup:

- DB2 checkpoint interval

  The DB2 checkpoint interval indicates the 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 good startup performance in most cases.

  You can use the LOGLOAD option, the CHKTIME option, or a combination of both of these options of the SET LOG command to modify the CHKFREQ value dynamically without recycling DB2. The value that you specify depends on your restart requirements. The default value for the CHKTIME option is 5 minutes.
- Long-running units of work

  DB2 rolls back uncommitted work during startup. The amount of time for this activity is approximately 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 takes 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 need to wait for tape mounts during startup.

Related concepts:

FlashCopy image copies (DB2 Utilities)

Related tasks:

"Dynamically changing the checkpoint frequency" on page 350

"Deferring restart processing" on page 372

Managing the opening and closing of data sets (DB2 Performance)

Setting the size of active log data sets (DB2 Performance)

Related reference:

-SET LOG (DB2) (DB2 Commands)

Where to find recovery information

Information that is needed for recovery is contained in several locations.

SYSIBM.SYSCOPY

SYSIBM.SYSCOPY is a catalog table that 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 that is recorded by the COPYTOCOPY utility.

SYSCOPY also contains entries with the same kinds of log RBAs that are recorded by the utilities QUIESCE, REORG, LOAD, REBUILD INDEX, RECOVER TOCOPY, and RECOVER TOLOGPOINT.

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 therefore, 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 regardless of whether the LOG option is YES or NO, or whether the UNLOAD PAUSE option is specified.

When DSNDB01.DBD01, DSNDB01.SYSUTILX, DSNDB06.SYSCOPY, and DSNDB01.SYSDBDXA 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
SYSIBM.SYSLGRNX is a directory table that contains records of the log RBA ranges that are 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.

BSDS (bootstrap data set)
The BSDS contains information about system-level backups, and the DB2 archive log data sets. The RECOVER utility uses the recovery base log point RBA or LRSN value associated with the system-level backup to look for the log information after restoring the object from the system-level backup.

In a data-sharing environment, each DB2 member keeps track of the system-level backups taken on that particular member in its BSDS.

DFSMShsm
The RECOVER utility queries DFSMShsm for information about whether a system-level backup resides on disk or tape.

Related reference:

DB2 catalog tables (DB2 SQL)

How to report recovery information

You can use the REPORT utility when you plan for recovery.

The REPORT utility provides information necessary for recovering a page set. The REPORT utility 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
- Recovery information about system-level backup copies retrieved from the bootstrap data sets of each member in the data sharing group

You can also use the REPORT utility to obtain recovery information about the catalog and directory.

Use the output from the DFSMShsm LIST COPYPOOL command with the ALLVOLS option, in conjunction with the DB2 system-level backup information in the PRINT LOG MAP (DSNJU004) utility output, to determine whether the system-level backups of your database copy pool still reside on DASD or if they have been dumped to tape. For a data sharing system, you should run the print log map utility with the MEMBER * option to obtain system-level backup information from all members.

Related concepts:

REPORT output (DB2 Utilities)

Related reference:

LIST command (z/OS DFSMShsm Storage Administration Reference)

REPORT (DB2 Utilities)

Discarding SYSCOPY and SYSLGRNX records

Use the MODIFY utility to delete obsolete records from SYSIBM.SYSCOPY and SYSIBM.SYSLGRNX.

About this task

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.

Procedure

To discard SYSCOPY and SYSLGRNX records:

1. Complete the first three steps of the procedure that is presented in "Locating archive log data sets" on page 357. In the third step, note the date of the earliest image copy that you intend to keep.

   Important: 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 RECOVERY utility to clean up obsolete entries in SYSIBM.SYSCOPY and SYSIBM.SYSLGRNX.

   Pick one of the following MODIFY strategies, based on keyword options, that is consistent with your overall backup and recovery plan:

DELETE DATE
With the DELETE DATE keyword, you can specify date of the earliest entry you want to keep.

DELETE AGE
With the DELETE AGE keyword, you can specify the age of the earliest entry you want to keep.

RETAIN LAST
With the RETAIN LAST keyword, you can specify the number of image copy entries you want to keep.

RETAIN GDGLIMIT
If GDG data sets are used for your image copies, you can specify RETAIN GDGLIMIT keyword to keep the number of image copies matching your GDG definition.

RETAIN LOGLIMIT
With the RETAIN LOGLIMIT keyword, you can clean up all of the obsolete entries that are older than the oldest archive log in your BOOT STRAP DATA SET (BSDS).

For example, you could enter one of the following commands:

The DELETE DATE option removes records that were written earlier than the given date. You can also specify the DELETE AGE option to remove records that are older than a specified number of days or the DELETE RETAIN option to specify a minimum number of image copies to keep.

```
MODIFY RECOVERY TABLESPACE dbname.tsname
   DELETE DATE date
```

```
MODIFY RECOVERY TABLESPACE dbname.tsname
    RETAIN LAST( n )
```

The RETAIN LAST(*n*) option keeps the *n* recent records and removes the older one.

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.

The MODIFY utility discards SYSLGRNX records that meet the deletion criteria when the AGE or DATE options are specified, even if no SYSCOPY records were deleted.

You cannot run the MODIFY utility on a table space that is in RECOVER-pending status.

Even if you take system-level backups, use the MODIFY utility to delete obsolete records from SYSIBM.SYSCOPY and SYSIBM.SYSLGRNX. You do not need to delete the system-level backup information in the bootstrap data set (BSDS) because only the last 50 system-level backups are kept.

Preparations for disaster recovery

If a DB2 computing center is totally lost, you can recover on another DB2 subsystem at a recovery site. To do this, you must regularly back up the data sets and the log for the recovery subsystem. As with all data recovery operations, the objectives of disaster recovery are to minimize the loss of data, workload processing (updates), and time.

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.

For data sharing environments, you can use the LIGHT(YES) or LIGHT(NOINDOUBTS) parameter to quickly bring up a DB2 member to recover retained locks. This option is not recommended for refreshing a single subsystem and is intended only for a cross-system restart for a system that has inadequate capacity to sustain the DB2 IRLM pair. Restart light can be used for normal restart and recovery.

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.

To perform a light start at the remote site:

1. Start the members that run permanently with the LIGHT(NO) option. This is the default.
2. Start other members in light mode. The members started in light mode use a smaller storage footprint. After their restart processing completes, they automatically shut down. If ARM is in use, ARM does not automatically restart the members in light mode 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), manually send it to the recovery site, and 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.
- Use the log capture exit routine to capture log data in real time and send it to the recovery site.

Related concepts:

Log capture routines

Restart light (DB2 Data Sharing Planning and Administration)

Related tasks:

Reading log records with the log capture exit routine

Related reference:

DSNTIPL: Active log data set parameters (DB2 Installation and Migration)

Related information:

Performing remote-site disaster recovery

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.

The following figure is an overview of the process of preparing to start DB2 at a recovery site.

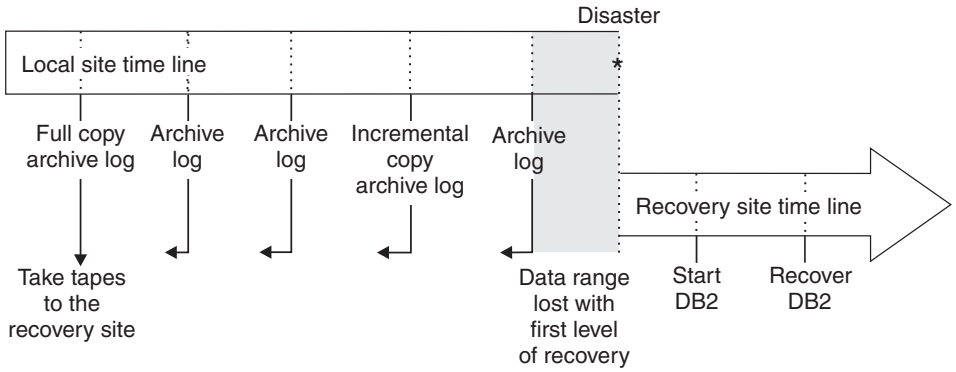


Figure 40. Preparing for disaster recovery. The information that you need to recover is contained in the copies of data (including the DB2 catalog and directory) and the archive log data sets.

Recommendations for more effective recovery from inconsistency

Data inconsistency problems can often be resolved by using the RECOVER utility. However, if the RECOVER utility requires data from the recovery log or image copy data sets, and that data is damaged or unavailable, you might need to resolve the problem manually.

Actions to take to aid in successful recovery of inconsistent data

You need to take certain steps to prepare for the successful recovery of inconsistent data when you set up the database.

- During the installation of, or migration to, DB2 10, make a full image copy of the DB2 directory and catalog by using installation job DSNTIJIC.

  If you did not take this copy during installation or migration, use the COPY utility to make a full image copy of the DB2 catalog and directory. If you do not take such a copy and later find inconsistent data in the DB2 catalog or directory, you cannot use the RECOVER utility to resolve the problem.

  This action is recommended even if you take system-level backups.
- Periodically make an image copy of the catalog, directory, and user databases.

  These copies minimize the time that the RECOVER utility requires to recover the data. This action also increases the probability that the necessary archive log

data sets are available when you need them. Keep two copies of each level of image copy data set. Having a second copy reduces the risk in case one image copy data set is lost or damaged.

This action is recommended even if you take system-level backups.

- Use dual logging for your active log, archive log, and bootstrap data sets.

  Dual logging increases the probability that you can recover from unexpected problems. Dual logging is especially useful in resolving data inconsistency problems.
- Before you use RECOVER, rename your data sets.

  Restriction: Do not rename your data sets if you take system-level backups.

  If the image copy or log data sets are damaged, you can compound the problem by using the RECOVER utility. Therefore, before you use RECOVER, rename your data sets by using one of the following methods:
  - Rename the data sets that contain the page sets that you want to recover.
  - Copy your data sets by using DSN1COPY.
  - 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 have a copy (under a different name) of the data set that you want to recover.
- Keep earlier copies of your data.

  If you make an image copy or system-level backup of a page set that contains inconsistent data, RECOVER 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 or system-level backup that was taken before the problem occurred. You can also resolve the inconsistency problem by using the RESTOREBEFORE recovery option 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 includes at least one table, and for every table in the set, all of the relationships in which the table participates and 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. Also, 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 that are defined by referential constraints.

  A referential structure must be kept consistent for point-in-time recovery. Use the QUIESCE utility to establish a point of consistency for a table space set. Later the table space set can be recovered to this point of consistency without introducing referential constraint violations.

  A base table space must be kept consistent with its associated LOB or XML table spaces for point-in-time recovery. Use the TABLESPACESET option of the REPORT utility to identify related objects. Related objects can include referentially related objects and auxiliary LOB or XML table spaces and their indexes. Run CHECK INDEX to validate the consistency of indexes with their associated table data. Run CHECK DATA to validate the consistency of base table space data with LOB, XML, and referentially related table spaces. If LOB columns exist, run CHECK LOB on any related LOB table spaces to validate the integrity of each LOB table space within itself.

Related concepts:

"How the initial DB2 logging environment is established" on page 339

Related tasks:

Installation step 21: Back up the DB2 directory and catalog: DSNTIJIC (DB2 Installation and Migration)

Related reference:

COPY (DB2 Utilities)

RECOVER (DB2 Utilities)

DSN1COPY (DB2 Utilities)

Related information:

Using Access Method Services (DFSMS Access Method Services for Catalogs)

Actions to avoid in recovery of inconsistent data

To minimize problems when recovering inconsistent data, you can avoid doing certain actions.

- Do not discard archive logs that 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.

- 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 that contains inconsistent data, the RECOVER utility cannot recover a problem that involves that page set unless you have an older image copy of that page set that was 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. You can also specify the CHECKPAGE parameter on the COPY utility which will cause the image copy to fail if an inconsistent page is detected. 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).

- Do not use the TERM UTILITY command on utility jobs that 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.

Related tasks:

"Discarding archive log records" on page 357

Related reference:

-TERM UTILITY (DB2) (DB2 Commands)

CHECK DATA (DB2 Utilities)

COPY (DB2 Utilities)

How to recover multiple objects in parallel

To recover multiple objects in parallel, you can either use the RECOVER utility with the PARALLEL keyword or schedule concurrent RECOVER jobs.

You can 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 table space level and not on partition or data set 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 data set for every DSNUM specification. This means that for a table space level copy on tape, the tape is always read from the beginning. It is recommended for image copies to tape, that a partitioned table space 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 if 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.**. You can simplify this specification by using LISTDEF with PARTLEVEL if all parts must be recovered.

You can 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 that is controlled by DFSMShsm is first recalled. It then resides on disk and is read in parallel.

Related reference:

RECOVER (DB2 Utilities)

Related information:

DSNU512I (DB2 Messages)

Recovery of page sets and data sets

You can recover objects in several ways, if the objects have the LOGGED attribute.

- If you made backup copies of table spaces by using the COPY utility, the COPYTOCOPY utility, or the inline copy feature of the LOAD or REORG utility, use the RECOVER utility to recover the table spaces to the current or a previous state.
- If you made backup copies of indexes by using the DB2 COPY utility, use the RECOVER utility to recover the indexes to the current or a previous state. Backing up indexes is optional.
- If you made system-level backups using the BACKUP SYSTEM utility, use the RECOVER utility to recover the objects to the current or a previous state.
- If you have z/OS Version 1.8 and have made backup copies using a method outside of DB2 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 want to restore the objects to currency, run the RECOVER utility with the LOGONLY option.

The RECOVER utility performs these actions:

- Restores the most current backup, which can be either a full image copy, concurrent copy, or system-level backup.
- 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 or a system-level backup and log records. It can also restore the object to a prior state, which is one of the following points in time:

- A specified point on the log (use the TOLOGPOINT or TORBA keyword)
- A particular image copy (use the TOCOPY, TOLASTCOPY, or TOLASTFULLCOPY keywords)

You can use the RECOVER utility with the BACKOUT YES keyword to recover data to a prior state by backing out committed work from the current state of an object.

With z/OS Version 1.8, the RECOVER utility can use system-level backups for object level recovery. The RECOVER utility chooses the most recent backup of either an image copy, concurrent copy, or system-level backup to restore. The most recent backup determination is based on the point of recovery (current or prior point in time) for the table spaces or indexes (with the COPY YES attribute) being recovered.

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.

Not every recovery operation requires RECOVER; see also:
"Recovering error ranges for a work file table space" on page 446
"Recovery of the work file database"
"Recovery of data to a prior point in time" on page 424

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 on page 443 under "Preparing to recover to a prior point of consistency" on page 441.

Related reference:

"Implications of moving data sets after a system-level backup" on page 435

REBUILD INDEX (DB2 Utilities)

RECOVER (DB2 Utilities)

Recovery of the work file database

The work file database (called DSNDB07, except in a data sharing environment) is used for temporary space for certain SQL operations, such as join and ORDER BY.

If DSNDB01.DBD01 or DSNDB01.SYSDBDXA is stopped or otherwise inaccessible when DB2 is started, the descriptor for the work file database is not loaded into main storage. Thus, the work file database is not allocated. To recover from this condition after DSNDB01.DBD01 and DSNDB01.SYSDBDXA are available, stop and then start the work file database again.

You cannot use RECOVER with the work file database.

Page set and data set copies

You can use the COPY utility to copy data from a page set to a z/OS sequential data set on disk or tape. The COPY utility makes a full or incremental image copy, depending on what you specify. You also can use the COPY utility to make copies that can be used for local or off-site recovery operations.

In addition, you can use the COPY utility to create a FlashCopy image copy in VSAM format from a page set. Fast replication makes the copy process virtually instantaneous. FlashCopy image copies are always full copies.

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.

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.

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 that you specify. DB2 determines whether to make a full or incremental image copy based on the values specified for the CHANGELIMIT option.

- If the percent of changed pages is greater than the low CHANGELIMIT value and less than the high CHANGELIMIT value, then DB2 makes an incremental image copy.
- If the percentage of changed pages is greater than or equal to the high CHANGELIMIT value, then DB2 makes a full image copy.
- If the ANY keyword is used with the CHANGELIMIT option is used, then DB2 makes a full image copy.

The CHANGELIMIT option does not apply to indexes.

If you want DB2 to recommend what image copies should be made but not to 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. For partitioned objects, if you only want the partitions in COPY-pending status or informational COPY-pending status to be copied, then you must specify a list of partitions. You can do this by invoking COPY on a LISTDEF list built with the PARTLEVEL option. An output image copy data set is created for each partition that is in COPY-pending or informational COPY-pending status.

If you want to copy objects that are in copy pending (COPY) or informational copy pending (ICOPY), you can use the SCOPE PENDING option of the COPY utility. If you specify the parameter DSNUM(ALL) with SCOPE PENDING for partitioned objects, and if one or more of the partitions are in COPY or ICOPY, the copy will be taken of the entire table or index space.

You can add conditional code to your jobs so that an incremental, 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; no CHANGELIMIT value is met. No image copy is recommended or taken.
2	Successful; the percentage 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; the percentage 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, you can take the following steps to prevent an empty image copy output data set from being created if no pages have been changed. You can perform either of the following actions:

- 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 want only image copy information to be 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.
- 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 use the COPY utility with the CHANGELIMIT option to determine whether any space map pages are broken. You can also use the COPY utility 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.

In addition, you can make a FlashCopy image copy when you run the LOAD, REORG TABLESPACE, REORG INDEX, or REBUILD INDEX utilities.

Related concepts:

"Plans for recovery of indexes" on page 397

FlashCopy image copies (DB2 Utilities)

Related reference:

COPY (DB2 Utilities)

MERGECOPY (DB2 Utilities)

Creating FlashCopy image copies

You can configure certain utilities to create image copies by using the FlashCopy function that is provided by z/OS DFSMS and the IBM TotalStorage ESS storage subsystems. FlashCopy can reduce both the unavailability of data during the copy operation and the amount of time that is required for recovery operations.

About this task

The following DB2 utilities support the creation of FlashCopy image copies:

- COPY
- LOAD
- REBUILD INDEX
- REORG INDEX
- REORG TABLESPACE

FlashCopy image copies are output to VSAM data sets. The following utilities accept the VSAM data sets that are produced by FlashCopy as input:

- COPYTOCOPY

- DSN1COMP
- DSN1COPY
- DSN1PRNT
- RECOVER

Procedure

To create a FlashCopy image copy:

Specify the FlashCopy option by using DB2 subsystem parameters, utility control statement parameters, or both.
You can use the FlashCopy subsystem parameters to define the FlashCopy option as the default behavior for each of the utilities that support the FlashCopy option. When the FlashCopy subsystem parameters are enabled as the default behavior, you do not need to specify the FlashCopy options in the utility control statement. If you specify the FlashCopy options in both the subsystem parameters and the utility control statement parameters, the specifications in the utility control statement override the specifications of the subsystem parameters.
When creating a FlashCopy image copy, the COPY and LOAD utilities can include additional phases of execution, depending on the options that are specified in the utility control statement.
When creating a FlashCopy image copy, the following utilities also can create one to four additional sequential format image copies in a single execution:

- COPY
- LOAD with the REPLACE option specified
- REORG TABLESPACE

Recommendation: To provide a recovery base for media failures, create one or more additional sequential format image copies when you create a FlashCopy image copy.

Related concepts:

FlashCopy image copies (DB2 Utilities)

Related reference:

COPY (DB2 Utilities)

LOAD (DB2 Utilities)

REBUILD INDEX (DB2 Utilities)

REORG INDEX (DB2 Utilities)

REORG TABLESPACE (DB2 Utilities)

How to make concurrent copies using DFSMS

The concurrent copy function of the Data Facility Storage Management Subsystem (DFSMS) can copy a data set at the same time that other processes access the data. Using DFSMS does not significantly impact application performance.

The two ways to use the concurrent copy function of DFSMS are:

- 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 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

You can use the IBM RAMAC Virtual Array (RVA) storage control with the peer-to-peer remote copy (PPRC) function or Enterprise Storage Server® to recover DB2 subsystems at a remote site in the event of a disaster at the local site. Both of these methods provide a faster way to recover subsystems.

About this task

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.

Using RVA SnapShot or Enterprise Storage Server FlashCopy for a DB2 backup requires a method of suspending all update activity for a DB2 subsystem. You suspend update activity to allow a remote copy of the entire subsystem to be made 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 is created, use the RESUME option on the SET LOG command to return to normal logging activities.

Related reference:

-SET LOG (DB2) (DB2 Commands)

Related information:

RAMAC Virtual Array: Implementing Peer-to-Peer Remote Copy (IBM Redbooks)

RAMAC Virtual Array (IBM Redbooks)

IBM TotalStorage Enterprise Storage Server Introduction and Planning Guide

System-level backups for object-level recoveries

If a system-level backup is chosen as a recovery base for an object, DFSMShsm is invoked by the RECOVER utility. This process restores the data sets for the object from the system-level backup of the database copy pool.

You need to set subsystem parameter SYSTEM_LEVEL_BACKUPS to YES so that the RECOVER utility will consider system-level backups.

- If the system-level backup resides on DASD, it is used for the restore of the object.
- If the system-level backup no longer resides on DASD, and has been dumped to tape, then the dumped copy is used for the restore of the object if FROMDUMP was specified.

Message DSNU1520I is issued to indicate that a system-level backup was used as the recovery base.

If DFSMShsm cannot restore the data sets, message DSNU1522I with reason code 8 is issued. If OPTIONS EVENT(ITEMERROR,SKIP) was specified, the object is skipped and the recovery proceeds on the rest of the objects. Otherwise, the RECOVER utility terminates.

If you specify YES for the RESTORE/RECOVER FROM DUMP option on installation panel DSNTIP6 or if you specify the FROMDUMP option on the RECOVER utility statement, then only dumps on tape of the database copy pool are used for the restore of the data sets. In addition, if you specify a dump class name on installation panel DSNTIP6 or if you specify the DUMPCLASS option on the RECOVER utility statement, then the data sets will be restored from the system-level backup that was dumped to that particular DFSMShsm dump class. If you do not specify a dump class name on installation panel DSNTIP6 or on the RECOVER utility statement, then the RESTORE SYSTEM utility issues the DFSMShsm LIST COPYPOOL command and uses the first dump class listed in the output.

Use the output from LISTCOPY POOL and PRINT LOG MAP to see the system-level backup information.

Use the output from the REPORT RECOVERY utility to determine whether the objects to be recovered have image copies, concurrent copies, or a utility LOG YES event that can be used as a recovery base. You also can determine if these objects are contained in a system-level backup.

You can take system-level backups using the BACKUP SYSTEM utility. However, if any of the following utilities were run since the system-level backup that was chosen as the recovery base, then the use of the system-level backup is prohibited for object level recoveries to a prior point in time:

- REORG TABLESPACE
- REORG INDEX
- REBUILD INDEX
- LOAD REPLACE
- RECOVER from image copy or concurrent copy

In the following illustration, RECOVER TOLOGPOINT receives message DSNU1528I with return code 8, indicating that the recovery has failed.

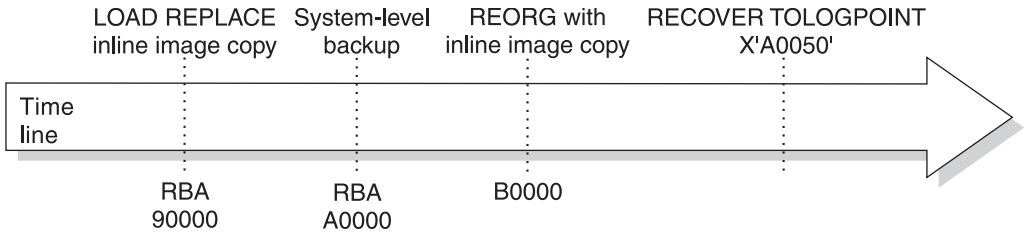


Figure 41. Failed recovery

In this example, use RECOVER with RESTOREBEFORE X'A0000' to use inline image recovery copy taken at X'90000' as a recovery base.

Related concepts:

"How to report recovery information" on page 409

Dump Tasks (z/OS DFSMShsm Storage Administration)

Recovery of data to a prior point in time

You can restore data to the state at which it existed at a prior point in time.

To restore data to a prior point in time, use the methods that are described in the following topics:

- "Options for restoring data to a prior point in time" on page 428
- "Restoring data by using DSN1COPY" on page 454
- "Backing up and restoring data with non-DB2 dump and restore" on page 455

The following terms apply to the subject of recovery to a prior point in time:

Term	Meaning
DBID	Database identifier
OBID	Data object identifier
PSID	Table space identifier

Plans for point-in-time recovery

In some circumstances, you cannot recover to the current point in time. Plan for this possibility by establishing a consistent point in time to which to recover if these circumstances occur.

TOCOPY is a viable alternative in many situations in which recovery to the current point in time is not possible or is not desirable. 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.

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.

You can use the QUIESCE utility to establish an RBA or LRSN to recover to. The RBA or LRSN can be used in point-in-time recovery.

For partitioned table spaces, if the data was redistributed across partitions by running REORG (either REORG REBALANCE or REORG to materialize limit key changes), additional recovery implications apply:

- If you recover to a point in time and use an image copy or system level-backup that was taken before the redistributing REORG, the affected partitions are placed in REORG-pending (REORP) status.
- If you recover to a point in time after you used ALTER to modify the limit keys but before running REORG to redistribute the data, the affected partitions are also placed in REORP status.

If you use the REORG TABLESPACE utility with the SHRLEVEL REFERENCE or SHRLEVEL CHANGE 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.

You can take system-level backups using the BACKUP SYSTEM utility.

Recommendation: 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.

Related concepts:

Point-in-time recovery (DB2 Utilities)

"Point-in-time recovery using the RECOVER utility" on page 427

Related reference:

RECOVER (DB2 Utilities)

Point-in-time recovery with system-level backups

System-level backups are fast replication backups that are created by using the BACKUP SYSTEM utility.

The BACKUP SYSTEM utility invokes z/OS Version 1 Release 5 or later DFSMShsm services to take volume copies of the data in a sharing DB2 system. All DB2 data sets that are to be copied (and then recovered) must be managed by SMS.

The BACKUP SYSTEM utility requires z/OS Version 1 Release 5 or later 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.

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, depending on your recovery needs. Choose FULL if you want to backup both your DB2 data and your DB2 logs.

Because the BACKUP SYSTEM utility does not quiesce transactions, the system-level backup is a fuzzy copy, which might not contain committed data and might contain uncommitted data. The RESTORE SYSTEM utility uses these backups to restore databases to a given point in time. The DB2 data is made consistent by DB2 restart processing and the RESTORE SYSTEM utility. DB2 restart processing determines which transactions were active at the given recovery point, and writes the compensation log records for any uncommitted work that needs to be backed out. The RESTORE SYSTEM utility restores the database copy pool, and then applies the log records to bring the DB2 data to consistency. During the

LOGAPPLY phase of the RESTORE SYSTEM utility, log records are applied to redo the committed work that is missing from the system-level backup, and log records are applied to undo the uncommitted work that might have been contained in the system-level backup.

Data-only system backups

The BACKUP SYSTEM DATA ONLY utility control statement creates system-level backups that contain only databases.

The RESTORE SYSTEM utility uses these backups to restore databases to a given point in time. In this type of recovery, you lose only a few seconds of data, or none, based on the given recovery point. However, recovery time varies and might be extended due to the processing of the DB2 logs during DB2 restart and during the LOGAPPLY phase of the RESTORE SYSTEM utility. The number of logs to process depends on the amount of activity on your DB2 system between the time of the system-level backup and the given recovery point.

Full system backups

The BACKUP SYSTEM FULL utility control statement creates system-level backups that contain both logs and databases. With these backups, you can recover your DB2 system to the point in time of a backup by using normal DB2 restart recovery, or to a given point in time by using the RESTORE SYSTEM utility.

To recover your DB2 system to the point in time of a backup by using normal DB2 restart recovery, stop DB2, and then restore both the database and log copy pools outside of DB2 by using DFSMShsm FRRECOV COPYPOOL (cpname) GENERATION (gen). After you successfully restart DB2, your DB2 system has been recovered to a point of consistency based on the time of the backup.

The RESTORE SYSTEM utility uses full system backup copies as input, but the utility does not restore the volumes in the log copy pool. If your situation requires that the volumes in the log copy pool be restored, you must restore the log copy pool before restarting DB2. For example, you should restore the log copy pool when you are using a full system-level backup at your remote site for disaster recovery.

When you recover your DB2 system to the point in time of a full system backup, you could lose a few hours of data, because you are restoring your DB2 data and logs to the time of the backup. However, recovery time is brief, because DB2 restart processing and the RESTORE SYSTEM utility need to process a minimal number of logs.

If you choose not to restore the log copy pool prior to running the RESTORE SYSTEM utility, the recovery is equivalent to the recovery of a system with data-only backups. In this type of recovery, you lose only a few seconds of data, or none, based on the given recovery point. However, recovery time varies and might be extended due to the processing of the DB2 logs during DB2 restart and during the LOGAPPLY phase of the RESTORE SYSTEM utility. The number of logs to process depends on the amount of activity on your DB2 system between the time of the system-level backup and the given recovery point.

You can use the BACKUP SYSTEM utility to manage system-level backups on tape. Choose either DUMP or DUMPONLY to dump to tape.

Restriction: The DUMP and DUMPONLY options require z/OS Version 1.8.

Use the DUMP and DUMPONLY options for:

- Managing the available DASD space.
- Retaining the system-level backups for the long term.
- Providing a means of recovery after a media failure.
- Remote site recovery procedure.

Related concepts:

"Considerations for using the BACKUP SYSTEM utility and DFSMShsm" on page 33

Related tasks:

"Recovering a DB2 subsystem to a prior point in time" on page 536

"Recovering a DB2 system to a given point in time using the RESTORE SYSTEM utility" on page 462

Related reference:

"Implications of moving data sets after a system-level backup" on page 435

BACKUP SYSTEM (DB2 Utilities)

RESTORE SYSTEM (DB2 Utilities)

Related information:

Defining Copy Pools (DFSMSdfp Storage Administration)

Point-in-time recovery using the RECOVER utility

Within certain limitations, the RECOVER utility can use system-level backups that you created by using the BACKUP SYSTEM utility for object-level recoveries.

Restriction: Recovery of objects from system-level backups requires z/OS Version 1.8 or later. You also must set the SYSTEM_LEVEL_BACKUPS subsystem parameter to YES.

The BACKUP SYSTEM utility invokes the DFSMShsm fast replication function to take volume level backups using FlashCopy. The RESTORE phase of the RECOVER utility is faster if you use these backups, because DB2 table spaces and index spaces can be restored by using FlashCopy. You might need to change some utility control statements to use FlashCopy technology for the LOAD utility, the REORG utility, and the REBUILD INDEX utility.

If any of the following utilities were run since the system-level backup that was chosen as the recovery base, the use of the system-level backup by the RECOVER utility is prohibited:

- REORG TABLESPACE
- REORG INDEX
- REBUILD INDEX
- LOAD REPLACE
- RECOVER from image copy or concurrent copy

In these cases, the recovery terminates with message DSNU1528I and return code 8.

Note: For z/OS Version 1 Release 11 and later, the RECOVER utility can use a system-level backup, even if the REBUILD INDEX, RECOVER, REORG, and LOAD

utilities ran after the system-level backup was created. The RECOVER utility has been modified so that you can use system-level backups, even if a data set has moved since the backup was created.

Related concepts:

"Data restore of an entire system" on page 469

Related reference:

"Implications of moving data sets after a system-level backup" on page 435

Related information:

DSNU1528I (DB2 Messages)

Options for restoring data to a prior point in time

TOCOPY, TOLOGPOINT, TOLASTCOPY, TORBA, and TOLASTFULLCOPY are options of the RECOVER utility. These RECOVER utility options recover data to a prior time, not to the present time. Therefore, RECOVER utility jobs that use these options result in what are referred to as point-in-time recoveries.

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*.

With TOLOGPOINT, the RECOVER utility restores the object from the most recent of either a full image copy or system-level backup taken prior to the recovery point. If a full image copy is restored, the most recent set of incremental copies that occur before the specified log point are restored. The logged changes are applied up to, and including, the record that contains the log point. If no full image copy or system-level backup 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.

Uncommitted transactions running at the recover point-in-time are automatically detected and the changes on the recovered objects are rolled back leaving them in a transactionally consistent state.

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 is to stop. 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 information.

Data consistency for point-in-time recoveries

The RECOVER utility can automatically detect uncommitted transactions that are running at the recover point in time and roll back the changes on the recovered objects. After recovery, the objects are left in their transactionally consistent state.

RECOVER TOLOGPOINT and RECOVER TORBA have the recover with consistency as their default behavior. However, RECOVER TOCOPY, TOLASTCOPY, and TOLASTFULLCOPY using SHRLEVEL CHANGE image copy do not ensure data 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

Tip: If you take SHRLEVEL CHANGE image copies and need to recover to a prior point in time, then you can use the RBA or LRSN (the START_RBA syscopy column value) associated with image copy as the TOLOGPOINT value.

All page sets must be restored to the same level; otherwise the data is inconsistent.

Point-in-time recovery can cause table spaces to be placed in CHECK-pending status if they have table check constraints or referential constraints that are 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 that contains the table with the referential constraint.

The RECOVER utility sets various states on table spaces. The following point-in-time recoveries set various 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 or XML 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 or XML table space is placed in CHECK-pending (CHKP) status.
- When you recover the LOB or XML table space, but not the base table space, to any previous point in time, the base table space is placed in auxiliary CHECK-pending (ACHKP) status, and the index space that contains 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 that contains 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 the redistribution of data across partitions, all affected partitions 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.
- When you recover a table space to a point in time prior to the REORG or LOAD REPLACE that first materializes the default value for the row change timestamp column, the table space that contains the table with the row change timestamp column is placed in REORG-pending (REORP) status.

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.

If a table space or partition that is in reordered row format is recovered to a point in time when the table space or partition was in basic row format, the table space or partition will revert to basic row format.

Related concepts:

Recovering a table space that contains LOB or XML data (DB2 Utilities)

Related information:

"Recovering from referential constraint violation" on page 547

The RECOVER TOLOGPOINT option in a data sharing system:

You can use the RECOVER utility with the TOLOGPOINT option to recover a data sharing DB2 subsystem.

The following figure shows a data sharing system with three DB2 members (A, B, and C). Table space TS1 and TS2 are being recovered to time TR using the RECOVER TOLOGPOINT option, they are listed in the same RECOVER job. UR1 was inflight at TR time running on member A, its start time was T1, at time T2 it updated TS1, and at time T3 it updated TS2, T2 is earlier than T3. UR2 was

aborting at TR time running on member C, its start time was T4 and at time T5 it updated TS2. There was no active UR on member B at time TR.

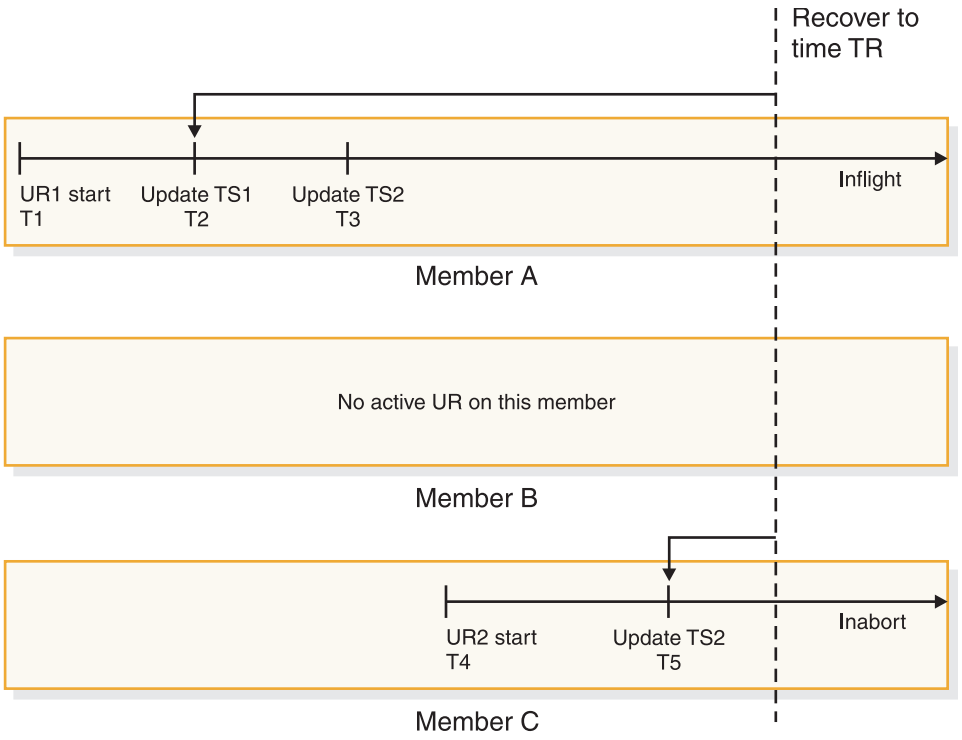


Figure 42. Using the RECOVER TOLOGPOINT option in a data sharing system

RECOVER utility job output messages

The RECOVER utility takes the following actions and provides the following messages during recovery in a data sharing system.

LOGCSR phase

After the RECOVER LOGAPPLY phase, the RECOVER utility enters the log analysis phase, known as the LOGCSR phase. The following messages are issued during this phase:

DSNU1550I

Indicates the start of log analysis on member A.

DSNU1551I

Indicates the end of log analysis on member A.

DSNU1550I

Indicates the start of log analysis on member C.

DSNU1551I

Indicates the end of log analysis on member C.

DSNU1552I

Indicates the end of LOGCSR phase of RECOVER utility.

DSNU1553I

Issued after the end of the LOGCSR phase. The following information is shown in the message:

- UR1 on member A modified TS1 at T2 time.
- UR1 on member A modified TS2 at T3 time.

- UR2 on member C modified TS2 at T5 time.

LOGUNDO phase

The RECOVER utility enters the LOGUNDO phase. The following messages are issued during this phase:

DSNU1554I

Indicates the start of backout for member A.

- Backout is performed on TS2 and TS1 and log records are read from time TR to T2.
- There maybe one or more DSNU1555I messages showing the backout progress for member A.

DSNU1556I

Indicates the end of backout on member A.

DSNU1554I

Indicates the start of backout on member C.

- Backout is performed on TS2 and log records are read from time TR to T5.
- There may be one or more DSNU1555I messages showing the backout progress for member C.

DSNU1556I

Indicates the end of backout for member C.

DSNU1557I

Indicates the end of the LOGUNDO phase of the RECOVER utility.

A special type of compensation log record will be written during the LOGUNDO phase of log apply, known as the *pseudo* compensation log record in the following context. For each undo log record applied during the LOGUNDO phase, there will be one pseudo compensation log record written for it. This is a REDO-only log record and not units of recovery (UR) related. It will only be used for future log apply in the case that you first recover objects to a point in time with consistency and then recover the same objects again to currency using the same image copy used by previous recover job.

This type of log record will be skipped by DB2 restart and the RESTORE SYSTEM utility LOGAPPLY. This type of log record will always be written to the active log data sets of the DB2 member that is running the RECOVER job, although it may compensate the log record which was originally from another DB2 member. The member ID part of this log record will always store the ID of the DB2 member that is running the RECOVER job.

During the LOGUNDO phase, if either of the following conditions exist:

- Applying the log record to a data page causes the data on the page logically inconsistent.
- The UNDO log record has already been applied and the page is now physically inconsistent.

The page is flagged as broken and the pseudo compensation log record of the log record that is being backed out is written to the active log data set. All of the subsequent log apply processes on this data page are skipped, but the pseudo compensation log records of the skipped log records are written to the active log data set. The log apply process continues for other pages in the same table space and for other objects in the recover list. For

each error that is encountered, a DSNI012I message is issued. At the end, the RECOVER utility completes with return code 8.

If an error occurs on the index during the LOGUNDO phase, the entire index is marked as REBUILD-pending (RBDP) and no further log is applied on this index. You have to rebuild this index after the RECOVER utility completes with return code 8.

UTILTERM phase
The RECOVER utility enters the UTILTERM phase, which is an existing phase of the RECOVER utility.

The RECOVER TOLOGPOINT option in a non-data sharing system:

You can use the RECOVER utility with the TOLOGPOINT option to recover a non-data sharing DB2 subsystem.

Figure 43 shows a non-data sharing system. Table spaces TS1 and TS2 are being recovered to time TR using the RECOVER TORBA option. TS1 and TS2 are listed in the same RECOVER job. UR1 was inflight at TR time, the start time was T1, at time T2 it updated TS1, and at time T3 it updated TS2. T2 is earlier than T3. UR2 was aborting at TR time, the start time was T4, and at time T5 it updated TS2.

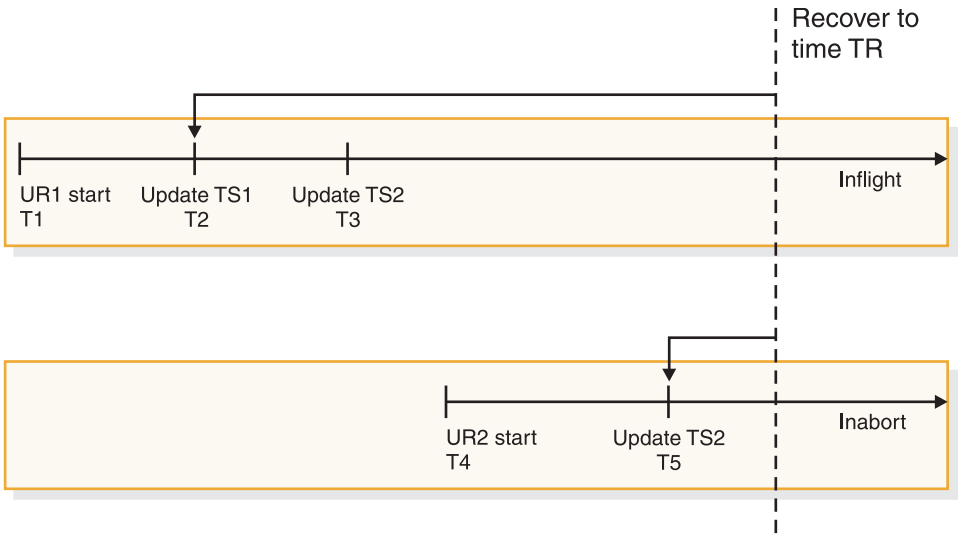


Figure 43. Using the RECOVER TOLOGPOINT option in a non-data sharing system

RECOVER utility job output messages

The RECOVER utility takes the following actions and provides the following messages during recovery in a non-data sharing system.

LOGCSR phase
After the LOGAPPLY phase, the RECOVER utility enters the log analysis phase, known as the LOGCSR phase. The following messages are issued during this phase:

DSNU1550I
Indicates the start of log analysis.

DSNU1551I
Indicates the end of log analysis.

DSNU1552I
Indicates the end of the LOGCSR phase of the RECOVER utility.

DSNU1553I
Issued after the end of the LOGCSR phase. The following information is shown in the message:

- UR1 modified TS1 at T2 time.
- UR1 modified TS2 at T3 time.
- UR2 modified TS2 at T5 time.

LOGUNDO phase
The RECOVER utility enters the LOGUNDO phase. The following messages are issued during this phase:

DSNU1554I
Indicates the start of backout.

- Backout is performed on TS2 and TS1 and log records are read from time TR to T2.
- There may be one or more DSNU1555I messages showing the backout progress.

DSNU1556I
Indicates the end of backout.

DSNU1557I
Indicates the end of LOGUNDO phase of the RECOVER utility.

A special type of compensation log record will be written during the LOGUNDO phase of log apply, known as the *pseudo* compensation log record in the following context. For each undo log record applied during the LOGUNDO phase, there will be one pseudo compensation log record written for it. This is a REDO-only log record and not units of recovery (UR) related. It will only be used for future log apply in the case that you first recover objects to a point in time with consistency and then recover the same objects again to currency using the same image copy used by previous recover job.

This type of log record will be skipped by DB2 restart and the RESTORE SYSTEM utility LOGAPPLY. This type of log record will always be written to the active log datasets of the DB2 member that is running the RECOVER job, although it may compensate the log record which was originally from another DB2 member. The member ID part of this log record will always store the ID of the DB2 member that is running the RECOVER job.

During the LOGUNDO phase, if either of the following conditions exist:

- Applying the log record to a data page causes the data on the page logically inconsistent.
- The UNDO log record has already been applied and the page is now physically inconsistent.

The page is flagged as broken and the pseudo compensation log record of the log record that is being backed out is written to the active log dataset. All of the subsequent log apply processes on this data page are skipped, but the pseudo compensation log records of the skipped log records are written to the active log dataset. The

log apply process continues for other pages in the same table space and for other objects in the recover list. For each error that is encountered, a DSNI012I message is issued. At the end, the RECOVER utility completes with return code 8.

If an error occurs on the index during the LOGUNDO phase, the entire index is marked as REBUILD-pending (RBDP) and no further log is applied on this index. You have to rebuild this index after the RECOVER utility completes with return code 8.

UTILTERM phase
The RECOVER utility enters the UTILTERM phase, which is an existing phase of the RECOVER utility.

Recommendations for recovery of compressed data

Use care 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.

Implications of moving data sets after a system-level backup

If you operate on z/OS Version 1 Release 10 or earlier, the movement of data sets after the creation of a system-level backup can affect the ability to do recovery to a prior point in time.

Note: For z/OS Version 1 Release 11 and later, to recover data sets from a system-level backup, the data sets do not need to reside on the same volume as they were on when the backup was made. The RECOVER utility has been modified so that you can use system-level backups, even if a data set has moved since the backup was created.

You can still recover the objects if you have an object-level backup such as an image copy or concurrent copy. Take object-level backups to augment system-level backups, or take new system-level backups whenever a data set is moved.

You can force the RECOVER utility to ignore system-level backups and use the latest applicable object-level backup by setting the SYSTEM-LEVEL-BACKUPS parameter on installation panel DSNTIP6 to NO. This subsystem parameter can be updated online.

Activities that can affect the ability to recover an object to a prior point in time from a system-level backup include:

- Migrating to new disk storage or redistributing data sets for performance reasons

  The movement of data sets for disk management should be restricted or limited when system-level backups are taken. When movement of data sets does occur, a new system-level backup or object-level backup should immediately be taken.
- Using the DFSMShsm migrate and recall feature

  Do not use the DFSMShsm migrate and recall feature if you take system-level backups.
- Using REORG TABLESPACE or LOAD REPLACE

  To minimize the impact of REORG TABLESPACE and LOAD REPLACE on recoverability, you should always take an inline copy of the underlying table space. You can then use this copy to recover at the object level.
- Using REORG INDEX or REBUILD INDEX

If a REORG INDEX or REBUILD INDEX is preventing the use of the system-level backup during recovery of an index, you can use the REBUILD INDEX utility to rebuild rather than recover the index.

- Using RECOVER from an image copy or concurrent copy

If you operate on z/OS Version 1 Release 10 or earlier, these activities prevent the recovery at the object level to a point in time that would use the system-level backup. This restriction happens because the current volume allocation information in the ICF catalog for the data set (or data sets) differs from the volume allocation at the time of the system-level backup.

Consider the following restrictions when planning your backup strategy:

- Data sets can move within a single volume, for example because the volume has been defragmented. In this case, the ability to restore the data is not affected. Recovery of data is not affected if the data set is extended to another volume after it has been copied.

  In the case where the data set originally spanned multiple volumes, the data set would need to be reallocated with extents on each of the same volumes before it could be recovered successfully. The amount of space for the data set extents at the time of the backup can differ from the amount of space that is available at recovery time.
- The RECOVER utility cannot use the copy pool backup as the source for a recovery if the data set has been moved to different volumes, if you operate on z/OS Version 1 Release 10 or earlier.
- The movement of one data set in a system-level backup does not prevent the object-level recovery of other objects from the backup.
- The movement of data sets does not prevent the use of the RESTORE SYSTEM utility.

Recovery of table spaces

The way that you recover DB2 table spaces depends on several factors, including the type of table space that needs recovery.

When you recover table spaces to a prior point of consistency, you need to consider:

- How partitioned table spaces, segmented table spaces, LOB table spaces, XML table spaces, and table space sets can restrict recovery.
- If you take system-level backups, how certain utility events can prohibit recovery to a prior point in time.

Related reference:

"Implications of moving data sets after a system-level backup" on page 435

Recovery of partitioned table spaces

Recovering partitioned table spaces to a point in time has several limitations.

You cannot recover a table space to a point in time that is 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 that is 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 that is before the data was redistributed across table space partitions, you must include all partitions that are affected by that redistribution in your recovery list.

See the information about point-in-time recovery for more details on these restrictions.

Related concepts:

Point-in-time recovery (DB2 Utilities)

Recovery of segmented table spaces

You can restore data on a segmented table space to a prior point in time. When you do this, information in the database descriptor (DBD) for the table space might not match the restored table space.

If you use the DB2 RECOVER utility, the 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 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.

Recovery of LOB table spaces

When you recover tables with LOB columns, recover the entire set of objects. The entire set of objects includes 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, the RECOVER utility might place the table space in a pending state.

Related concepts:

"Options for restoring data to a prior point in time" on page 428

Recovery of XML table spaces

At times, recovering database objects to a prior point in time is necessary. When this type of recovery is needed, all related objects (including XML objects) must be recovered to a consistent point in time.

The RECOVER utility performs consistency checking of the recovery points of these related objects during point-in-time recoveries. Use the REPORT utility with the TABLESPACESET keyword to obtain a list of these related objects. Use the QUIESCE utility with the TABLESPACESET keyword if you need to quiesce all objects in the related set.

Recovery of 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.

A group of related table spaces is called a *table space set*. Specifically, a table space set includes the following groups of table spaces:

- Table spaces that contain referentially related tables
- A base table space and its LOB table spaces
- A base table space and its XML table spaces

- A table space with a system-period temporal table and the table space with the related history table

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 identify a department that does not exist in the restored DEPARTMENT table.

Run the CHECK INDEX utility to validate the consistency of indexes with their associated table data. Run the CHECK DATA utility to validate the consistency of base table space data with related table spaces. If LOB columns exist, run the CHECK LOB utility on any related LOB table spaces to validate the integrity of each LOB table space within itself.

You can use the REPORT utility with the TABLESPACESET option to determine all of the page sets that belong to a single table space set. Then, you can 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 of those page sets on your own.

To determine a valid quiesce point for a table space set, determine a RECOVER TOLOGPOINT value.

Related concepts:

Creation of relationships with referential constraints (Introduction to DB2 for z/OS)

Large object table spaces (Introduction to DB2 for z/OS)

XML table spaces (Introduction to DB2 for z/OS)

Temporal tables

Related reference:

CHECK DATA (DB2 Utilities)

CHECK INDEX (DB2 Utilities)

CHECK LOB (DB2 Utilities)

RECOVER (DB2 Utilities)

REPORT (DB2 Utilities)

Recovery of partition-by-growth table spaces

The RECOVER utility supports both table space-level and partition-level recovery on a partition-by-growth table space.

If an image copy was made on the partition level, the table space can only be recovered at the partition level. A violation of this rule causes message DSNU512I to be issued.

Because the number of partitions is defined on demand, the total number of partitions in an image copy may not be consistent with the number of partitions that reside in the current table space. In such a case, when recovering the table space back to a point in time with an image copy that has less partitions, the excess partitions in the table space will be empty because the recover process has reset the partitions.

Recovery of indexes

When you recover indexes to a prior point of consistency, some rules apply.

In general, the following rules apply:

- If image copies exists for an indexes, use the RECOVER utility.
- If you take system-level backups, use the RECOVER utility. (This use requires z/OS Version 1.8.)
- If indexes do not have image copies or system-level backups, 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.

Before attempting recovery, analyze the recovery information.

Related concepts:

"How to report recovery information" on page 409

Related reference:

"Implications of moving data sets after a system-level backup" on page 435

Recovery of indexes on altered tables

Using some ALTER statements interferes with the use of the RECOVER utility to restore the index to a point in time before the ALTER statement was used.

GUPI

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
- ALTER INDEX REGENERATE

GUPI

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 point of recovery, you must use the REBUILD INDEX utility to rebuild these indexes.

Related reference:

REBUILD INDEX (DB2 Utilities)

RECOVER (DB2 Utilities)

Recovery of indexes on tables in partitioned table spaces

The partitioning of secondary indexes allows you to copy and recover indexes at the entire index level or individual partition level. Certain restrictions apply to using the COPY and RECOVER utilities on the index level and partition level.

If you use the COPY utility at the partition level, you need to use the RECOVER utility at the partition level, too. If you use the COPY utility at the partition level and then try to the RECOVER utility the index, an error occurs. If the COPY utility is used at the index level, you can use the RECOVER utility at either the index level or the partition level.

You cannot recover an index space to a point in time that is prior to rotating partitions. After you rotate a partition, you cannot recover the contents of that partition to a point in time that is before the rotation.

If you recover to a point in time that is 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.

Recovery of FlashCopy image copies

Using FlashCopy image copies can reduce the amount of time that is required for recovery operations.

To provide a recovery base for media failures, create one or more additional sequential format image copies when you create a FlashCopy image copy. If the FlashCopy image copy has been migrated or deleted, the recovery proceeds from the sequential image copies if available. The following utilities can create additional sequential format image copies in a single execution:

- COPY
- LOAD with the REPLACE option specified
- REORG TABLESPACE

You can recover an object to a specific FlashCopy image copy by specifying the RECOVER utility with the TOCOPY, TOLASTCOPY, or TOLASTFULLCOPY options. If the object is partitioned, you must specify the data set number on the DSNUM parameter in the RECOVERY utility control statement for each partition that is being recovered.

A FlashCopy image copy with consistency consumes more processing resources when the image copy is created, and when the image copy is used for recovery. To recover from a FlashCopy image copy with consistency, the RECOVER utility must read the logs to apply changes that were made to the recovered object after the point of consistency. Some of the changes could be work that was previously backed out and that must be reapplied, because the work was uncommitted when the FlashCopy image copy was created.

For recovery to a log point or full recovery, if uncommitted work was backed out from the FlashCopy image copy during consistency processing, recovery requires more analysis of the logs during the preliminary LOGCSR phase (PRELOGC). The preliminary log apply phase (PRELOGA) and the other log phases also require more analysis.

Related concepts:

FlashCopy image copies (DB2 Utilities)

Related tasks:

"Creating FlashCopy image copies" on page 420

Preparing to recover to a prior point of consistency

DB2 begins recovery with the image copy or system-level backup that you made and reads the log only up to the point of consistency. At that point, no indoubt units of recovery hinder restarting DB2.

About this task

Related reference:

Implications of moving data sets after a system-level backup

Identifying objects to recover

You must recover dependent objects to the same point in time. Dependent objects include table spaces that are part of a table space set.

Procedure

To identify the objects that must be recovered to the same point in time:

Run the REPORT utility with the TABLESPACESET option.

Related concepts:

"Recovery of table space sets" on page 437

Creation of relationships with referential constraints (Introduction to DB2 for z/OS)

Large object table spaces (Introduction to DB2 for z/OS)

XML table spaces (Introduction to DB2 for z/OS)

Related reference:

Syntax and options of the REPORT control statement (DB2 Utilities)

Resetting exception status

You need to determine whether the data is in an exception status, and then you can release the data from any exception status.

Procedure

To determine whether the data is in an exception status:

Issue the DISPLAY DATABASE RESTRICT command.

Copying the data

You can use the COPY utility to copy the data, taking appropriate precautions about concurrent activity.

About this task

In one operation, you can copy the data and establish a point of consistency for a list of objects 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 time when copying starts and remains consistent until copying ends. The advantage of this operation 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 while 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.

Ideally, you should copy data without allowing updates. However, restricting updates is not always possible. To allow updates while the data is being copied, you can take either of the following actions:

- Use the COPY utility with the SHRLEVEL CHANGE option.
- Use an offline program to copy the data, such as DSN1COPY, DFSMShsm, or disk dump.

If you allow updates while copying, step 3 is recommended. With concurrent updates, the copy can include uncommitted changes. Those might be backed out after copying ends. Thus, the copy does not necessarily contain 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. Although RECOVER can recover your data to any point in time and ensure data consistency, recovering to a quiesce point can be more efficient. Therefore, taking periodic quiesce points is recommended, if possible.

You can copy all of the data in your system, including the DB2 catalog and directory data by using the BACKUP SYSTEM utility. Since the BACKUP SYSTEM utility allows updates to the data, the system-level backup can include uncommitted data.

Related reference:

COPY (DB2 Utilities)

Establishing a point of consistency

After copying the data, immediately establish a point when the data is consistent and no unit of work is changing it.

Procedure

To establish a single point of consistency, or a *quiesce point*, for one or more page sets:

Run the QUIESCE utility.
Typically, you name all of the table spaces in a table space set that you want recovered to the same point in time to avoid referential integrity violations. Alternatively, you can use the QUIESCE utility with the TABLESPACESET keyword for referential integrity-related tables.
The QUIESCE utility writes changed pages from the page set to disk. The SYSIBM.SYSCOPY catalog table 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 that is quiesced. Page sets DSNDB06.SYSCOPY, DSNDB01.DBD01, DSNDB01.SYSUTILX, and DSNDB01.SYSDBDXA 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.
The QUIESCE utility allows concurrency with many other utilities; however, it does not allow concurrent updates until it has quiesced all specified page sets and depending on the amount of activity, that can take considerable time. Try to run the QUIESCE utility when system activity is low.
Also, consider using the MODE(QUIESCE) option of the ARCHIVE LOG command when planning for off-site 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.

Example

The following statement quiesces two table spaces in database DSN8D10A:

```
QUIESCE TABLESPACE DSN8D10A.DSN8S10E
        TABLESPACE DSN8D10A.DSN8S10D
```

Related tasks:

"Archiving the log" on page 348

Related reference:

QUIESCE (DB2 Utilities)

Preparing to recover an entire DB2 subsystem to a prior point in time using image copies or object-level backups

Under certain circumstances, you might want to reset the entire DB2 subsystem to a point of consistency.

Procedure

To prepare a point of consistency:

1. Display and resolve any indoubt units of recovery.
2. Use the COPY utility to make image copies of all data, including user data, DB2 catalog and directory table spaces, and optionally indexes. Copy SYSLGRNX and SYSCOPY last.

   Installation 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.

   Alternate method: Alternatively, you can use an off-line method to copy the data. In that case, stop DB2 first; that is, do the next step before doing this step. 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).

   Important: Be sure to use MODE (QUIESCE); otherwise, I/O errors can occur when you fall back before a DB2 restart.

   DB2 does not actually stop until all currently executing programs have completed processing.
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.

Related tasks:

Installation step 21: Back up the DB2 directory and catalog: DSNTIJIC (DB2 Installation and Migration)

Creating essential disaster recovery elements

You must take steps to create essential disaster recovery elements. For example, you must determine how often to make copies and send them to the recovery site.

Procedure

To create essential disaster recovery elements:

1. Make image copies:
   a. 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.

      Tip: You can also use these copies on a subsystem that is installed with the LOCALSITE option if you run RECOVER with the RECOVERYSITE option. Alternatively, you can use copies that are prepared for the local site on a recovery site if you run RECOVER with the option LOCALSITE.

      Important: Do not produce copies by invoking COPY twice.
   b. Optional: Catalog the image copies if you want to track them.
   c. Create a QMF report or use SPUFI to issue a SELECT statement to list the contents of SYSCOPY.
   d. Send the image copies, and report to the recovery site.
   e. 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.
2. 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.

      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. Optional: 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 can also create your own tracking method and do it manually.
   c. Use the print log map utility to create a BSDS report.

d. Send the archive copy, the BSDS report, and any additional information about the archive log to the recovery site.
e. Record this activity at the recovery site when the archive copy and the report are received.

3. Choose consistent system time:

   Important: After 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.

   a. 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.

   b. Ensure that the system clock remains consistent.

4. Back up integrated catalog facility catalogs:
   a. Back up all DB2-related integrated catalog facility catalogs with the VSAM EXPORT command on a daily basis.
   b. Synchronize the backups with the cataloging of image copies and archives.
   c. Use the VSAM LISTCAT command to create a list of the DB2 entries.
   d. Send the EXPORT backup and list to the recovery site.
   e. Record this activity at the recovery site when the EXPORT backup and list are received.
5. Back up DB2 libraries:
   a. 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.
   b. Back up the DSNTIJUZ job that builds the ZPARM and DECP modules.
   c. Back up the data set allocations for the BSDS, logs, directory, and catalogs.
   d. Document your backups.
   e. Send backups and corresponding documentation to the recovery site.
   f. Record activity at the recovery site when the library backup and documentation are received.

What to do next

For disaster recovery to be successful, all copies and reports must be updated and sent to the recovery site regularly. Data is up to date through the last archive that is sent.

Related concepts:

Multiple image copies (DB2 Utilities)

Related tasks:

"Archiving the log" on page 348

Related information:

"Performing remote-site disaster recovery" on page 553

Resolving problems with a user-defined work file data set

You can resolve problems on a volume of a user-defined data set for the work file database.

Procedure

To resolve problems:

1. Issue the following DB2 command:

   ```
   -STOP DATABASE (DSNDB07)
   ```

   Note: If you are adding or deleting work files, you do not need to 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)
   ```

Resolving problems with DB2-managed work file data sets

You can resolve problems on a volume in a DB2 storage group for the work file database, such as a system I/O problem.

Procedure

GUPI To resolve problems with the work file database:

1. Issue 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 to stop the database:

   ```
   -STOP DATABASE (DSNDB07)
   ```

   Note: If you are adding or deleting work files, you do not need to stop database DSNDB07.
3. Issue the following SQL statement to drop the table space that has 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 volume.

   ```
   CREATE TABLESPACE tsname
     IN DSNDB07
     USING STOGROUP stogroup-name;
   ```
5. Issue the following command to restart the database:

   ```
   -START DATABASE (DSNDB07)
   ```

GUPI

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 that you use to recover them. You cannot reset error ranges in a work file table space by using the ERROR RANGE option of the RECOVER utility.

Procedure

To recover error ranges for a work file table space:

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.

Recovery of error ranges for a work file table space

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, DB2 sets the error range again.

Recovering after a conditional restart of DB2

After a DB2 conditional restart in which a log record range is specified, a portion of the DB2 recovery log is no longer available.

About this task

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 fails with ABEND04E RC00E40119.

Procedure

Instead of using the RECOVER utility, use the following procedure to recover those table spaces and their indexes:

1. Run DSN1COPY to restore 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 make the table spaces and indexes recoverable.

Recovery of the catalog and directory

Catalog and directory objects must be recovered in a particular order.

The order differs, depending on whether you are in conversion mode, enabling-new-function mode, or new-function mode. For some catalog and directory objects the order differs, depending on whether objects were dropped or deleted, and new objects created.

Sometimes the recovery of some catalog and directory objects depends on information that is derived from other catalog and directory objects. You must recover some of these objects in separate RECOVER utility control statements.

However, you can 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 that you perform on the catalog.

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, you must set subsystem parameters DEFER and ALL. You can do this by setting the values DEFER in field 1 and ALL in field 2 of installation panel DSNTIPS.

Important: Recovering the DB2 catalog and directory to a prior point in time is strongly discouraged.

Related concepts:

"How to report recovery information" on page 409

Copying catalog and directory objects (DB2 Utilities)

Related tasks:

"Deferring restart processing" on page 372

Recovering catalog and directory objects (DB2 Utilities)

Related reference:

RECOVER (DB2 Utilities)

Regenerating missing identity column values

You can regenerate missing identity column values.

Procedure

To regenerate missing identity column values:

1. GUPI 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
```

   GUPI
2. Run the REORG utility to regenerate lost sequence values.

   If you do not choose a starting value in step 1, the REORG utility generates a sequence of identity column values that starts with the next value that DB2 would have assigned before the recovery.

Recovery of 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.

Identity column already defined

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.

For example, assume that at time T1 an identity column that is incremented 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.

GUPI

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
```

GUPI

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.

Identity column not yet defined

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.

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 that is before the identity column was defined. To access the recovered table, you need to remove this status.

Related concepts:

"Data consistency for point-in-time recoveries" on page 429

Recovering a table space and all of its indexes

You can recover a table space and all of its indexes (or a table space set and all related indexes). Use a single RECOVER utility statement that specifies the TOLOGPOINT option.

About this task

For the log point, you can 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, which ensures that the table spaces and indexes are synchronized and eliminates the need to run the CHECK INDEX utility.

With recovery to a point in time with consistency, which is the default recovery type, you do not need to identify a quiesce point or a common SHRLEVEL REFERENCE copy point. This recovery might be faster because inconsistencies do not have to be resolved.

Procedure

To recover to a log point:

1. Use the RECOVER utility to recover table spaces to the log point.
2. Use concurrent REBUILD INDEX jobs to rebuild the indexes for each table space.

Recovery implications for objects that are not logged

You can use the RECOVER utility on objects that have the NOT LOGGED attribute. The NOT LOGGED attribute does not mean that the contents of an object are not recoverable. However, the modifications to the object that is not logged are not recoverable.

Objects that are not logged include the table space, the index, and the index space. Recovery can be to any recoverable point. A *recoverable point* is established when:

- A table space is altered from logged to not logged.
- When an image copy is taken against an object that is not logged.
- An ALTER TABLE statement is issued with the ADD PARTITION clause, against a table in a table space that has the NOT LOGGED attribute.
- DB2 adds a new partition in response to insertion of data into a partition-by-growth table space.

The TORBA or TOLOGPOINT keywords can also be used for a point-in-time recovery on an object that is not logged, but the RBA or LRSN must correspond to a recoverable point or message DSNU1504I is issued.

If a base table space is altered so that it is not logged, and its associated LOB table spaces already have the NOT LOGGED attribute, then the point where the table space is altered is not a recoverable point.

If DB2 restart recovery determines that a table space that is not logged might have been in the process of being updated at the time of the failure, then the table space or partition is placed in the LPL and is marked RECOVER-pending.

Related concepts:

"The NOT LOGGED attribute" on page 118

Related reference:

ALTER TABLE (DB2 SQL)

Clearing the informational COPY-pending status (ICOPY)

If you update a table space that is defined with the NOT LOGGED attribute, the table space is put in informational COPY-pending status (ICOPY).

Procedure

To clear the ICOPY status:

1. First, use the DISPLAY DATABASE ADVISORY command to display the ICOPY status for table spaces. For example:

```
DSNT36I1 - * DISPLAY DATABASE SUMMARY
              * ADVISORY
     DSNT360I - ************************************
     DSNT362I -  DATABASE = DBIQUA01 STATUS = RW
                 DBD LENGTH = 8066
     DSNT397I -
     NAME     TYPE PART STATUS       HYERRLO PHYERRHI CATALOG PIECE
     -------- ---- ---- ------------ ------- -------- ------- -----
     TPIQUQ01 TS     001 RW,AUXW
     TPIQUQ01 TS     002 RW,AUXW
     TPIQUQ01 TS     003 RW,AUXW
     TPIQUQ01 TS     004 RW,ICOPY
```

2. To clear the ICOPY status, you must take a full image copy of the table space.

The LOG option of the LOAD or REORG utilities

The LOG option that you specify when you run the LOAD and REORG utilities has different results depending on the logging attribute of the table space that is being logged.

The following tables show how the logging attribute of the utility and the logging attribute of the table space interact:

Table 34. Attribute interaction for LOB table spaces

LOAD or REORG keyword	Table space logging attribute	What is logged	Table space status after completion
LOG YES	LOGGED	Control records and LOB data redo information. LOB data undo information is never logged for LOB table spaces.	No pending status
LOG YES	NOT LOGGED	Control information.	No pending status
LOG NO	LOGGED	Nothing.	COPY-pending
LOG NO	NOT LOGGED	Nothing.	No pending status

Table 35. Attribute interaction for non-LOB table spaces

LOAD or REORG keyword	Table space logging attribute	What is logged	Table space status after completion
LOG YES	LOGGED	Control records and data.	No pending status

Table 35. Attribute interaction for non-LOB table spaces (continued)

LOAD or REORG keyword	Table space logging attribute	What is logged	Table space status after completion
LOG YES	NOT LOGGED	LOG YES is changed to LOG NO.	See Table 36
LOG NO	LOGGED	Nothing.	COPY-pending
LOG NO	NOT LOGGED	Nothing.	See Table 36

The following table shows the possible table space statuses for non-LOB tables spaces that are not logged:

Table 36. Status of non-LOB table spaces that are not logged, after LOAD or REORG with LOG NO keyword

Inline copy	Records discarded	Table space status
Yes	No	No pending status
Yes	Yes	ICOPY-pending
No	not applicable	ICOPY-pending

Clearing the RECOVER-pending status

If DB2 needs to undo work that has not been logged (as when a rollback occurs), the table space has lost its data integrity and is marked RECOVER-pending. To prevent access to corrupt data, DB2 places the pages in the logical page list (LPL).

About this task

Tip: Application programmers should commit frequently and try to avoid duplicate key or referential integrity violations when modifying a table in a NOT LOGGED table space.

If DB2 restart recovery determines that a not logged table space may have been in the process of being updated at the time of the failure, then the table space or partition is placed in the LPL and is marked RECOVER-pending. You have several options for removing a table space from the LPL and resetting the RECOVER-pending status:

- Dropping and re-creating the table space and repopulating the table
- "Using a REFRESH TABLE statement"
- "Using the RECOVER utility" on page 453
- "Using the LOAD REPLACE utility" on page 453
- "Using a DELETE statement without a WHERE clause" on page 453
- "Using a TRUNCATE TABLE statement" on page 454

When a job fails and a rollback begins, the undo records are not available for table spaces that are not logged during the back-out. Therefore, the rows that are in the table space after recovery might not be the correct rows. You can issue the appropriate SQL statements to re-create the intended rows.

Using a REFRESH TABLE statement:

About this task

GUPI

Use the REFRESH TABLE statement to repopulate a materialized query table, but only if the materialized query table is alone in its table space. If the table is not alone in its table space, a utility must be used to reset the table space and remove it from RECOVER-pending status.

GUPI

Using the RECOVER utility:
About this task

Use the RECOVER utility to recover to a recoverable point.

You can run the RECOVER utility against a table space with the NOT LOGGED logging attribute. To do so, the current logging attribute of the table space must match the logging attribute of the recovery base (that is, the logging attribute of the table space when the image copy was taken). If no changes have been made to the table space since the last point of recovery, the utility completes successfully. If changes have been made, the utility completes with message DSNU1504I.

You can use RECOVER with the TOCOPY, TOLASTFULLCOPY, or TOLASTCOPY keyword to identify which image copy to use. You can also use TORBA or TOLOGPOINT, but the RBA or LRSN must correspond to a recoverable point.

You cannot use RECOVER with the LOGONLY keyword.

Using the LOAD REPLACE utility:
About this task

Use the LOAD REPLACE utility or the LOAD REPLACE PART utility in the following situations:

- With an input data set to empty the table space and repopulate the table.
- Without an input data set to empty the table space to prepare for one or more INSERT statements to repopulate the table.

Using a DELETE statement without a WHERE clause:
About this task

GUPI

Use the DELETE statement without a WHERE clause to empty the table, when the table space is segmented or universal, the table is alone in its table space and the table does not have:

- A VALIDPROC
- Referential constraints
- Delete Triggers
- A SECURITY LABEL column (or it does have such a column, but multilevel security with row level granularity is not in effect)

GUPI

Using a TRUNCATE TABLE statement:
About this task

GUPI

Use the TRUNCATE TABLE statement to empty the table, when the table space is segmented and the table is alone in its table space and the table does not have:

- A VALIDPROC
- Referential constraints
- A SECURITY LABEL column (or it does have such a column, but multilevel security with row level granularity is not in effect)

GUPI

Removing various pending states from LOB and XML table spaces

You can remove various pending states from a LOB table space or an XML table space by using a collection of utilities in a specific order.

Procedure

To remove pending states from a LOB table space or an XML table space:

1. Use the REORG TABLESPACE utility to remove the REORP status.
2. If the table space status is auxiliary CHECK-pending status:
   a. Use CHECK LOB for all associated LOB table spaces.
   b. Use CHECK INDEX for all LOB indexes, as well as the document ID, node ID, and XML indexes.
3. Use the CHECK DATA utility to remove the CHECK-pending status.

Restoring data by using DSN1COPY

You can use the DSN1COPY utility to restore data that was previously backed up by the DSN1COPY utility or by the COPY utility. If you use the DSN1COPY utility to restore data or move data, the data definitions for the target object must be exactly the same as when the copy was created.

About this task

You cannot use the DSN1COPY utility to restore data that was backed up with the DFSMSdss concurrent copy facility.

Be careful when creating backups with the DSN1COPY utility. 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.

Related reference:

DSN1COPY (DB2 Utilities)

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. However, certain limitations exist.

About this task

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.

Recovering accidentally dropped objects

If a table or table space is inadvertently dropped, you can recover the object.

About this task

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 Log Analysis Tool for z/OS

Procedure

To recover the object:

1. Run regular catalog reports that include a list of all OBIDs in the subsystem.
2. Create catalog reports that list dependencies on the table or (such as referential constraints, indexes, and so on). After a table is dropped, this information disappears from the catalog.
3. If an OBID has been reused by DB2, 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.

How to avoid accidentally dropping objects

To avoid the problem of accidentally dropping tables, you can create a table with the clause WITH RESTRICT ON DROP.

GUPI

When a table has been created with the clause WITH RESTRICT ON DROP, then nobody can drop the table, or the table space or database that contains 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.

GUPI

Recovering an accidentally dropped table

To drop tables in a partitioned table space, you need to drop the table space itself.

Before you begin

To recover a dropped table, you need a full image copy or a DSN1COPY file that contains the data from the dropped table.

For segmented or universal 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.

Procedure

To recover 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 the 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. By searching 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:

   GUPI

   ```
   -STOP DATABASE(database-name) SPACENAM(tablespace-name)
   ```

   GUPI

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 that you created in step 2 on page 456 by querying the SYSIBM.SYSTABLES catalog table.

   GUPI

   The following statement returns the object ID (OBID) for the table:

   ```
   SELECT NAME, OBID FROM SYSIBM.SYSTABLES
     WHERE NAME='table_name'
       AND CREATOR='creator_name';
   ```

   This value is returned in decimal format, which is the format that you need for DSN1COPY.

   GUPI

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 on page 456 as both the translation source and the translation target
   b. The PSID that you recorded in step 1 on page 456 as both the translation source and the translation target
   c. The original OBID that you recorded in step 1 on page 456 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 on page 456 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.
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 Log Analysis Tool for z/OS at this point in the recovery procedure.
7. Start the table space for normal use by using the following command:

   GUPI

   ```
   -START DATABASE(database-name) SPACENAM(tablespace-name)
   ```

   GUPI
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.
11. Re-create the objects that are dependent on the recovered table.

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.

Related concepts:

"Implications of dropping a table" on page 162

"Page set and data set copies" on page 418

"Recovery of data to a prior point in time" on page 424

Related tasks:

"Recovering an accidentally dropped table space"

Related reference:

DSN1COPY (DB2 Utilities)

Recovering an accidentally dropped table space

If you accidentally drop a table space, including LOB table spaces, you can recover that table space.

About this task

You might accidentally 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.

The recovery procedures for user-managed data sets and for DB2-managed data sets are slightly different.

Recovering accidentally dropped DB2-managed data sets

If a consistent full image copy or DSN1COPY file is available, you can use DSN1COPY to recover a dropped table space that is part of the catalog.

Procedure

To recover a dropped table space:

1. Find the original DBID for the database, the PSID for the table space, and the OBIDs of all tables that are contained in the dropped table space. For information about how to do this, see step 1 on page 456 of "Recovering an accidentally dropped table" on page 456.
2. Re-create the table space and all tables. This re-creation can be difficult when any of the following conditions is 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. PSPI Find the new PSID and OBIDs by querying the SYSIBM.SYSTABLESPACE and SYSIBM.SYSTABLES catalog tables.

   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';
   ```

   These values are returned in decimal format, which is the format that you need for DSN1COPY. (Find the OBID of the dummy table that you created in step 2 on page 458 if you could not re-create a table.) PSPI
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 on page 458 as both the translation source and the translation target
   b. The PSID that you recorded in step 1 on page 458 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 on page 458 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.
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 on page 461 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 IBM DB2 Recovery Expert for z/OS or IBM DB2 Log Analysis Tool for z/OS at this point in the recovery procedure.

   For more information about point-in-time recovery, see “Recovery of data to a prior point in time” on page 424.
8. Start the table space for normal use 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 "Page set and data set copies" on page 418 for more information about the COPY utility.
14. Re-create the objects that are dependent on the table.

    See step 11 on page 457 of "Recovering an accidentally dropped table" on page 456 for more information.

Related reference:

DSN1COPY (DB2 Utilities)

Recovering accidentally dropped user-managed data sets

You can recover dropped table spaces that are not part of the catalog. You need to copy the data sets that contain the data from the dropped table space to redefined data sets.

About this task

To copy the data sets, use the OBID-translate function of the DSN1COPY utility.

Procedure

To recover a dropped data set:

1. Find the DBID for the database, the PSID for the dropped table space, and the OBIDs for the tables that are contained in the dropped table space. For information about how to do this, see step 1 on page 456 of "Recovering an accidentally dropped table" on page 456.
2. Rename the data set that contains 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 the DSN1COPY utility to access the DB2 data sets, stop the table space by 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. PSPI

   Find the target identifiers of the objects that 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.

   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';
```

These values are returned in decimal format, which is the format that you need for the DSN1COPY utility.

PSPI

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 460 as both the translation source and the translation target
   b. The PSID that you recorded in step 1 on page 460 as the translation source and the PSID that you recorded in step 6 on page 460 as the translation target
   c. The original OBIDs that you recorded in step 1 on page 460 as the translation sources and the OBIDs that you recorded in step 6 on page 460 as the translation targets

   Be sure that you have named the VSAM data sets correctly by checking messages DSN1998I and DSN1997I after DSN1COPY completes.
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 use the IBM DB2 UDB Log Analysis Tool for z/OS.

   For more information about point-in-time recovery, see “Recovery of data to a prior point in time” on page 424.
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 “Page set and data set copies” on page 418 for more information about the COPY utility.
13. Re-create the objects that are dependent on the table.

    See step 11 on page 457 of “Recovering an accidentally dropped table” on page 456 for more information.

Related reference:

DSN1COPY (DB2 Utilities)

Recovering a DB2 system to a given point in time using the RESTORE SYSTEM utility

Use the RESTORE SYSTEM utility to recover your DB2 system to a given point in time. Recovering to a given point in time minimizes the amount of data that you lose when you recover.

Before you begin

The RESTORE SYSTEM utility uses system-level backups that contain only DB2 objects to restore your DB2 system to a given point in time. The following prerequisites apply:

- Before you can use the RESTORE SYSTEM utility, you must use the BACKUP SYSTEM utility to create system-level backups. Choose either DATA ONLY or FULL, depending on your recovery needs. Choose FULL if you want to backup both your DB2 data and your DB2 logs.
- When a system-level backup on tape is the input for the RESTORE SYSTEM utility, the user who submits the job must have the following two RACF authorities:
  - Operations authority, as in ATTRIBUTES=OPERATIONS
  - DASDVOL authority, which you can set in the following way:

    ```
    SETROPTS GENERIC(DASDVOL)
    REDEFINE DASDVOL * UACC(ALTER)
    SETROPTS CLASSACT(DASDVOL)
    SETROPTS GENERIC(DASDVOL) REFRESH
    ```

    You can restrict this authority to specific user IDs.

  This RACF authority is required, because the RESTORE SYSTEM utility invokes DFSMSdss when tape is the input. However, when you restore database copy pools from a FlashCopy on disk, the RESTORE SYSTEM utility invokes DFSMShsm, which does not require Operations or DASDVOL authority.

Procedure

To recover data to a given point in time:

1. Issue the STOP DB2 command to stop your DB2 system. If your system is a data sharing group, stop all members of the group.
2. If the backup is a full system backup, you might need to restore the log copy pool outside of DB2 by using DFSMShsm FRRECOV COPYPOOL (cpname) GENERATION (gen). For data-only system backups, skip this step.
3. Create a conditional restart record, where the SYSPITR option specifies the given point in time that you want to recover to. Run DSNJU003 (the change log inventory utility) with the CRESTART SYSPITR and SYSPITRT options, and specify 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.
4. For data sharing systems, delete all CF structures that the data sharing group owns.

5. Restore any logs on tape to disk.
6. Issue the START DB2 command to restart your DB2 system. For data sharing systems, start all active members.
7. Run the RESTORE SYSTEM utility. If you manually restored the backup, use the LOGONLY option of RESTORE SYSTEM to apply the current logs.
8. Stop and restart DB2 again to remove ACCESS(MAINT) status.

Results

After the RESTORE SYSTEM utility completes successfully, your DB2 system has been recovered to the given point in time with consistency.

Related concepts:

"Backup and recovery involving clone tables" on page 468

Related tasks:

"Recovering a DB2 subsystem to a prior point in time" on page 536

Related reference:

RESTORE SYSTEM (DB2 Utilities)

Related information:

Tape Authorization for DB2 RESTORE SYSTEM Utility

Recovering by using DB2 restart recovery

If you created a backup by using the BACKUP SYSTEM utility, you can recover your DB2 subsystem to the point in time of the backup by using normal DB2 restart recovery.

Procedure

To recover your DB2 system to the point in time of a backup:

1. Back up your system by issuing the BACKUP SYSTEM FULL command.

   DFSMShsm maintains up to 85 versions of system backups on disk at any given time.
2. Recover the system:
   a. Stop the DB2 subsystem. For data sharing systems, stop all members of the group.
   b. 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 to be restored.
   c. For data sharing systems, delete all CF structures that are owned by this group.
   d. Restore any logs on tape to disk.
   e. Start DB2. For data sharing systems, start all active members.
   f. 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.

Related concepts:

"Point-in-time recovery with system-level backups" on page 425

Recovering by using FlashCopy volume backups

You can use FlashCopy volume backups to recover your DB2 system to the point in time of a backup.

Procedure

To recover by using FlashCopy volume backups:

1. Back up your system:
   a. Issue the DB2 command SET LOG SUSPEND to suspend logging and update activity, and to quiesce 32 KB page writes and data set extensions. For data sharing systems, issue the command to each member of the group.
   b. 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.
   c. 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 that you just created to a lower-cost medium, such as tape.
2. Recover your system:
   a. Stop the DB2 subsystem. For data sharing systems, stop all members of the group.
   b. Use DFSMSdss RESTORE to restore the FlashCopy data sets to disk.
   c. For data sharing systems, delete all CF structures that are owned by this group.
   d. Start DB2. For data sharing systems, start all active members.
   e. 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.

Related information:

FlashCopy (DFSMS Advanced Copy Services)

Making catalog definitions consistent with your data after recovery to a prior point in time

Avoiding point-in-time recovery of the catalog is easier than attempting to correct the inconsistencies that this kind of recovery causes.

About this task

If you choose to recover catalog and directory tables to a prior point in time, you need to first shut down the DB2 subsystem cleanly and then restart in ACCESS(MAINT) mode before the recovery.

Procedure

To make catalog definitions consistent with your data after a point-in-time recovery:

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*.*y*0001.A00*n*

2. PSPI Execute the following SELECT statements to find a list of table space and table definitions in the DB2 catalog:

```
SELECT NAME, DBID, PSID FROM SYSIBM.SYSTABLESPACE;
SELECT NAME, TSNAME, DBID, OBID FROM SYSIBM.SYSTABLES;
```

PSPI

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. Check 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 statements 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.

      You can drop a LOB table space 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 by 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 DSN1COPY utility 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 is 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 "Recovering an accidentally dropped table space" on page 458 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.
6. Ensure that the IPREFIX values of user table spaces and index spaces that were reorganized match the IPREFIX value in the VSAM data set names that are associated with each table space or partition. If the IPREFIX that is 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.
   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.

      Important: For objects involved in cloning, rename the base and clone objects at the same time.

   Example: Assume that 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*.**I**0001.A001

   You must rename this data set to:

   catname.DSNDBC.*dbname*.*spname*.**J**0001.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.

Related concepts:

"Recovery of tables that contain identity columns" on page 449

Related reference:

DROP (DB2 SQL)

CREATE TABLESPACE (DB2 SQL)

DSN1COPY (DB2 Utilities)

DSN1PRNT (DB2 Utilities)

Recovery of catalog and directory tables

Recovering catalog and directory tables to a prior point in time is strongly discouraged for several 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 need to restore user databases; if you restore user databases, you might need 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 SHRLEVEL REFERENCE or SHRLEVEL CHANGE.
- 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.

Performing 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.

About this task

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 that you produce in some other way. For DB2 remote-site recovery procedures that do not use the RESTORE SYSTEM utility, see "Performing remote-site disaster recovery" on page 553.

Recovering with the BACKUP SYSTEM and RESTORE SYSTEM utilities

You can use the BACKUP SYSTEM and RESTORE SYSTEM utilities to recover your DB2 subsystem.

Procedure

To recover your DB2 subsystem:

1. Prepare for recovery:
   a. Use BACKUP SYSTEM FULL to take the system backup.
   b. Transport the system backups to the remote site.
2. Recover:
   a. Run the DSNJU003 utility using either of the following control statements:
      - In this control statement, substitute *log-truncation-point* with the RBA or LRSN of the point to which you want to recover

        ```
        CRESTART CREATE, SYSPITR=log-truncation-point
        ```

        where .
      - In this control statement, substitute *log-truncation-timestamp* with the timestamp of the point to which you want to recover.

```
CRESTART CREATE,SYSPITRT=log-truncation-timestamp
```

b. Start DB2.

c. Run the RESTORE SYSTEM utility by issuing the `RESTORE SYSTEM` control statement.

   This utility control statement performs a recovery to the current time (or to the time of the last log transmission from the local site).

Recovering without using the BACKUP SYSTEM utility

You can recover your DB2 subsystem if you do not use the BACKUP SYSTEM utility to produce backups.

Procedure

To recover your DB2 subsystem without using the BACKUP SYSTEM utility:

1. Prepare for recovery.
   a. Issue the DB2 command SET LOG SUSPEND to suspend logging and update activity, and to quiesce 32 KB page writes and data set extensions.

      For data sharing systems, issue the command to each member of the data sharing group.

   b. 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.

   c. Issue the DB2 command SET LOG RESUME to resume normal DB2 activity.

   d. 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 that you make to the remote site.

2. Recover your DB2 subsystem.
   a. Use DFSMSdss to restore the FlashCopy data sets to disk.

   b. Run the DSNJU003 utility by using the CRESTART CREATE, SYSPITR=*log-truncation-point* control statement.

      The *log-truncation-point* is the RBA or LRSN of the point to which you want to recover.

   c. Restore any logs on tape to disk.

   d. Start DB2.

   e. Run the RESTORE SYSTEM utility using the RESTORE SYSTEM LOGONLY control statement to recover to the current time (or to the time of the last log transmission from the local site).

Backup and recovery involving clone tables

When you recover a clone table that has been exchanged, you can use an image copy that was made prior to an exchange. However, no point-in-time recovery is possible prior to the most recent exchange.

Introductory concepts:

Types of tables (Introduction to DB2 for z/OS)

Clone tables spaces and index spaces are stored in separate physical data sets. You must copy them and recover them separately. Output from the REPORT utility includes information about clone tables, if they exist.

The QUIESCE command and the COPY and RECOVER utilities each use the CLONE keyword to function on clone objects.

- Running QUIESCE with the CLONE keyword establishes a quiesce point for a clone object.
- Running the COPY utility with the CLONE keyword takes a copy of a clone object.
- Running the RECOVER utility with the CLONE keyword recovers a clone object.

Recovery of temporal tables with system-period data versioning

You must recover a system-period temporal table that is defined with system-period data versioning and its corresponding history table, as a set, to a single point in time. You can recover the table spaces individually only if you specify the VERIFYSET NO option in the RECOVER utility statement.

If you specify TOCOPY, TOLASTCOPY, or TOLASTFULLCOPY in the RECOVER utility statement, you cannot explicitly specify the VERIFYSET option. However, in this case, VERIFYSET NO is the default.

If you recover a system-period temporal table with system-period data versioning to a point in time before the table was altered to have system-period data versioning, the table is still defined with system-period data versioning, but the history table is empty.

Related concepts:

"Temporal tables" on page 53

Data restore of an entire system

The RESTORE SYSTEM utility invokes z/OS DFSMShsm services to recover a DB2 subsystem to a prior point in time.

The RESTORE SYSTEM utility restores the databases in the volume copies that the BACKUP SYSTEM utility provided. After restoring the data, the RESTORE SYSTEM utility can then recover a DB2 subsystem to a given 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 the RESTORE SYSTEM utility.

The SYSPITRT option of DSNJU003 CRESTART allows you to add a timestamp, in the ENDTIME format, to truncate logs for system point-in-time recovery restart.

You can specify a value of 'FFFFFFFFFFFF' to cause system point-in-time recovery to occur without log truncation.

Related reference:

DSNJU003 (change log inventory) (DB2 Utilities)

RESTORE SYSTEM (DB2 Utilities)

Chapter 13. Recovering from different DB2 for z/OS problems

You can troubleshoot and recover from many DB2 problems on your own by using the provided recovery procedures.

Recovering from IRLM failure

You can recover from an IRLM failure, regardless of whether the failure results in a wait, loop, or abend.

Symptoms

The IRLM waits, loops, or abends. The following message might be issued:

```
DXR122E irlmnm ABEND UNDER IRLM TCB/SRB IN MODULE xxxxxxxx
ABEND CODE zzzz
```

Environment

If the IRLM abends, DB2 terminates. If the IRLM waits or loops, the IRLM terminates, and DB2 terminates automatically.

Resolving the problem

Operator response:

1. Start the IRLM if you did not set it for automatic start when you installed DB2.
2. Start DB2.
3. Connect IMS to DB2, by issuing the following command, where *ssid* is the subsystem ID:

   ```
   /START SUBSYS ssid
   ```
4. Connect CICS to DB2 by issuing the following command:

   ```
   DSNC STRT
   ```

   Related tasks

   Connecting from CICS

   Starting DB2

   Starting the IRLM

Recovering from z/OS or power failure

You can recover from a situation in which z/OS or your processor power fails.

Symptoms

No processing is occurring.

Resolving the problem

Operator response:

- If the power failure or z/OS failure has occurred:
  1. IPL z/OS, and initialize the job entry subsystem (JES).
  2. If you normally run VTAM with DB2, start VTAM at this point.
  3. Start the IRLM if it was not set for automatic start during DB2 installation.
  4. Start DB2.
  5. Use the **RECOVER POSTPONED** command if postponed-abort units of recovery were reported after restarting DB2, and if the AUTO or LIGHTAUTO option

© Copyright IBM Corp. 1982, 2016

of the LIMIT BACKOUT field on installation panel DSNTIPL was not specified. If the LIGHTAUTO option is specified, postponed-abort units of recovery are processed during the next normal DB2 restart.

6. Restart IMS or CICS.
   - IMS automatically connects and resynchronizes when it is restarted.
   - CICS automatically connects to DB2 if the CICS PLT contains an entry for the attachment facility module DSNCCOM0. Alternatively, use the command **DSNC STRT** to connect the CICS attachment facility to DB2.

- If you know that a power failure is imminent, issue a **STOP DB2 MODE(FORCE)** command to allow DB2 to stop 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 operational.

Related concepts

Connections to the IMS control region

Related tasks

Connecting from CICS

Starting DB2

Starting the IRLM

Recovering from disk failure

When a disk hardware failure occurs and an entire unit is lost, you can recover from this situation.

Symptoms

No I/O activity occurs for the affected disk address. Databases and tables that reside on the affected unit are unavailable.

Resolving the problem

Operator response:

1. Assure that no incomplete I/O requests exist for the failing device. One way to do this is to force the volume offline by issuing the following z/OS command, where *xxx* is the unit address:

   ```
   VARY xxx,OFFLINE,FORCE
   ```

   To check disk status, issue the following command:

   ```
   D U,DASD,ONLINE
   ```

   The following console message is displayed after you force 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 previously set the I/O timing interval for the device class, the I/O timing facility terminates all requests that are incomplete at the end of the specified time interval, and you can proceed to the next step without varying the volume offline. You can set the I/O timing interval either through the IECIOSxx z/OS parameter library member or by issuing the following z/OS command:

   ```
   SETIOS MIH,DEV=devnum,IOTIMING=mm:ss.
   ```

2. Issue (or request that an authorized operator issue) the following DB2 command to stop all databases and table spaces that reside on the affected volume:

```
-STOP DATABASE(database-name) SPACENAM(space-name)
```

If the disk unit must be disconnected for repair, stop all databases and table spaces on all volumes in the disk unit.

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 volume serial number (VOLSER).

```
// Job
//ICKDSF   EXEC PGM=ICKDSF
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   *
     REVAL UNITADDRESS(yyy) VERIFY(volser)
```

If you initialize a 3380 or 3390 volume, use **REVAL** with the VERIFY parameter to ensure that you initialize the intended volume, or to revalidate the home address of the volume and record 0. Alternatively, use ISMF to initialize the disk unit.

4. Issue the following z/OS console command, where *yyy* is the new unit address:

```
VARY yyy,ONLINE
```

5. To check disk status, issue the following command:

```
D U,DASD,ONLINE
```

The following console message is displayed:

```
UNIT  TYPE  STATUS   VOLSER  VOLSTATE
7D4   3390  O        XTRA02  PRIV/RSDNT
```

6. 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, where *y* is either I or J:

```
DELETE catnam.DSNDBC.dbname.tsname.y0001.Annn CLUSTER NOSCRATCH
```

where *nnn* is the data set or partition number, left padded by 0 (zero).

7. For user-managed table spaces, define the VSAM cluster and data components for the new volume by issuing the access method services **DEFINE CLUSTER** command with the same data set name as in the previous step, in the following format:

```
catnam.DSNDBx.dbname.tsname.y0001.znnn
```

The *y* is I or J, the *x* is C (for VSAM clusters) or D (for VSAM data components), and *znnn* is the data set or partition number, left padded by 0 (zero). For more information, see Data set naming conventions.

8. For a user-defined table space, define the new data set before an attempt to recover it. You can recover table spaces that are defined in storage groups without prior definition.
9. Issue the following DB2 command to start all the appropriate databases and table spaces that were previously stopped:

```
-START DATABASE(database-name) SPACENAM(space-name)
```

10. Recover the table spaces by using the DB2 RECOVER utility.

Related reference

RECOVER (DB2 Utilities)

Related information

DFSMS Access Method Services Commands

Device Support Facilities (ICKDSF) Device Support Facilities (ICKDSF) User's Guide and Reference

z/OS MVS System Commands

z/OS MVS Initialization and Tuning Reference

Recovering from application errors

You can recover from a problem in which an application program placed a logically incorrect value in a table.

Symptoms

Unexpected data is returned from an SQL SELECT statement, even though the SQLCODE that is associated with the statement is 0.

Causes

An SQLCODE of 0 indicates that DB2 and SQL did not cause the problem, so the cause of the incorrect data in the table is the application.

Resolving the problem

System programmer response: You might be able to use the DB2 RECOVER utility with the TOLOGPOINT option to restore the database to a point before the error occurred. However, in many circumstances you must manually back out the changes that were introduced by the application. Among those circumstances are:

- Other applications changed the database after the error occurred. If you recover the table spaces that were modified by the bad application, all subsequent changes that were made by the other applications are lost.
- 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. However, all subsequent changes to the database are lost.

If you have a situation for which using RECOVER TOLOGPOINT is appropriate, you can use one of the following procedures as a basis for backing out the incorrect changes that were made by the application. The procedure that you use depends on whether you have established a quiesce point.

Backing out incorrect application changes (with a quiesce point)

If you have an established quiesce point, you can back out incorrect changes that your application made.

Procedure

To back out the incorrect changes:

1. Run the REPORT utility twice, once using the RECOVERY option and once using the TABLESPACESET option. On each run, specify the table space that contains 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. Run RECOVER TOLOGPOINT with the RBA that you found, specifying the names of all related table spaces.

Results

Recovering all related table spaces to the same quiesce point prevents violations of referential constraints.

Backing out incorrect application changes (without a quiesce point)

Even if you do not have an established quiesce point, you can back out incorrect changes that your application made. Be aware, however, that if you use this procedure, you lose any updates to the database that occurred after the last checkpoint and before the application error occurred.

Procedure

To back out the incorrect changes:

1. Run the DSN1LOGP stand-alone utility on the log scope that is available at DB2 restart, using the SUMMARY(ONLY) option.
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, which looks similar to this message:

     ```
     DSNR003I RESTART  ..... PRIOR CHECKPOINT RBA=000007425468
     ```

     The required RBA in this example is X'7425468'.

     This technique works only if no checkpoints have been taken 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, and an identification of the databases that are affected by each active user.
4. Find the unit of recovery in which the error was made. 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 that 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. Run the DB2 RECOVER utility with the TOLOGPOINT option, and specify the starting RBA that you found in the previous step.
6. Recover any related table spaces or indexes to the same point in time.

Related concepts

DSN1LOGP summary report

Related reference

DSN1LOGP (DB2 Utilities)

Recovering from IMS-related failures

When you work in a DB2-IMS environment and problems occur, you can recover from those problems.

Symptoms

Problems that occur in a DB2-IMS environment can result in a variety of symptoms:

- An IMS wait, loop, or abend is accompanied by a DB2 message that goes to the IMS console. This symptom indicates an IMS control region failure.

- When IMS connects to DB2, DB2 detects one or more units of recovery that are indoubt.
- When IMS connects to DB2, DB2 detects that it has committed one or more units of recovery that IMS indicates should be rolled back.
- Messages are issued to the IMS master terminal, to the logical terminal, or both to indicate that some sort of IMS or DB2 abend has occurred.

Environment

DB2 can be used in an XRF (Extended Recovery Facility) recovery environment with IMS.

To resolve IMS-related problems, follow the appropriate procedure.

Related concepts

Plans for extended recovery facility toleration

Recovering from IMS control region failure

You can recover from a problem in which the IMS control region fails.

Symptoms

- IMS waits, loops, or abends.
- DB2 attempts to send the following message to the IMS master terminal during an abend:

  `DSNM002 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 for this problem to the z/OS console.

Environment

- DB2 detects that IMS has failed.
- DB2 either backs out or commits work that is in process.
- DB2 saves indoubt units of recovery, which need to be resolved at reconnection time.

Resolving the problem

Operator response: Use normal IMS restart procedures, which include starting IMS by issuing the z/OS **START IMS** command. The following results occur:

1. All DL/I and DB2 updates that have not been committed are backed out.
2. IMS is automatically reconnected to DB2.
3. IMS passes the recovery information for each entry to DB2 through the IMS attachment facility. (IMS indicates whether to commit or roll back.)
4. DB2 resolves the entries according to IMS instructions.

Recovering from IMS indoubt units of recovery

When IMS connects to DB2, and DB2 has indoubt units of recovery that have not been resolved, these units of recovery need to be resolved.

Symptoms

If DB2 has indoubt units of recovery that IMS did not resolve, the following message is issued at the IMS master terminal, where *xxxx* is the subsystem identifier:

`DSNM004I RESOLVE INDOUBT ENTRY(S) ARE OUTSTANDING FOR SUBSYSTEM xxxx`

Causes

When this message is issued, IMS was either cold started, or it was started with an incomplete log tape. Message DSNM004I might also be issued if DB2 or IMS abnormally terminated in response to a software error or other subsystem failure.

Environment

- 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 might start to back up. If the IMS queues fill to capacity, IMS terminates. Be aware of this potential difficulty, and monitor IMS until the indoubt units of work are fully resolved.

Resolving the problem

System programmer response:

1. Force the IMS log closed by using the **/DBR FEOV** command.
2. Archive the IMS log.
3. Issue the command **DFSERA10** to print the records from the previous IMS log tape for the last transaction that was processed in each dependent region. Record the PSB and the commit status from the X'37' log that contains the recovery ID.
4. Run the DL/I batch job to back out each PSB that is involved that has not reached a commit point. The process might be time-consuming because transactions are still being processed. This process might also lock a number of records, which could affect the rest of the processing and the rest of the message queues.
5. Enter the DB2 command **DISPLAY THREAD (***imsid*) TYPE (INDOUBT).
6. Compare the NIDs (IMSID + OASN in hexadecimal) that are displayed in the **DISPLAY THREAD** output with the OASNs (4 bytes decimal) as shown in the DFSERA10 output. Decide whether to commit or roll back.
7. 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, for use during the next step.
8. GUPI

   Enter the following DB2 command, choosing to commit or roll back, and specify the correlation ID:

   ```
   -RECOVER INDOUBT (imsid) ACTION(COMMIT|ABORT) NID (nid)
   ```

   If the command is rejected because of associated network IDs, use the same command again, substituting the recovery ID for the network ID.

GUPI

Related concepts

Duplicate IMS correlation IDs

Recovering IMS indoubt units of work that need to be rolled back

When units of recovery between IMS and DB2 are indoubt at restart time, DB2 and IMS sometimes handle the indoubt units of recovery differently. When this situation happens, you might need to roll back the changes.

Symptoms

The following messages are issued after a DB2 restart:

```
DSNM005I  IMS/TM RESOLVE INDOUBT PROTOCOL PROBLEM WITH SUBSYSTEM xxxx

DFS3602I xxxx SUBSYSTEM RESOLVE-INDOUBT FAILURE,RC=yyyy
```

Causes

The reason that these messages are issued is that indoubt units of work exist for a DB2-IMS application, and the way that DB2 and IMS handle these units of work differs.

At restart time, DB2 attempts to resolve any units of work that are indoubt. DB2 might commit some units and roll back others. DB2 records the actions that it takes for the indoubt units of work. At the next connect time, DB2 verifies that the actions that it took are consistent with the IMS decisions. If the DB2 **RECOVER INDOUBT** command is issued prior to an IMS attempt to reconnect, DB2 might decide to commit the indoubt units of recovery, whereas IMS might decide to roll back the units of recovery. This inconsistency results in the DSNM005I message being issued. Because DB2 tells IMS to retain the inconsistent entries, the DFS3602I message is issued when the attempt to resolve the indoubt units of recovery ends.

Environment

- 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.

Resolving the problem

System programmer response: Do not use the DB2 **RECOVER INDOUBT** command. The problem is that DB2 was not indoubt but should have been. Database updates have probably been committed on one side (IMS or DB2) and rolled back on the other side.

1. Enter the IMS command **/DISPLAY OASN SUBSYS DB2** to display the IMS list of units of recovery that need to be resolved. This command generates the list of OASNs in a decimal format, not in 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 that were recorded at the time of failure and during restart. Look at the X'37', X'56', and X'5501FE' records at reconnect time. Notify IBM Software Support about the problem.
4. Determine what the inconsistent unit of recovery was doing by examining the log information, and manually make the IMS and DB2 databases consistent.

Related concepts

Duplicate IMS correlation IDs

Recovering from IMS application failure

You can recover from a situation in which an IMS application abnormally terminates in a DB2 environment.

Symptoms

The following messages are issued at the IMS master terminal and at the LTERM that entered the transaction that is 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
```

Causes

The problem might be caused by a usage error in the application or by a DB2 problem.

Environment

- The failing unit of recovery is backed out by both DL/I and DB2.
- The connection between IMS and DB2 remains active.

Resolving the problem

Operator response:

- If you think that the problem was caused by a usage error, investigate and resolve the error.
- If you think that the problem is a DB2 problem, rather than a usage error, try to diagnose the problem using standard diagnostic procedures. You might need to contact IBM Software Support if you cannot resolve the problem yourself.

Related concepts

Obtaining DB2 Diagnosis Guide and Reference ()

Techniques for debugging programs in IMS (DB2 Application programming and SQL)

Recovering from a DB2 failure in an IMS environment

When DB2 fails in a DB2-IMS environment, you can recover from this situation.

Symptoms

DB2 fails or is not running, and 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 either of these situations, the IMS master terminal receives IMS message DFS554, and the terminal that is involved in the problem receives IMS message DFS555.

Resolving the problem

Operator response:

1. Restart DB2.
2. Follow the standard IMS procedures for handling application abends.

Recovering from CICS-related failure

When you work in a DB2-CICS environment and problems occur, you can recover from those problems.

Symptoms

Problems that occur in a DB2-CICS environment can result in a variety of symptoms, such as:

- Messages that indicate an abend in CICS or the CICS attachment facility
- A CICS wait or a loop
- Indoubt units of recovery between CICS and DB2

Environment

DB2 can be used in an XRF (Extended Recovery Facility) recovery environment with CICS.

Resolving the problem

To resolve CICS-related problems, follow the appropriate procedure.

Related concepts

Plans for extended recovery facility toleration

Recovering from CICS application failures

You can recover from a CICS application abend in a DB2 environment.

Symptoms

The following message is issued at the user's terminal:

`DFH2206 TRANSACTION tranid ABEND abcode BACKOUT SUCCESSFUL`

In this message, *tranid* represents the transaction that abnormally terminated, and *abcode* represents the specific abend code.

Environment

- The failing unit of recovery is backed out in both CICS and DB2.
- The connection between CICS and DB2 remains active.

Resolving the problem

Operator response: Investigate the abend by reading about the abend code.

- For an AEY9 abend, start the CICS attachment facility.
- For an ASP7 abend, determine why the CICS **SYNCPOINT** was unsuccessful.
- For other abends, follow appropriate diagnostic procedures.

Related concepts

Obtaining DB2 Diagnosis Guide and Reference ()

CICS Transaction Server for z/OS troubleshooting and support

Recovering DB2 when CICS is not operational

You can recover DB2 from a situation in which CICS is not operational.

Symptoms

Any of the following symptoms might occur:

- CICS waits or loops.
- CICS abends, as indicated by messages or dump output.

Environment

DB2 performs each of the following actions:

- Detects the CICS failure.
- Backs out inflight work.

- Saves indoubt units of recovery that need to be resolved when CICS is reconnected.

Diagnosing the problem

If you think that CICS is in a wait or loop situation, find the origin of the wait or loop. The origin might be in CICS, in CICS applications, or in the CICS attachment facility.

If you receive messages that indicate a CICS abend, examine the messages and dump output 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 cleaned up and disconnected any connected threads.

Resolving the problem

Operator response:

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. The CICS attachment facility 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

Related tasks

Connecting from CICS

Related information

Troubleshooting for CICS DB2 (CICS DB2 Guide)

Recovering DB2 when the CICS attachment facility cannot connect to DB2

You can recover DB2 when the CICS attachment facility cannot connect to DB2.

Symptoms

Any of the possible symptoms can occur:

- CICS remains operational, but the CICS attachment facility abends.
- The CICS attachment facility issues a message that indicates the reason for the connection failure, or it requests a X'04E' dump.
- The reason code in the X'04E' dump indicates the reason for failure.
- CICS issues message DFH2206 that indicates that the CICS attachment facility has terminated abnormally with the DSNC abend code.
- CICS application programs that try to access DB2 while the CICS attachment facility is inactive are abnormally terminated. The code AEY9 is issued.

Environment

CICS backs out the abnormally terminated transaction and treats it like an application abend.

Resolving the problem

Operator response: Start the CICS attachment facility by entering the appropriate command for your release of CICS. After you start the CICS attachment facility, the following events occur:

1. The CICS attachment facility initializes and reconnects to DB2.
2. The CICS attachment facility requests information about the indoubt units of recovery and passes the information to CICS.
3. CICS resolves the indoubt units of recovery.

Recovering CICS indoubt units of recovery

When the CICS attachment facility abends, CICS and DB2 build lists of indoubt units of work, either dynamically or during restart, depending on the failing subsystem. If any units of recovery are indoubt at connect time, you can recover from this situation.

Symptoms

One of the following messages is sent to the user-named CICS destination that is specified for the MSGQUEUE*n*(*name*) attribute in the RDO (resource definition online): DSN2001I, DSN2034I, DSN2035I, or DSN2036I.

Causes

For CICS, a DB2 unit of recovery might be indoubt if the forget entry (X'FD59') of the task-related installation exit routine is absent from the CICS system journal. The indoubt condition applies only to the DB2 unit of recovery in this case because CICS 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.

Environment

The following table summarizes the situations that can exist when CICS units of recovery are indoubt.

GUPI

Table 37. Situations that involve CICS abnormal indoubt units of recovery

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 it is indoubt according to CICS log information. The reason is probably a CICS restart with the wrong tape. The problem might also be caused by a DB2 restart to a prior point in time.
DSN2035I	The named unit of recovery is indoubt for DB2, but it is not in the CICS indoubt list. This is probably due to an incorrect CICS restart. The CICS attachment facility continues the startup process and provides a transaction dump. The problem might also be caused by a DB2 restart to a prior point in time.

Table 37. Situations that involve CICS abnormal indoubt units of recovery (continued)

Message ID	Meaning
DSN2036I	CICS indicates rollback for the named unit of recovery, but DB2 has already committed the unit of recovery. The CICS attachment facility continues the startup process.

GUPI

CICS retains details of indoubt units of recovery that were not resolved during connection startup. An entry is purged when it no longer shows up on the list that is presented by DB2 or, when the entry is present in the list, when DB2 resolves it.

Resolving the problem

System programmer response: If CICS cannot resolve one or more indoubt units of recovery, resolve them manually by using DB2 commands. Using the steps in this procedure is rarely necessary because it is required only where operational errors or software problems have prevented automatic resolution.

1.

GUPI

Obtain a list of the indoubt units of recovery from DB2 by issuing the following command:

```
-DISPLAY THREAD (connection-name) TYPE (INDOUBT)
```

Messages like these are then issued:

```
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 1.2 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

GUPI

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, use the network ID (NID) 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 that is provided by CICS at the time that the syncpoint log entries are written. This unique number is an 8-byte store clock value that is stored in records that are 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*.

2. Scan the CICS log for entries that are related to a particular unit of recovery. Look for a PREPARE record (JCRSTRID X'F959'), for the task-related installation

where the recovery token field (JCSRMTKN) equals the value that is obtained from the network-ID. The network ID is supplied by DB2 in the **DISPLAY THREAD** command output.

You can find the CICS task number by locating the prepare log record in the CICS log for indoubt units of recovery. Using the CICS task number, you can locate all other entries on the log for this CICS task.

You can use the CICS journal print utility DFHJUP to scan the log.

3. Use the change log inventory utility (DSNJU003) to scan the DB2 log for entries that are related to a particular unit of recovery. Locate the End Phase 1 record with the required network ID. 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 startup message DSNJ099I provides the start log RBA for this session.

4. If needed, do indoubt resolution in DB2.

   GUPI

   To invoke DB2 to take the recovery action for an indoubt unit of recovery, issue the DB2 **RECOVER INDOUBT** command, where the *correlation_id* is unique:

   ```
   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*) that is returned by **DISPLAY THREAD** for *thread#.tranid* in the **RECOVER INDOUBT** command. 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. The *groupname* is the first transaction that is 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 that are associated with *connection-name*, omit the ID option.

   The command results that are in either of the following messages indicate whether the thread is committed or rolled back:

   ```
   DSNV414I - THREAD thread#.tranid COMMIT SCHEDULED
   DSNV414I - THREAD thread#.tranid ABORT SCHEDULED
   ```

   When you resolve indoubt units of work, note that CICS and the CICS 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 presented list is empty or when the list does not include a unit of recovery that CICS remembers.

   Investigate any inconsistencies that you found in the preceding steps.

   GUPI

Related reference

DSNJU003 (change log inventory) (DB2 Utilities)

Related information

Reading log streams using batch jobs (for example, DFHJUP) (CICS Transaction Server for z/OS)

Recovering from CICS attachment facility failure

You can recover DB2 when the CICS attachment facility abends or when a CICS attachment thread subtask abends.

Symptoms

The symptoms depend on whether the CICS attachment facility or one of its thread subtasks terminated:

- If the main CICS attachment facility 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, a X'04E' dump is issued, and the CICS application abends with a DSNC dump code. The X'04E' dump generally indicates the cause of the abend. The CICS attachment facility remains active.

Resolving the problem

Operator response: Correct the problem that caused the abend by analyzing the CICS formatted transaction dump or subtask SNAP dump. If the CICS attachment facility shuts down, use CICS commands to stop the execution of any CICS-DB2 applications.

Recovering from a QMF query failure

Receipt of a -805 SQL code in a DB2 for z/OS environment that includes QMF might occur after you start QMF or issue a QMF command. If the DB2 subsystem was recently migrated to a new release or to new-function mode, you might need to rerun certain QMF installation jobs.

Symptoms

One of the following QMF messages is issued:

- DSQ10202
- DSQ10205
- DSQ11205
- DSQ12105
- DSQ13005
- DSQ14152
- DSQ14153
- DSQ14154
- DSQ15805
- DSQ16805
- DSQ17805
- DSQ22889
- DSQ30805
- DSQ31805
- DSQ32029
- DSQ35805

- DSQ36805

Causes
Key QMF installation jobs were not run.

Environment
The DB2 for z/OS subsystem was migrated to a new release or to new-function mode after completion of an installation or migration to a new release of QMF.

Diagnosing the problem
User response:

- If your DB2 for z/OS subsystem was recently migrated to a new release or to new-function mode, continue with "Resolving the problem."
- If your DB2 for z/OS subsystem was not recently migrated, see other possible causes of the DSQxxxxx messages in Troubleshooting and correcting QMF bind problems associated with a -805 SQL code.

Resolving the problem
User response: Rerun QMF installation jobs as described in Tasks to perform when you upgrade DB2 for z/OS after you install QMF.

Recovering from subsystem termination

You can recover DB2 after DB2 or an operator-issued cancel causes the subsystem to terminate.

Symptoms
When a DB2 subsystem terminates, the specific failure is identified in one or messages. The following messages might be issued at the z/OS console:

```
DSNV086E - DB2 ABNORMAL TERMINATION REASON=XXXXXXXX
DSN3104I - DSN3EC00 -TERMINATION COMPLETE
DSN3100I - DSN3EC00 - SUBSYSTEM ssnm READY FOR -START COMMAND
```

The following message might be issued to the IMS master terminal:

```
DSNM002I   IMS/TM xxxx DISCONNECTED FROM SUBSYSTEM
      yyyy RC=rc
```

The following message might be issued to the CICS transient data error destination, which is defined in the RDO:

```
DSNC2025I  - THE ATTACHMENT FACILITY IS INACTIVE
```

Environment
- IMS and CICS continue.
- In-process IMS and CICS 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'.
- IMS applications that begin to run after subsystem termination begins 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.
- CICS applications that begin to run after subsystem termination begins 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).

Resolving the problem

Operator response:

1. Restart DB2 by issuing the command **START DB2**.
2. For IMS environments, re-establish the IMS connection by issuing the IMS command **/START SUBSYS DB2**.
3. For CICS environments, re-establish the CICS connection by issuing the CICS attachment facility command **DSNC STRT**.

Recovering from temporary resource failure

DB2 sometimes experiences a temporary problem when it accesses log data sets. In this case, you need to recover from the situation so that processing can continue as normal.

Symptoms

DB2 issues messages for the access failure for each log data set. These messages provide information that is needed to resolve the access error. For example:

```
DSNJ104I  ( DSNJR206 RECEIVED ERROR STATUS 00000004
            FROM DSNPCLOC FOR DSNAME=DSNC710.ARCHLOG1.A0000049

*DSNJ153E ( DSNJR006 CRITICAL LOG READ ERROR
            CONNECTION-ID = TEST0001
            CORRELATION-ID = CTHDCORID001
            LUWID = V71A.SYEC1DB2.B3943707629D=10
            REASON-CODE = 00D10345
```

Causes

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 canceled

Resolving the problem

User response: You can attempt to recover from temporary failures by issuing a positive reply (Y) to the following 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.

Recovering from active log failures

A variety of active log failures might occur, but you can recover from them.

Symptoms

Most active log failures are accompanied by or preceded by error messages to inform you of out-of-space conditions, write or read I/O errors, or loss of dual active logging.

If you receive message DSNJ103I at startup time, the active log is experiencing dynamic allocation problems. If you receive message DSNJ104I, the active log is experiencing open-close problems. In either case, you should follow procedures in "Recovering from BSDS or log failures during restart" on page 500.

Recovering from being out of space in active logs

The available space in the active log is finite, so the active log might fill to capacity for one of several reasons. For example, delays in offloading and excessive logging can fill the active log. You can recover from out-of-space conditions in the active log.

Symptoms

The following warning message is issued when the last available active log data set is 5% full:

```
DSNJ110E - LAST COPY n ACTIVE LOG DATA SET IS nnn PERCENT FULL
```

The DB2 subsystem 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.

If the active log fills to capacity, after having switched to single logging, DB2 issues the following message, and an offload is started.

```
DSNJ111E - OUT OF SPACE IN ACTIVE LOG DATA SETS
```

The DB2 subsystem then halts processing until an offload is completed.

Causes

The active log is out of space.

Environment

An out-of-space condition on the active log has very serious consequences. Corrective action is required before DB2 can continue processing. When the active log becomes full, the DB2 subsystem cannot do any work that requires writing to the log until an offload is completed. Until that offload is completed, DB2 waits for an available active log data set before resuming normal DB2 processing. Normal shutdown, with either a **QUIESCE** or **FORCE** command, is not possible because the shutdown sequence requires log space to record system events that are related to shutdown (for example, checkpoint records).

Resolving the problem

Operator response:

1. Ensure that the offload is not waiting for a tape drive. If it is, mount a tape. DB2 then processes the offload task.
2. 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. Issuing this command might solve the problem.

3. If issuing this command does not solve the problem, determine and resolve the cause of the problem, and then reissue the command. If the problem cannot be resolved quickly, have the system programmer define additional active logs until you can resolve the problem.

System programmer response: Define additional active log data sets so that DB2 can continue its normal operation while the problem that is causing the offload failures is corrected.

1. Use the z/OS command **CANCEL** to stop DB2.
2. Use the access method services **DEFINE** command to define new active log data sets.
3. Run utility DSNJLOGF to initialize the new active log data sets.
4. Define the new active log data sets in the BSDS by using the change log inventory utility (DSNJU003). .
5. Restart DB2. Offload is started automatically during startup, and restart processing occurs.

Recommendation: To minimize the number of offloads that are taken per day in your installation, consider increasing the size of the active log data sets.

Related reference

DSNJU003 (change log inventory) (DB2 Utilities)

Recovering from a write I/O error on an active log data set

You can recover from a situation in which a write error occurs on an active log data set.

Symptoms

The following message is issued:

```
DSNJ105I - csect-name LOG WRITE ERROR DSNAME=..., LOGRBA=...,
           ERROR STATUS= ccccffss
```

Causes

Although this problem can be caused by several problems, one possible cause is a CATUPDT failure.

Environment

When a write error occurs on an active log data set, the following characteristics apply:

- DB2 marks the failing DB2 log data set TRUNCATED in the BSDS.
- DB2 goes on to the next available data set.
- If dual active logging is used, DB2 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 a DSNJ104 message indicates a CATUPDT failure, the data set is marked STOPPED.

Resolving the problem

System programmer response: If the DSNJ104 message indicates a CATUPDT failure, use access method services and the change log inventory utility

(DSNJU003) to add a replacement data set. In this case, you need to stop DB2. The timing of when you should take this action depends on how widespread the problem is.

- If the additional problem is localized and does not affect your ability to recover from any other problems, you can wait until the earliest convenient time.
- If the problem is widespread (perhaps affecting an entire set of active log data sets), stop DB2 after the next offload.

Related reference

DSNJU003 (change log inventory) (DB2 Utilities)

Recovering from a loss of dual active logging

If you use dual active logs, which is generally recommended, and one of the active log fails, DB2 reverts to use of a single active log. You can recover from this situation and return to dual active-log mode.

Symptoms

The following message is issued:

```
DSNJ004I - ACTIVE LOG COPY n INACTIVE, LOG IN SINGLE MODE,
       ENDRBA=...
```

Causes

This problem occurs when DB2 completes one active log data set and then finds that the subsequent copy (COPY *n*) data sets have not been offloaded and are marked STOPPED.

Environment

DB2 continues in single mode until offloading completes and then returns to dual mode. If the data set is marked STOPPED, however, intervention is required.

Resolving the problem

System programmer response:

1. Verify that offload is proceeding and is not waiting for a tape mount. You might need to run the DB2 print log map utility (DSNJU004) to determine the status of all data sets.
2. If any data sets are marked STOPPED, use IDCAMS to delete the data sets, and then re-add them by using the DB2 change log inventory utility (DSNJU003).

Related reference

DSNJU003 (change log inventory) (DB2 Utilities)

Recovering from I/O errors while reading the active log

You can recover from situations in which an I/O error occurs when DB2 is reading the active log.

Symptoms

The following message is issued:

```
DSNJ106I - LOG READ ERROR DSNAME=..., LOGRBA=...,
           ERROR STATUS=ccccffss
```

Environment

- If the error occurs during offload, offload tries to identify the RBA range from a second copy of the active log.
  - If no second copy of the active log exists, the data set is stopped.

- If the second copy of the active log 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 of the active log is satisfactory, the first copy is not stopped.

- If the error occurs during recovery, DB2 provides data from specific log RBAs that are 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 that is being recovered is not accessible.

Resolving the problem

System programmer response:

- 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.
- You can use the IDCAMS **REPRO** command 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).
- If you need to replace the data set:
  1. Ensure that the data is saved.

     If you have dual active logs, the data is saved on the other active log, which becomes your new data set. Skip to step 4 on page 492.

     If you are not 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 utility (DSNJU004) 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 utility output is greater than the RBA range of the data set with the error), manually add a new archive log to the BSDS by using the change log inventory utility (DSNJU003). Use IDCAMS to define a new log that has the same LRECL and BLKSIZE values as 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, although the data still exists and is not overlaid. If this occurs, see "Recovering from BSDS or log failures during restart" on page 500.

4. Stop DB2, and use the change log inventory utility 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 **DELETE** before **NEWLOG** in the SYSIN input data set. This step clears the stopped status, and DB2 eventually archives it.
   c. Delete the data set that is in error by using access method services.
   d. Redefine the data set so that you can write to it. Use the access method services **DEFINE** command to define the active log data sets. If you use dual logs and have a good copy of the log, use the **REPRO** command to copy the contents to the new data set that you just created. If you do not use dual logs, initialize the new data set by using the DSNJLOGF utility.

Related reference

PRIMARY QUANTITY field (PRIQTY subsystem parameter) (DB2 Installation and Migration)

DSNJU004 (print log map) (DB2 Utilities)

Recovering from archive log failures

You can recover from situations in which archive logging fails.

Symptoms

Archive log failures can result in a variety of DB2 and z/OS messages that identify problems with archive log data sets.

One specific symptom that might occur is message DSNJ104I, which indicates an open-close problem on the archive log.

Recovering from allocation problems with the archive log

You can recover from situations in which allocation problems occur for the archive log.

Symptoms

The following message is issued:

```
DSNJ103I - csect-name LOG ALLOCATION ERROR DSNAME=dsname,
ERROR STATUS=eeeeiiii, SMS REASON CODE=ssssssss
```

z/OS dynamic allocation provides the ERROR STATUS information. If the allocation is for offload processing, the following message is also issued:

```
DSNJ115I - OFFLOAD FAILED, COULD NOT ALLOCATE AN ARCHIVE DATA SET
```

Causes

Archive log allocation problems can occur when various DB2 operations fail; for example:

- The RECOVER utility executes and requires an archive log. If neither archive log can be found or used, recovery fails.

- The active log becomes full, and an offload is scheduled. Offload tries again the next time it is triggered. The active log does not wrap around; therefore, if no more active logs are available, the offload fails, but data is not lost.
- The input is needed for restart, which fails. If this is the situation that you are experiencing, see "Recovering from BSDS or log failures during restart" on page 500

Resolving the problem

Operator response: 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. If a DFSMSdfp ACS user-exit filter exists for an archive log data set, be careful because this can cause the DB2 subsystem to fail on a device allocation error when DB2 attempts to read the archive log data set.

Recovering from write I/O errors during archive log offload

You can recover from write I/O errors that occur during the offload of an archive log.

Symptoms

No specific DB2 message is issued for write I/O errors. Only a z/OS error recovery program message is issued.

If DB2 message DSNJ128I is issued, an abend in the offload task occurred, in which case you should follow the instructions for this message.

Environment

- 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. For dual archiving, the second copy waits.
- If an error occurs on the new data set, these additional actions occur:
  - For dual archive mode, the following DSNJ114I message 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
    ```

  - For single mode, the offload process 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 no more active logs are available, data is not lost.

Resolving the problem

Operator response: Ensure that offload activity is allocated on a drive and control unit that are operational.

Related information

DSNJ128I (DB2 Messages)

Recovering from read I/O errors on an archive data set during recovery

You can recover from read I/O errors that occur on an archive log during recovery.

Symptoms

No specific DB2 message is issued; only the z/OS error recovery program message is issued.

Environment

- If a second copy of the archive log exists, the second copy is allocated and used.
- If a second copy of the archive log does not exist, recovery fails.

Resolving the problem

Operator response: If you recover from tape, try recovering by using a different drive. If this approach does not work, contact the system programmer.

System programmer response: Recover to the last image copy or to the RBA of the last quiesce point.

Related reference

RECOVER (DB2 Utilities)

Recovering from insufficient disk space for offload processing

If offload processing terminates unexpectedly when DB2 is offloading the active log data sets to disk, you can recover from this situation.

Symptoms

Prior to the failure, z/OS issues abend message IEC030I, IEC031I, or IEC032I. Offload processing terminates unexpectedly. DB2 issues the following message:

```
DSNJ128I - LOG OFFLOAD TASK FAILED FOR ACTIVE LOG nnnnn
```

Additional z/OS abend messages might accompany message DSNJ128I.

Causes

The following situations can cause problems with insufficient disk space during DB2 offload processing:

- 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.
- All available space on the disk volumes to which the archive data set is being written has been exhausted.
- 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.

Environment

DB2 deallocates the data set on which the error occurred. If the subsystem is in dual archive mode, DB2 changes to single archive mode and continues the offload. If the offload cannot complete 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.

Resolving the problem

System programmer response: The actions that you take depend on what caused DB2 message DSNJ128I to be issued:

- If z/OS abend message IEC030I precedes DB2 message DSNJ128I, 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.

Modifications to DSNZP*xxx* require that you stop and start DB2 for the changes to take effect. If the data that is to be offloaded is particularly large, you can mount another online storage volume or make one available to DB2.

- If z/OS abend message IEC032I precedes message DSNJ128I, make space available on the disk volumes, or make another online storage volume available for DB2. After you make additional space available, issue the DB2 command **ARCHIVE LOG CANCEL OFFLOAD**. DB2 then retries the offload.
- If z/OS abend message IEC032I precedes DB2 message DSNJ128I, make space available on the disk volumes, or make available another online storage volume for DB2. If this approach is not possible, adjust the value of PRIQTY in the DSNZP*xxx* module to reduce the primary allocation. If the primary allocation is reduced, you might need to increase the size of the secondary space allocation to avoid future abends.

Related reference

PRIMARY QUANTITY field (PRIQTY subsystem parameter) (DB2 Installation and Migration)

Recovering from BSDS failures

When the bootstrap data set (BSDS) is damaged, you need to recover that BSDS, regardless of whether you are running DB2 in dual-BSDS or single-BSDS mode.

Symptoms

If a BSDS is damaged, DB2 issues one of the following message numbers: DSNJ126I, DSNJ100I, or DSNJ120I.

Related concepts

Management of the bootstrap data set

Recovering from an I/O error on the BSDS

When an I/O error occurs on the only copy of the BSDS, you need to recover the BSDS before DB2 can operate normally. If an I/O error occurs on one copy of the BSDS in a dual-BSDS mode environment, you need to recover that copy of the BSDS before the next restart.

Symptoms

The following message is issued:

```
DSNJ126I - BSDS ERROR FORCED SINGLE BSDS MODE
```

The following messages are then issued:

```
DSNJ107I - READ ERROR ON BSDS
           DSNAME=... ERROR STATUS=...
DSNJ108I - WRITE  ERROR ON BSDS
           DSNAME=... ERROR STATUS=...
```

Causes

A write I/O error occurred on a BSDS.

Environment

If DB2 is in a dual-BSDS mode and one copy of the BSDS is damaged by an I/O error, the BSDS mode changes from dual-BSDS mode to single-BSDS mode. If DB2 is in a single-BSDS mode when the BSDS is damaged by an I/O error, DB2 terminates until the BSDS is recovered.

Resolving the problem

System programmer response:

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. You can find control statements 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.

   Related tasks

   Recovering the BSDS from a backup copy

Recovering from an error that occurs while opening the BSDS

You need to recover the bootstrap data set (BSDS) if an error occurs when DB2 opens the BSDS.

Symptoms

The following message is issued:

```
DSNJ100I - ERROR OPENING BSDS n DSNAME=..., ERROR STATUS=eeii
```

Resolving the problem

System programmer response:

1. Use access method services to delete or rename the damaged data set, to define a replacement data set, and to copy (with the **REPRO** command) the remaining BSDS to the replacement.
2. Use the **START DB2** command to start the DB2 subsystem.

   Related tasks

   Recovering the BSDS from a backup copy

Recovering from unequal timestamps on BSDSs

When timestamps on different copies of the bootstrap data set (BSDS) differ, DB2 attempts to resynchronize the BSDSs and restore dual BSDS mode. If this attempt succeeds,DB2 restart continues automatically. If this attempt fails, you need to recover from the situation.

Symptoms

The following message is issued:

```
DSNJ120I - DUAL BSDS DATA SETS HAVE UNEQUAL TIMESTAMPS,
           BSDS1 SYSTEM=..., UTILITY=..., BSDS2 SYSTEM=..., UTILITY=...
```

Causes

Unequal timestamps can occur for the following reasons:

- One of the volumes that contains the BSDS has been restored. All information of the restored volume is outdated. If the volume contains any active log data sets or DB2 data, their contents are also outdated. The outdated volume has the lower timestamp.
- Dual BSDS mode has degraded to single BSDS mode, and you are trying to start without recovering the bad copy of the BSDS.
- The DB2 subsystem abended after updating one copy of the BSDS, but prior to updating the second copy.

Resolving the problem

Operator response: If DB2 restart fails, notify the system programmer.

System programmer response: If DB2 fails to automatically resynchronize the BSDS data sets:

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 outdated 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 the access method services **REPRO** command to copy the current version of the active log to the outdated data set if all the following conditions are true:
   - The problem was caused by a restored outdated 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, cold start the subsystem.

   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

In some situations, the bootstrap data set (BSDS) becomes damaged, and you need to recover the BSDS from a backup copy.

About this task

DB2 stops and does not restart until dual-BSDS mode is restored in the following situations:

- DB2 is operating in single-BSDS mode, and the BSDS is damaged.
- DB2 is operating in dual-BSDS mode, and both BSDSs are damaged.

Procedure

To recover the BSDS from a backup copy:

1. Locate the BSDS that is associated with the most recent archive log data set. The data set name of the most recent archive log is displayed 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, keep a log of all successful archives that are 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 during its offload), locate an earlier copy of the BSDS from an earlier offload.
3. Rename or delete any damaged BSDS.
   - To rename a damaged BSDS, use the access method services **ALTER** command with the NEWNAME option.
   - To delete a 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 (KSDS) that has three components: cluster, index, and data. You must rename all components of the data set. Avoid changing the high-level qualifier.

4. Use the access method services **REPRO** command to copy the BSDS from the archive log to one of the replacement BSDSs that you defined in the prior step. 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. Use the print log map utility (DSNJU004) to print the contents of the replacement BSDS. You can then 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. Therefore, you need to update the BSDS inventory of the archive log data sets 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. Ensure that 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 the DB2 subsystem for your installation 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 might 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 that 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. Ensure 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. Ensure 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 that are listed in the BSDS with the RBAs that are 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, use the change log inventory utility to change the RBAs that are found in the BSDS to the RBAs in the actual active log data sets. Take the appropriate action, described below, 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 that you use in the change log inventory utility must be at the beginning of a control interval. Similarly, the ENDRBA value that 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 that are 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 step 4f without any additional work.

  If the RBA ranges are not equal, adjust the values in the BSDS to reflect the actual values. For each active log data set for which you need to adjust the RBA range, 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 might have difficulty during restart. The difficulty can arise when one of the active log data sets is full and has not been offloaded, whereas 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 situation.

   To add a new active log data set for each copy of the active log, use the access method services **DEFINE** command. The control statements to accomplish this task can be found in job DSNTIJIN. After 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. You do not need to specify the RBA ranges 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.
6. Optional: Use the print log map utility (DSNJU004) to print the contents of the second replacement BSDS at this point.
7. If you have lost your current active log data set, refer to the following topics:
   - "Recovering from BSDS or log failures during restart" on page 500
   - "Task 4: Truncate the log at the point of error" on page 512, which provides information about how to construct a conditional restart control record (CRCR).
8. Restart DB2, using the newly constructed BSDS. DB2 determines the current RBA and what active logs need to be archived.

Related information

DFSMS Access Method Services Commands

Recovering from BSDS or log failures during restart

When the bootstrap data set (BSDS) or part of the recovery log for DB2 is damaged or lost and that damage prevents restart, you need to recover from that situation. What you do to recover varies based on the particular circumstances.

If the problem is discovered at restart, begin with one of the following recovery procedures:

- "Recovering from active log failures" on page 488
- "Recovering from archive log failures" on page 492
- "Recovering from BSDS failures" on page 495

If the problem persists, return to the procedures in this section.

When DB2 recovery log damage terminates restart processing, DB2 issues messages to the console to identify the damage and issue an abend reason code. (The SVC dump title includes a more specific abend reason code to assist in problem diagnosis.) If the explanations for the reason codes indicate that restart failed because of some problem that is not related to a log error, contact IBM Software Support.

To minimize log problems during restart, the system requires two copies of the BSDS. Dual logging is also recommended.

Basic approaches to recovery: The two basic approaches to recovery from problems with the log are:

- 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 by 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 441. Methods of recovery are described under "Recovering from unresolvable BSDS or log data set problem during restart" on page 523.

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.

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		

The following figure illustrates the general problem.

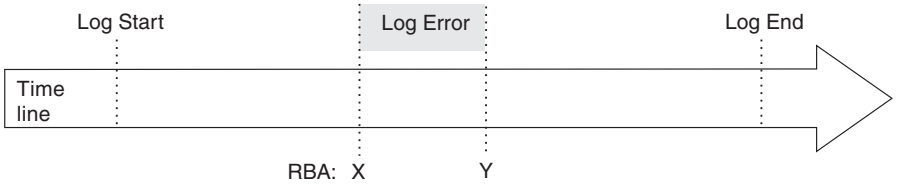


Figure 44. General problem of damaged DB2 log information

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 the preceding figure occurs in the log RBA range between *X* to *Y*. You can restrict restart to all of the log before *X*; then changes later than *X* are not made. Alternatively, 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 that are to be completed and the page sets that they modified.

   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 529.

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 that is identified is the one that was in control when the log problem was detected. Accordingly, start at:

- "Recovering from failure during log initialization or current status rebuild" on page 502
- "Recovering from a failure during forward log recovery" on page 514
- "Recovering from a failure during backward log recovery" on page 519

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 also be issued.

Message ID	Procedure to use
DSNJ001I	"Recovering from failure during log initialization or current status rebuild" on page 502

Message ID	Procedure to use
DSNJ100I	"Recovering from unresolvable BSDS or log data set problem during restart" on page 523
DSNJ107	"Recovering from unresolvable BSDS or log data set problem during restart" on page 523
DSNJ1191	"Recovering from unresolvable BSDS or log data set problem during restart" on page 523
DSNR002I	None. Normal restart processing can be expected.
DSNR004I	"Recovering from a failure during forward log recovery" on page 514
DSNR005I	"Recovering from a failure during backward log recovery" on page 519
DSNR006I	None. Normal restart processing can be expected.
Other	"Recovering from failure during log initialization or current status rebuild"

Another procedure ("Recovering from a failure resulting from total or excessive loss of log data" on page 525) provides information to use if you determine (by using "Recovering from failure during log initialization or current status rebuild") that an excessive amount (or all) of DB2 log information (BSDS, active, and archive logs) has been lost.

The last procedure,"Resolving inconsistencies resulting from a conditional restart" on page 529, can be used to resolve inconsistencies introduced while using one of the restart procedures in this information. If you decide to use "Recovering from unresolvable BSDS or log data set problem during restart" on page 523, you do not need to use "Resolving inconsistencies resulting from a conditional restart" on page 529.

Because of the severity of the situations described, the procedures 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.

Related reference:

DSN1LOGP (DB2 Utilities)

Recovering from failure during log initialization or current status rebuild

When a failure occurs during the log initialization phase or the current status rebuild phase of restart, you need to recover from this situation.

Symptoms

An abend was issued, indicating that restart failed. In addition, either the last restart message that was received was a DSNJ001I message that indicates a failure during current status rebuild, or none of the following messages was issued:

- DSNJ001I
- DSNR004I
- DSNR005I

If none of the preceding messages was issued, the failure occurred during the log initialization phase of restart.

Environment

What happens in the environment depends on whether the failure occurred during log initialization or 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 at the prior DB2 termination. Possible states include: outstanding units of recovery, outstanding database writes, and exception database conditions.

Resolving the problem

Operations management response: 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, read the relevant information about the messages and codes that you received. The explanations for the messages and codes identify the corrective action that can be taken to resolve the problem.
- Restore the DB2 log and all data to a prior consistent point, and then start DB2. This procedure is described in "Recovering from unresolvable BSDS or log data set problem during restart" on page 523.
- Start DB2 without completing some database changes. Using a combination of DB2 services and your own knowledge, determine what work is likely to be lost if you truncate the log. The procedure for determining the page sets that contain incomplete changes is described in "Restarting DB2 by truncating the log" on page 505.

Failure during log initialization phase

When a failure occurs during the log initialization phase, certain characteristics of the situation are evident.

The following figure illustrates the timeline of events that exist when a failure occurs during the log initialization phase.

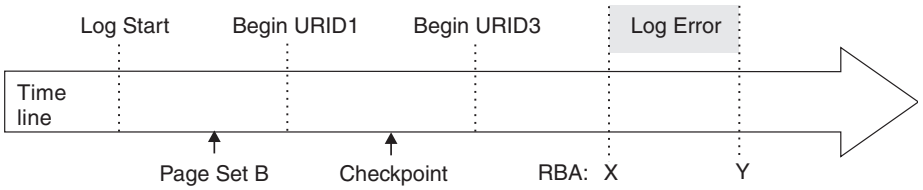


Figure 45. Failure during log initialization

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 an abend reason code. Some of the abend reason codes that might be issued include:
   - 00D10261
   - 00D10262
   - 00D10263

- 00D10264
- 00D10265
- 00D10266
- 00D10267
- 00D10268
- 00D10329
- 00D1032A
- 00D1032B
- 00D1032C
- 00E80084

During its operations, DB2 periodically records in the BSDS the RBA of the last log record that was 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 that are 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 is 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 that are 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.

A restart of DB2 in this situation requires truncation of the log.

Related tasks

Restarting DB2 by truncating the log

Description of failure during current status rebuild

When a failure occurs during current status rebuild, certain characteristics of the situation are evident.

The following figure illustrates the timeline of events that exist when a failure occurs during current status rebuild.

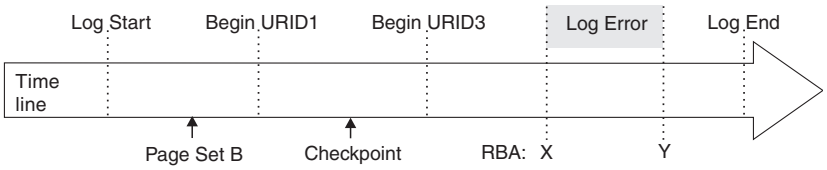


Figure 46. Failure during current status rebuild

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 status of the subsystem 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 and issues an abend reason code. Some of the abend reason codes that might be issued include:
   - 00D10261
   - 00D10262
   - 00D10263
   - 00D10264
   - 00D10265
   - 00D10266
   - 00D10267
   - 00D10268
   - 00D10329
   - 00D1032A
   - 00D1032B
   - 00D1032C
   - 00E80084

Because the end of the log is inaccessible in this case, some information is 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 that are 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.

A restart of DB2 in this situation requires truncation of the log.

Related tasks

Restarting DB2 by truncating the log

Restarting DB2 by truncating the log

A portion of the log is inaccessible during the log initialization or current status rebuild phases of restart. When the log is inaccessible, DB2 cannot identify

precisely what units of recovery failed to complete, what page sets had been modified, and what page sets have writes pending. You need to gather that information, and restart DB2.

Task 1: Find the log RBA after the inaccessible part of the log:

The first task in restarting DB2 by truncating the log is to locate the log RBA after the inaccessible part of the log.

About this task

The range of the log between RBAs *X* and *Y* is inaccessible to all DB2 processes.

Procedure

To find the RBA after the inaccessible part of the log, take the action that is associated with the message number that you received (DSNJ007I, DSNJ012I, DSNJ103I, DSNJ104I, DSNJ106I, and DSNJ113E):

- **When message DSNJ007I is issued:**

  The problem is that an operator canceled a request for archive mount. Reason code 00D1032B is associated with this situation and indicates that an entire data set 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 (DSNJU004) to list the log inventory information. The output of this utility provides each log data set name and its associated log RBA range, the values of *X* and *Y*.

- **When message DSNJ012I is issued:**

  The problem is that a log record is logically damaged. Message DSNJ012I identifies the log RBA of the first inaccessible log record that DB2 detects. The following reason codes are associated with this situation:
  - 00D10261
  - 00D10262
  - 00D10263
  - 00D10264
  - 00D10265
  - 00D10266
  - 00D10267
  - 00D10268
  - 00D10348

  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
  ```

  A given physical log record is actually a set of logical log records (the log records that are 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. (For the example message above, the rounded 4-KB boundary value would be X'7429000'.)

- **When message DSNJ103I or DSNJ104I is issued:**

  For message DSNJ103I, the underlying problem depends on the reason code that is issued:

  – For reason code 00D1032B, an allocation error occurred for an archive log data set.

  – For reason code 00E80084, an active log data set that is named in the BSDS could not be allocated during log initialization.

  For message DSNJ104I, the underlying problem is that an open error occurred for an archive and active log data set.

  In any of these cases, the message that accompanies the abend identifies an entire data set that is inaccessible. For example, the following DSNJ103I 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.

```
DSNJ103I - csect-name LOG ALLOCATION ERROR
         DSNAME=DSNCAT.ARCHLOG1.A0000009,ERROR
         STATUS=04980004
         SMS REASON CODE=reasond-code
```

  To determine the value of *X*, run the print log map utility (DSNJU004) to list the log inventory information. The output of the utility 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 need 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 begins 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 might be delayed by other processing. Therefore, a possible outcome is 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.

  If the data set is marked "Not Reusable," 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. If the data set is marked "Reusable," you can assume for the purposes of this restart that the starting log RBA (*X*) for this data set is one greater than the highest log RBA that is listed in the BSDS for all other active log data sets.

- **When message DSNJ106I is issued:**

  The problem is that an I/O error occurred while a log record was being read. The message identifies the log RBA of the first inaccessible log record that DB2 detects. Reason code 00D10329 is associated with this situation.

  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
```

A given physical log record is actually a set of logical log records (the log records that are generally spoken of) and the log control interval definition (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. This is due 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. (For the example message above, the rounded 4-KB boundary value would be X'7429000'.)

- **When message DSNJ113E is issued:**

  The problem is that the log RBA could not be found in the BSDS. Message DSNJ113E identifies the log RBA of the inaccessible log record. This log RBA is not registered in the BSDS. Reason code 00D1032B is associated with this situation.

  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 (DSNJU004) to list the contents of the BSDS.

  A given physical log record is actually a set of logical log records (the log records that are generally spoken of) and the log control interval definition (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 that is closest to, but less than, X'7429ABA' for the value of *X*. If you do not find an RBA that is less than X'7429ABA', a considerable amount of log information has been lost. If this is the case, continue with "Recovering from a failure resulting from total or excessive loss of log data" on page 525. Otherwise, continue with the next topic.

  Related concepts

  Description of failure during current status rebuild

  Failure during log initialization phase

  Related reference

  DSNJU004 (print log map) (DB2 Utilities)

  Related information

  DSNJ007I (DB2 Messages)

  DSNJ012I (DB2 Messages)

  DSNJ103I (DB2 Messages)

  DSNJ104I (DB2 Messages)

  DSNJ106I (DB2 Messages)

  DSNJ113E (DB2 Messages)

Task 2: Identify lost work and inconsistent data:

In certain recovery situations (such as when you recover by truncating the log), you need to identify what work was lost and what data is inconsistent.

Procedure

To identify lost work and inconsistent data:

1. Obtain available information to help you determine the extent of the loss. DB2 cannot 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 steps below explain what to do to obtain the information that is available within DB2 to help you 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.

   Attention: If the inconsistent page sets are not identified and the problems in them are not resolved after starting DB2, be aware that following this procedure and allowing access to inconsistent data involves some risk.

   a. Run the print log map utility. The report that the utility produces includes a description of the last 100 checkpoints and provides, for each checkpoint the following information:
      - 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*.

      Continue with the step 2 unless one of the following conditions exists:
      - You cannot find such a checkpoint. This means that a considerable amount of log has been lost.
      - You find the checkpoint, but the checkpoint is several days old, and DB2 has been operational during the interim.

      In these two cases, use one of the following procedures:
      - "Recovering from a failure resulting from total or excessive loss of log data" on page 525
      - "Recovering from unresolvable BSDS or log data set problem during restart" on page 523

2. Determine what work is lost and what data is inconsistent. The portion of the log that represents activity that occurred before the failure provides information about work that was in progress at that point. From this information, you might be able to deduce what work 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 that perform database loads or image copies), you might be able 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.
   - Use the DSN1LOGP utility to specify the "BEGIN CHECKPOINT" RBA prior to the point of failure, which was determined in the previous task as the RBASTART. Terminate the DSN1LOGP scan prior to the point of failure on the log (*X* - 1) by using the RBAEND specification.
   - Specify the last complete checkpoint. This is very important for ensuring that complete information is obtained from DSN1LOGP.
   - Specify the SUMMARY(ONLY) option to produce a summary report.

The following figure is an example of a DSN1LOGP job that obtains summary information for the checkpoint that was described previously.

```
//ONE EXEC PGM=DSN1LOGP
//STEPLIB  DD DSN=prefix.SDSNLOADSDSNLOAD,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 47. Sample JCL for obtaining DSN1LOGP summary output for restart

3. Analyze the DSN1LOGP utility output.

Related reference

DSN1LOGP (DB2 Utilities)

DSN1LOGP summary report:

The DSN1LOGP utility generates a summary report, which is placed in the SYSSUMRY file. The report includes a summary of completed events and a restart summary. You can use the information in this report to identify lost work and inconsistent data that needs to be resolved.

The following figure shows an excerpt from the restart summary in a sample **DSN1LOGP** summary report. The report is described 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 48. Partial sample of DSN1LOGP summary output

The following message acts as a heading, which 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 that have database writes that are 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 that is 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 that were modified by these units of recovery are inconsistent after completion of restart when you attempt to identify lost work and inconsistent data.

All page sets that are 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 previously described case, all of these page sets are inconsistent after completion of restart when you attempt to identify lost work and inconsistent data.

At this point, you need to identify only the page sets in preparation for restart. After restart, you need to resolve the problems in the page sets that are inconsistent.

Because the end of the log is inaccessible, some information is lost; therefore, the information is inaccurate. Some of the units of recovery that appear to be inflight might have successfully committed, or they might have modified additional page sets beyond point *X*. Additional data might have been written, including those page sets that are identified as having pending writes in the accessible portion of the log. New units of recovery might have been created, and these might have modified data. DB2 cannot detect that these events occurred.

From this and other information (such as system accounting information and console messages), you might be able to determine what work was actually outstanding and which page sets are likely to be inconsistent after you start DB2. This is because the record of each event contains the date and time to help you determine how recent the information is. In addition, the information is displayed in chronological sequence.

Task 3: Determine what status information is lost:

The third task in restarting DB2 by truncating the log is to determine what status information has been lost.

About this task

Depending on what was going on in your environment before the problem occurred, some amount of system status information might have been lost.

Procedure

To determine what system status information is lost:

1. If you already know what system status information is lost (such as in the case in which utilities are in progress), you do not need to do anything. Continue with the next topic.
2. If you do not already know what system status information is lost, examine all relevant messages that provide details about the loss of status information (such as in the cases of deferred restart pending or write error ranges). If the messages provide adequate information about what information is lost, you do not need to do anything more. Continue with the next step.
3. If you find that all system status information is lost, try 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.

Task 4: Truncate the log at the point of error:

The fourth task in restarting DB2 by truncating the log is to truncate the log at the point of error.

About this task

No DB2 process, including the RECOVER utility, 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*.

Procedure

To truncate the log at the point of error:

Create a conditional restart control record (CRCR) in the BSDS by using the change log inventory utility. Specify the following options:

ENDRBA=*endrba*
The *endrba* value is the RBA at which DB2 begins writing new log records. If point *X* is X'7429000', specify ENDRBA=7429000 on the CRESTART control statement.

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, DB2 performs normal restart processing within the scope of the log. Because log information is lost, DB2 errors might occur. For example, a unit of recovery that has actually been committed might be rolled back. Also, some changes that were made by that unit of recovery might not be rolled back because information about data changes is lost.

FORWARD=NO
Terminates forward-log recovery before log records are processed. This option and the BACKOUT=NO option minimize errors that might result from normal restart processing.

BACKOUT=NO
Terminates backward-log recovery before log records are processed. This option and the FORWARD=NO option minimize errors that might result from normal restart processing.

Results

Recovering and backing out units of recovery with lost information might introduce more inconsistencies than the incomplete units of recovery.

Example

The following example is a CRESTART control statement for the ENDRBA value of X'7429000':

```
CRESTART  CREATE,ENDRBA=7429000,FORWARD=NO,BACKOUT=NO
```

Task 5: Start DB2 and resolve data inconsistencies:

The final task in restarting DB2 by truncating the log is to restart DB2 and resolve inconsistencies.

Before you begin

You must have a CRESTART control statement with the correct ENDRBA value and the FORWARD and BACKOUT options set to NO.

Procedure

To start DB2 and resolve data inconsistencies:

1. Start DB2 with the following command:

   ```
   -START DB2 ACCESS (MAINT)
   ```

   In response to this command, DB2 performs the following actions:

   - Discards from the checkpoint queue any entries with RBAs that are beyond the ENDRBA value in the CRCR (for example, X'7429000').
   - Reconstructs the system status up to the point of log truncation.
   - Performs pending database writes that the truncated log specifies and that have not already been 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 standard forward-log recovery processing for the different unit of work states does not occur.
   - Marks all units of recovery that have committed or are indoubt as complete on the log.
   - Leaves inflight and in-abort units of recovery incomplete. Inconsistent data is left in tables that are modified by inflight or indoubt units of recovery. 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 would be normal during backward-log recovery) DB2 might back out database changes incompletely, which introduces additional errors.
     – Data that is modified by in-abort units of recovery might have been modified again after the point of damage on the log. For in-abort units of recovery, DB2 might have written backout processing to disk after the

point of log truncation. If these units of recovery are backed out (as would be normal during backward-log recovery), DB2 might introduce additional data inconsistencies by backing out units of recovery that are already partially or fully backed out.

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 completes successfully, the CRCR is in effect. Until data is consistent or page sets are stopped, start DB2 with the ACCESS (MAINT) option.

2. Resolve all data inconsistency problems.

Related concepts

Phase 4: Backward log recovery

Phase 3: Forward log recovery

Related tasks

Resolving inconsistencies resulting from a conditional restart

Recovering from a failure during forward log recovery

If a failure occurs during the forward-log recovery phase of restart, operations management can recover from this situation.

Symptoms

A DB2 abend occurred, indicating that restart had failed. In addition, the last restart message that was received was a DSNR004I message, which indicates that log initialization completed; therefore, the failure occurred during forward log recovery.

Environment

DB2 terminates because a portion of the log is inaccessible, and DB2 is therefore unable to guarantee the consistency of the data after restart.

Resolving the problem

Operations management response: To start DB2 successfully, choose one of the following approaches:

- Read the information about relevant messages and codes that you received to determine if this approach is possible. The explanations of the messages and codes identify any corrective action that you can take to resolve the problem. If it is possible, correct the problem that made the log inaccessible, and start DB2 again.
- Restore the DB2 log and all data to a prior consistent point and start DB2. This procedure is described in "Recovering from unresolvable BSDS or log data set problem during restart" on page 523.
- Start DB2 without completing some database changes. Do this only if the exact changes cannot be identified; all that can be determined is which page sets might have 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 "Recovering from BSDS or log failures during restart" on page 500. Other topics might help you better understand the problem.

Forward-log recovery failure

When a failure occurs during the forward-log recovery phase of DB2 restart, certain characteristics of the situation are evident.

The following figure illustrates the events surrounding a failure during the forward-log recovery phase of DB2 restart.

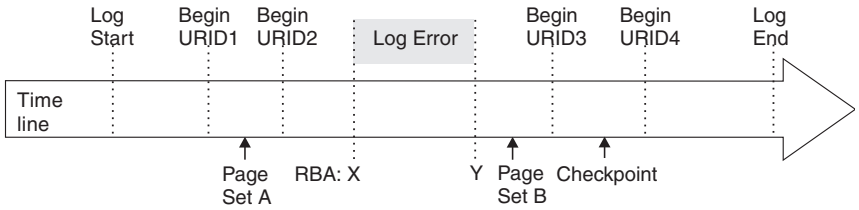


Figure 49. Illustration of failure during forward-log recovery

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*.

Assume that the following work was outstanding at restart:

- The unit of recovery that is identified as URID1 was in-commit.
- The unit of recovery that is identified as URID2 was inflight.
- The unit of recovery that is identified as URID3 was in-commit.
- The unit of recovery that is identified as URID4 was inflight.
- Page set A had writes that were pending prior to the error on the log, continuing to the end of the log.
- Page set B had writes that were 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 49. 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 that are made by URID1 and URID2 might not be fully committed or backed out. Writes that were pending for page set A (from points in the log prior to *Y*) are lost.

Starting DB2 by limiting restart processing

When a portion of the log is inaccessible during forward recovery, starting DB2 is possible. You need to identify the units of recovery for which database changes cannot be fully guaranteed (either committed or backed out). You also need to identify the page sets that these units of recovery changed.

About this task

You must determine which page sets are involved because after this procedure is used, the page sets will contain inconsistencies that you must resolve. In addition, using this procedure results in the completion of all database writes that are pending.

Related concepts

Write operations (DB2 Performance)

Task 1: Find the log RBA after the inaccessible part of the log:

The first task in restarting DB2 by limiting restart processing is to locate the log RBA that is after the inaccessible part of the log.

About this task

The range of the log between RBAs *X* and *Y* is inaccessible to all DB2 processes.

Procedure

To find the RBA after the inaccessible part of the log, take the action that is associated with the message number that you received (DSNJ007I, DSNJ012I, DSNJ103I, DSNJ104I, DSNJ106I, and DSNJ113E):

- **When message DSNJ007I is issued:**

  The problem is that an operator canceled a request for archive mount. Reason code 00D1032B is associated with this situation and indicates that an entire data set 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 (DSNJU004) to list the log inventory information. The output of this utility provides each log data set name and its associated log RBA range, the values of *X* and *Y*.

- **When message DSNJ012I is issued:**

  The problem is that a log record is logically damaged. Message DSNJ012I identifies the log RBA of the first inaccessible log record that DB2 detects. The following reason codes are associated with this situation:
  - 00D10261
  - 00D10262
  - 00D10263
  - 00D10264
  - 00D10265
  - 00D10266
  - 00D10267
  - 00D10268
  - 00D10348

  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
  ```

  A given physical log record is actually a set of logical log records (the log records that are 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 forward log recovery, 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. (For the example message above, the rounded 4-KB boundary value would be X'7429000'.)

- **When message DSNJ103I or DSNJ104I is issued:**

  For message DSNJ103I, the underlying problem depends on the reason code that is issued:

  – For reason code 00D1032B, an allocation error occurred for an archive log data set.

  – For reason code 00E80084, an active log data set that is named in the BSDS could not be allocated during log initialization.

  For message DSNJ104I, the underlying problem is that an open error occurred for an archive and active log data set.

  In any of these cases, the message that accompanies the abend identifies an entire data set that is inaccessible. For example, the following DSNJ103I 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.

```
DSNJ103I - csect-name LOG ALLOCATION ERROR
          DSNAME=DSNCAT.ARCHLOG1.A0000009,ERROR
          STATUS=04980004
          SMS REASON CODE=reasond-code
```

  To determine the value of *X*, run the print log map utility (DSNJU004) to list the log inventory information. The output of the utility provides each log data set name and its associated log RBA range, the values of *X* and *Y*.

- **When message DSNJ106I is issued:**

  The problem is that an I/O error occurred while a log record was being read. The message identifies the log RBA of the first inaccessible log record that DB2 detects. Reason code 00D10329 is associated with this situation.

  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
```

  A given physical log record is actually a set of logical log records (the log records that are 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, and beyond it to the end of the log data set, are inaccessible to the forward log recovery phase of restart. This is due 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. (For the example message above, the rounded 4-KB boundary value would be X'7429000'.)

- **When message DSNJ113E is issued:**

  The problem is that the log RBA could not be found in the BSDS. Message DSNJ113E identifies the log RBA of the inaccessible log record. This log RBA is not registered in the BSDS. Reason code 00D1032B is associated with this situation.

  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 (DSNJU004) to list the contents of the BSDS.

  A given physical log record is actually a set of logical log records (the log records that are 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 are inaccessible.

Using the print log map output, locate the RBA that is closest to, but less than, X'7429ABA' for the value of *X*. If you do not find an RBA that is less than X'7429ABA', the value of *X* is zero. Locate the RBA that is closest to, by greater than, X'7429ABA'. This is the value of *Y*.

Related concepts

Forward-log recovery failure

Related reference

DSNJU004 (print log map) (DB2 Utilities)

Related information

DSNJ007I (DB2 Messages)

DSNJ012I (DB2 Messages)

DSNJ103I (DB2 Messages)

DSNJ104I (DB2 Messages)

DSNJ106I (DB2 Messages)

DSNJ113E (DB2 Messages)

Task 2: Identify incomplete units of recovery and inconsistent page sets:

The second task in restarting DB2 by limiting restart processing is to identify incomplete units of recovery and inconsistent page sets.

About this task

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*.

Procedure

To identify incomplete units of recovery and inconsistent page sets:

1. Determine the location of the latest checkpoint on the log 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'. For example:

     ```
     DSNR003I RESTART ... PRIOR CHECKPOINT
                          RBA=00007425468
     ```

   - The print log map utility output identifies the last checkpoint, including its BEGIN CHECKPOINT RBA
2. Obtain a report of the outstanding work that is to be completed at the next restart of DB2 by running the DSN1LOGP utility. When you run the DSN1LOGP utility, specify the checkpoint RBA as the STARTRBA and the SUMMARY(ONLY) option. In order to obtain complete information, be sure to include the last complete checkpoint from running DSN1LOGP.
3. Analyze the output of the DSN1LOGP utility. The summary report that is placed in the SYSSUMRY file contains two sections of information: a complete summary of completed events and a restart summary.

Related concepts

DSN1LOGP summary report

Related reference

DSN1LOGP (DB2 Utilities)

Task 3: Restrict restart processing to the part of the log after the damage:

The third task in restarting DB2 by limiting restart processing is to restrict restart processing to the part of the log that is after the damage.

Procedure

To restrict restart processing to the part of the log after the damage:

1. Create a conditional restart control record (CRCR) in the BSDS by using the change log inventory utility.
2. Identify the accessible portion of the log beyond the damage by using the STARTRBA specification, which will be used at the next restart.
3. Specify the value *Y*+1 (that is, if *Y* is X'7429FFF', specify STARTRBA=742A000). Restart restricts its processing to the portion of the log beginning with the specified STARTRBA and continuing to the end of the log. For example:

   ```
   CRESTART CREATE,STARTRBA=742A000
   ```

Task 4: Start DB2 and resolve inconsistent data:

The final task in restarting DB2 by limiting restart processing is to start DB2 and resolve problems with inconsistent data.

About this task

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.

Procedure

To start DB2 and resolve data inconsistencies:

1. Start DB2 with the following command:

   ```
   -START DB2 ACCESS (MAINT)
   ```

   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 completes successfully, the CRCR is in effect. Until data is consistent or page sets are stopped, start DB2 with the ACCESS (MAINT) option.
2. Resolve all data inconsistency problems.

   Related tasks

   Resolving inconsistencies resulting from a conditional restart

Recovering from a failure during backward log recovery

When a failure occurs during backward log recovery, DB2 terminates because it cannot access a portion of the log that it needs. Operations management can recover from this situation.

Symptoms

An abend is issued to indicate that restart failed because of a log problem. In addition, the last restart message that is received is a DSNR005I message, indicating that forward log recovery completed and that the failure occurred during backward log recovery.

Environment

Because a portion of the log is inaccessible, DB2 needs to roll back some database changes during restart.

Resolving the problem

Operations management response: To start DB2, choose one of the following approaches:

- Read the information about relevant messages and codes that you received to determine if this approach is possible. The explanations of the messages and codes identify any corrective action that you can take to resolve the problem. If it is possible, correct the problem that made the log inaccessible, and start DB2 again.
- Restore the DB2 log and all data to a prior consistent point and start DB2. This procedure is described in "Recovering from unresolvable BSDS or log data set problem during restart" on page 523.
- Start DB2 without completing some database changes. Do this only if 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 "Bypassing backout before restarting" on page 521. Other related topics might help you better understand the problem.

Backward log recovery failure

If a failure occurs during the backward-log recovery phase of restart, operations management can recover from this situation.

When a failure occurs during the backward log recovery phase, certain characteristics of the situation are evident, as the following figure shows.

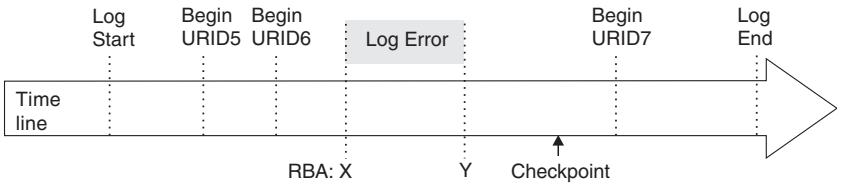


Figure 50. Illustration of failure during backward log recovery

The portion of the log between log RBA *X* and *Y* is inaccessible. The restart process 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 that were 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 of the log 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 that were 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.

Related concepts

Recommendations for changing the BSDS log inventory

Bypassing backout before restarting

A portion of the log becomes inaccessible when a failure occurs during backward log recovery. Operations management can recover from this situation by starting DB2 in a certain way, and then identifying the page sets that are inconsistent because of the incomplete units of recovery.

Procedure

To bypass backout before recovery:

1. Determine the units of recovery that cannot be backed out and the page sets that will be inconsistent after the completion of restart.
   a. Determine the location of the latest checkpoint on the log 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
        ```

      - The print log map utility output identifies the last checkpoint, including its BEGIN CHECKPOINT RBA.

   b. Obtain a report of the outstanding work that is to be completed at the next DB2 restart by running the DSN1LOGP. 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.

      Analyze the output of the DSN1LOGP utility. The summary report that is placed in the SYSSUMRY file contains two sections of information. The heading of first section of the output is the following message:

      ```
      DSN1150I SUMMARY OF COMPLETED EVENTS
      ```

      That message is followed by other messages that identify completed events, such as completed units of recovery. That section of the output does not apply to this procedure.

      The heading of the second section of the output is the following 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. After the summary of outstanding units of recovery is a summary of page sets with database writes that are pending.

      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 that are 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 that were 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, you only need to identify the page sets in preparation for restart.

2. Use the change log inventory utility to create a conditional restart control record (CRCR) in the BSDS, and direct restart to bypass backward recovery processing during the subsequent restart by using the BACKOUT specification. At restart, all units of recovery that require 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, resolve all data inconsistency problems. "Resolving inconsistencies resulting from a conditional restart" on page 529 describes how to do this. At this time, make all other data available for use.

   Related concepts

   DSN1LOGP summary report

Recovering from a failure during a log RBA read request

A failure might occur during a log RBA read request. For example, because of problems with the BSDS, the requested RBA, which contains the dropped log data set, cannot be read. You can recover from the situation.

Symptoms

Abend code 00D1032A is issued, and message DSNJ113E is displayed:

```
DSNJ113E RBA log-rba NOT IN ANY ACTIVE OR ARCHIVE
        LOG DATA SET.  CONNECTION-ID=aaaaaaaa, CORRELATION-ID=aaaaaaaa
```

Causes

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.

Resolving the problem

System programmer response:

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 that are displayed, and correct the errors that are indicated. Locate the output from an old execution of the print log map utility, 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.

Related tasks

Updating subsystem parameter values (DB2 Installation and Migration)

Related reference

RECORDING MAX field (MAXARCH subsystem parameter) (DB2 Installation and Migration)

DSNJU003 (change log inventory) (DB2 Utilities)

Recovering from unresolvable BSDS or log data set problem during restart

During a restart of DB2, serious problems with the bootstrap data set (BSDS) or log data sets might occur. However, operations management can recover from these problems. Use of dual logging (active logs, archive logs, and bootstrap data sets) can reduce your efforts in resolving these sorts of problems.

Symptoms

The following messages are issued:

- DSNJ100I
- DSNJ107I
- DSNJ119I

Causes

Any of the following problems might cause problems with the BSDS or log data sets during restart:

- A log data set is physically damaged.
- Both copies of a log data set are physically damaged in the case of dual logging mode.
- A log data set is lost.
- An archive log volume was reused even though it was still needed.
- A log data set contains records that are not recognized by DB2 because they are logically broken.

Environment

DB2 cannot be started until this procedure is performed.

Resolving the problem

Operations management response: Serious cases such as this sometimes necessitate a fallback to a prior shutdown level.

- If you decide to fall back (because the amount of lost information is not excessive):
  1. See "Preparing to recover to a prior point of consistency" on page 441.
  2. Follow the procedure in "Falling back to a prior shutdown point" on page 524.

  If you use do a fallback, all database changes between the shutdown point and the present are lost. However, all the data that is retained will be consistent within DB2.

- If you decide not to fall back (because too much log information has been lost), use the alternative approach that is described in "Recovering from a failure resulting from total or excessive loss of log data" on page 525.

Falling back to a prior shutdown point

When a failure occurs in your environment, you might decide to fall back to a prior shutdown point.

Procedure

To fallback to a prior shutdown point:

1. Use the print log map utility on 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 is issued.)
2. Use the access method services **IMPORT** command to restore the backed-up versions of the BSDS and active log data sets.
3. Use the print log map utility on 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 that is shown in the BSDS that is being used to restart DB2.
     – If no archive log data sets with a higher sequence number exist, you do not need 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 that is 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 **START DB2 ACCESS(MAINT)** command until data is consistent or page sets are stopped.
6. 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 proper order.
   - For data that has not been changed after the shutdown point (data used with RO access), you do not need 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.
   - Re-create any objects that were created after the shutdown point.

   You must recover all data that has potentially been modified after the shutdown point. If you do not use the RECOVER utility to recover modified data, serious problems might can occur because of data inconsistency.

If you try to access inconsistent data, any of the following events can occur (and the list is not comprehensive):

- You can successfully access the correct data.
- You can access data without DB2 recognizing any problem, but it might not be the data that 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 users (TSO users, QMF users, and batch users for whom 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 might exist. This occurs if transactions are indoubt when the **STOP DB2 MODE (QUIESCE)** command is issued. 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 **RECOVER INDOUBT** command.
9. If a table space was dropped and re-created after the shutdown point, drop and re-create it 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 might contain inconsistent data) that is being recovered.
10. If any table spaces and indexes were created after the shutdown point, re-create these after DB2 is restarted. You can accomplish this in these ways:
    - For data sets that are 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 the access method services **DELETE** command to delete the old data sets. After these data sets have been deleted, use the access method services **DEFINE** command to redefine them; then use the CREATE TABLESPACE statement.

Related reference

RECOVER (DB2 Utilities)

Recovering from a failure resulting from total or excessive loss of log data

If a situation occurs that causes the entire log or an excessive amount of log data to be lost or destroyed, operations management needs to recover from that situation.

Symptoms

This situation is generally accompanied by messages or abend reason codes that indicate that an excessive amount of log information, or the entire log, has been lost.

Diagnosing the problem

In this situation, you need to rely on your own sources to determine what data is inconsistent as a result of the failure; DB2 cannot provide any hints of inconsistencies. 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 that they modified. If you cannot identify the page sets that are inconsistent, you must decide whether you are willing to assume the risk that is involved in restarting DB2 under those conditions.

Resolving the problem

Operations management response: If you decide that a restart is needed, restart DB2 without any log data by using the appropriate procedure, depending on whether the log is totally or partially (but excessively) lost.

Recovering from a total loss of the log

If all system and user table spaces remain intact and you have a recent copy of the BSDS, you can recover from a total loss of the log. DB2 can still be restarted, and data that belongs to that DB2 subsystem can still be accessed.

Before you begin

All system and user table spaces must be intact, and you must have a recent copy of the BSDS. Other VSAM clusters on disk, such as the system databases DSNDB01, DSNDB04, and DSNB06, and also user databases are assumed to exist.

Procedure

To restart DB2 when the entire log is lost:

1. Define and initialize the BSDSs by recovering the BSDS from a backup copy.
2. Define the active log data sets by using the access method services **DEFINE** command. Run utility DSNJLOGF to initialize the new active log data sets.
3. Prepare to restart DB2 with no log data. Each data and index page contains the log RBA of the last log record that was 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. You have two choices:
   - Determine the highest possible log RBA of the prior log. From previous console logs that were 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 that specifies 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 you choose this method, you do not need to use the RESET option of the DSN1COPY utility, and you can save a lot of time.
   - 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 might 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, you can force logging 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
```

4. Start DB2. Use the **START DB2 ACCESS(MAINT)** command 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 529.

Related tasks

Deferring restart processing

Recovering the BSDS from a backup copy

Related reference

DSNJLOGF (preformat active log) (DB2 Utilities)

Recovering from an excessive loss of active log data

When your site experiences an excessive loss of active log data, you can develop a procedure for restarting in this situation. Do not redefine the BSDS.

About this task

You can recover from an excessive loss of active log data in one of two ways.

Recovering DB2 by creating a gap in the active log:

If your site experiences an excessive loss of active log data, you can recover by creating a gap in the active log.

Procedure

To recover by creating a gap in the active log:

1. Use the print log map utility (DSNJU004) on 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 the access method services **LISTCAT** command to determine which active log data sets are no longer available or usable.
3. Use the access method services **DELETE** command to delete all active log data sets that are no longer usable.
4. Use the access method services **DEFINE** command to define new active log data sets. Run the DSNJLOGF utility to initialize the new active log data sets. Define one active log data set for each one that is found to be no longer available or usable in step 2. Use the active log data set name that is found in the BSDS as the data set name for the access method services DEFINE command.
5. Refer to 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 recently 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 recently 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 to the BSDS active log data set inventory. Do not specify RBA ranges on the NEWLOG statement.

If the corresponding archive log data sets exist, two courses of action are available:

- If you want to minimize the number of potential read operations on the archive log data sets, use the access method services **REPRO** command to copy the data from each archive log data set into the corresponding active log data set. Ensure that you copy the proper RBA range into the active log data set.

  Ensure that the active log data set is large 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, and then pads the archive log data set with binary zeros 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 recently defined active log data set, the new active log data set might need to be larger 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 steps 5a on page 527 and 5b (as though the archive data sets did not exist).

6. Choose the appropriate point for DB2 to start logging. To do this, determine the highest possible log RBA of the prior log. From previous console logs that were 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 that specifies 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 you choose this method, you do not need to use the RESET option of the DSN1COPY utility, and you can save a lot of time.
7. To restart DB2 without using any log data, create a conditional restart control record for the change log inventory utility (DSNJU003).
8. Start DB2. Use the **START DB2 ACCESS(MAINT)** command 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 529.

Results

This procedure causes all phases of restart to be bypassed and logging to begin at the point in the log RBA that you identified in step 6 (X'8C000' in the example given in this procedure). This procedure creates a gap in the log between the highest RBA kept in the BSDS and, in this example, X'8C000', and that portion of the log is inaccessible.

What to do next

Because no DB2 process can tolerate a gap, including RECOVER, you need to take image copies of all data after a cold start, even data that you know is consistent.

Related reference

DSNJU003 (change log inventory) (DB2 Utilities)

Recovering DB2 without creating a gap in the active log:

You can do a cold start without creating a gap in the log. Although this approach does eliminate the gap in the physical log record, you cannot use a cold start to resolve the logical inconsistencies.

Procedure

To recover without creating a gap in the active log:

1. Locate the last valid log record by using the DSN1LOGP utility to scan the log. Message DSN1213I identifies the last valid log RBA.
2. Identify the last RBA that is known to be valid by examining message DSN1213I. For example, if message DSN1213I indicates that the last valid log RBA is X'89158', round this value up to the next 4-KB boundary, which in this example is X'8A000'.
3. Create a conditional restart control record (CRCR). For example:
   ```
   CRESTART CREATE,STARTRBA=8A000,ENDRBA=8A000
   ```
4. Start DB2 with the **START DB2 ACCESS(MAINT)** command until data is consistent or page sets are stopped.
5. Take image copies of all data for which data modifications were recorded beyond the log RBA that was used in the CRESTART statement (in this example, X'8A000'). If you do not know what data was modified, take image copies of all data.

   If you do not take image copies of data that has been modified beyond the log RBA that was used in the CRESTART statement, future RECOVER utility operations might fail or result in inconsistent data.

What to do next

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, several problems might occur. Recovery from these problems is possible and varies based on the specific situation.

About this task

The following problems might occur after a conditional restart of DB2:

- Some amount of work is left incomplete.
- Some data is left inconsistent.
- Information about the status of objects within the DB2 subsystem is made unusable.

Inconsistencies in a distributed environment

In a distributed environment, when a DB2 subsystem restarts, DB2 indicates its restart status and the name of its recovery log to the systems that it communicates with. The two possible conditions for restart status are warm and cold.

A cold status is associated with a *cold start*, which is a process by which a DB2 subsystem restarts without processing any log records. DB2 has no memory of previous connections with its partner. The general process that occurs with a cold start in a distributed environment is as follows:

1. The partner (for example CICS) accepts the cold start connection and remembers the recovery log name of the DB2 subsystem that experienced the cold start.
2. If the partner has indoubt thread resolution requirements with the cold-starting DB2 subsystem, those requirements cannot be satisfied.
3. The partner terminates its indoubt resolution responsibility with the cold-starting DB2 subsystem. However, as a participant, the partner has indoubt logical units of work that must be resolved manually.
4. Because the DB2 subsystem has an incomplete record of resolution responsibilities, DB2 attempts to reconstruct as much resynchronization information as possible.
5. DB2 displays the information that it was able to reconstruct in one or more DSNL438 or DSNL439 messages.
6. DB2 then discards the synchronization information that it was able to reconstruct and removes any restrictive states that are maintained on the object.

Resolving inconsistencies

In some problem situations, you need to determine what you must do in order to resolve any data inconsistencies that exist.

Procedure

To resolve inconsistencies:

1. Determine the scope of any inconsistencies that are introduced by the situation.
   a. If the situation is either a cold start that is beyond the current end of the log or a conditional restart that skips backout or forward log recovery, use the DSN1LOGP utility to determine what units of work have not been backed out and which objects are involved. For a cold start that is beyond the end of the log, you can also use DSN1LOGP to help identify any restrictive object states that have been lost.
   b. If a conditional restart truncates the log in a non-data sharing environment, recover all data and indexes to the new current point in time, and rebuild the data and indexes as needed. You need to recover or rebuild (or both recover and rebuild) the data and indexes because data and index updates might exist without being reflected in the DB2 log. When this situation occurs, a variety of inconsistency errors might occur, including DB2 abends with reason code 00C200C1.
2. Decide what approach to take to resolve inconsistencies by reading related topics about the approaches:
   - Recovery to a prior point of consistency
   - Restoration of a table space
   - Use of the REPAIR utility on the data

The first two approaches are less complex than, and therefore preferred over, the third approach.

3. If one or more of the following conditions are applicable, take image copies of all DB2 table spaces:
   - 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.

   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 allows 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. Therefore, you need to assume that all data recovery activity requires that portion of the log.

   A cold start might cause down-level page set errors, which you can find out about in different ways:
   - Message DSNB232I is sometimes displayed during DB2 restart, once for each down-level page set that DB2 detects. After you restart DB2, check the console log for down-level page set messages.
     - If a small number of those messages exist, run DSN1COPY with the RESET option to correct the errors to the data before you take image copies of the affected data sets.
     - If a large number of those messages exist, the actual problem is not that page sets are down-level but that the conditional restart inadvertently caused a high volume of DSNB232I messages. In this case, temporarily turn off down-level detection by turning off the DLDFREQ ZPARM.

     In either case, continue with step 4.
   - If you run the COPY utility with the SHRLEVEL REFERENCE option to make image copies, the COPY utility sometimes issues message DSNB232I about down-level page sets that DB2 does not detect during restart. If any of those messages were issued when you are making image copies, correct the errors, and continue making image copies of the affected data sets.
   - If you use some other method to make image copies, you will find out about down-level page set errors during normal operation. In this case, you need to correct the errors by using the information in "Recovering from a down-level page set problem" on page 537.
4. For any DB2 (catalog and directory) system table spaces that are inconsistent, recover them in the proper order. You might need to recover to a prior point in time, prior to the conditional restart.
5. For any objects that you suspect might be inconsistent, resolve the database inconsistencies before proceeding.
   - First, resolve inconsistencies in DB2 system databases DSNDB01 and DSNDB06. Catalog and directory inconsistencies need to be resolved before inconsistencies in other databases because the subsystem databases describe all other databases, and access to other databases requires information from DSNDB01 and DSNDB06.

- If you determine that the existing inconsistencies involve indexes only (not data), use the REBUILD INDEX utility to rebuild the affected indexes. Alternatively, you can use the RECOVER utility to recover the index if rebuilding the indexes is not possible.
- For a table space that cannot be recovered (and is thus inconsistent), determine the importance of the data and whether it can be reloaded. If the data is not critical or can be reloaded, drop the table after you restart DB2, and reload the data rather than trying to resolve the inconsistencies.

Related concepts:

Recovery of data to a prior point in time

Related tasks:

Recovering catalog and directory objects (DB2 Utilities)

Related reference:

LEVELID UPDATE FREQ field (DLDFREQ subsystem parameter) (DB2 Installation and Migration)

DSN1COPY (DB2 Utilities)

RECOVER (DB2 Utilities)

Restoring the table space:

You can restore the table space by reloading data into it or by re-creating the table space, which requires advance planning. Either of these methods is easier than using REPAIR.

About this task

Reloading the table space is the preferred approach, when it is possible, because reloading is easier and requires less advance planning than re-creating a table space. Re-creating a table space involves dropping and then re-creating the table space and associated tables, indexes, authorities, and views, which are implicitly dropped when the table space is dropped. Therefore, re-creating the objects means that you need to plan ahead so that you will be prepared to re-establish indexes, views, authorizations, and the data content itself.

Restriction:

You cannot drop DB2 system tables, such as the catalog and directory. For these system tables, follow one of these procedures instead of this one:

- "Recovery of data to a prior point in time" on page 424
- "Using the REPAIR utility on inconsistent data" on page 533

Procedure

To restore the table space:

1. Decide whether you can reload the table space or must drop and re-create it.
   - If you can reload the table space, run the appropriate LOAD utility jobs to do so; specify the REPLACE option. After you load the content of the table space, skip to step 6 on page 533.
   - If you cannot reload the table space, continue with step 2.
2. Issue an SQL DROP TABLESPACE statement for the table space that is suspected of being involved in the problem.

3. Re-create the table space, tables, indexes, synonyms, and views by using SQL CREATE statements.
4. Grant access to these objects the same way that access was granted prior to the time of the error.
5. Reconstruct the data in the tables.
6. Run the RUNSTATS utility on the data.
7. Use the COPY utility to acquire a full image copy of all data.
8. Use the **REBIND** command on all plans that use the tables or views that are involved in this activity.

Related concepts:

Recovery of data to a prior point in time

Related tasks:

Using the REPAIR utility on inconsistent data

Related reference:

LOAD (DB2 Utilities)

COPY (DB2 Utilities)

Using the REPAIR utility on inconsistent data:

You can resolve inconsistencies with the REPAIR utility. However, using REPAIR is not recommended unless the inconsistency is limited to a small number of data or index pages.

About this task

DB2 does not provide a mechanism to automatically inform users about data that is physically inconsistent or damaged. When you use SQL to access data that is physically damaged, DB2 issues messages to indicate that data is not available due to a physical inconsistency.

However, DB2 includes several utilities that can help you identify data that is physically inconsistent before you try to access it. These utilities are:

- CHECK DATA
- CHECK INDEX
- CHECK LOB
- COPY with the CHECKPAGE option
- DSN1COPY with the CHECK option

Attention: If you decide to use this method to resolve data inconsistencies, use extreme care. Use of the REPAIR utility to correct inconsistencies requires in-depth knowledge of DB2 data structures. Incorrect use of the REPAIR utility can cause further corruption and loss of data. Read this topic carefully because it contains information that is important to the successful resolution of the inconsistencies.

Recommendation: Avoid using this procedure if you are experiencing extensive data inconsistency because it is more time-consuming and complex (and therefore prone to error) than recovering to a point in time or re-creating the table spaces. If possible, use those alternative procedures instead.

Restrictions:

- Although the DSN1LOGP utility can identify page sets that contain inconsistencies, this utility cannot identify the specific data modifications that are involved in the inconsistencies within a given page set.
- Any pages that are on the logical page list (perhaps caused by this restart) cannot be accessed by using the REPAIR utility.

Procedure

To use the REPAIR utility to resolve the inconsistency:

1. Issue the following command to start DB2 and allow access to data:

   ```
   START DATABASE (dbase) SPACENAM (space) ACCESS(FORCE)
   ```

   In this command, *space* identifies the table space that is involved.
2. If any system data is inconsistent, use the REPAIR utility to resolve those inconsistencies. DB2 system data (such as data that is in the catalog and directory) exists in interrelated tables and table spaces. Data in DB2 system databases cannot be modified with SQL, so use of the REPAIR utility is necessary to resolve the inconsistencies that are identified.
3. Determine if you have any structural violations in data pages. 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 continue to exist after conditional restart.
4. Use the DSN1COPY utility with the CHECK option to identify any violations that you detected in the previous step, and then resolve the problems, possibly by recovering or rebuilding the object or by dropping and re-creating it.
5. Examine the various types of pointers that DB2 uses to access data (indexes, hashes, and links), and identify inconsistencies that need to be manually corrected.

   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, data pages might be modified without update of the corresponding pointers. When this occurs, one of the following actions might 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 that identifies 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 by using the damaged hash or link pointer. The data might, however, be visible and accessible by some functions, such as SQL functions that use another pointer that was not damaged. This situation can result in inconsistencies.

   If a row that contains 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 might 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 are issued to identify the condition and the page that contains the pointer.

6. Use the REPAIR utility to resolve any inconsistencies that you detected in the previous step.
7. Reset the log RBA in every data and index page set that are to be corrected with this procedure by using the DSN1COPY RESET option. This step is necessary for the following reason. If the log was truncated, changing data by using the REPAIR utility can cause problems. 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 that contains a log RBA that is higher than the current end of the log.
8. When all known inconsistencies have been resolved, take full image copies of all modified table spaces, in order to use the RECOVER utility to recover from any future problems. This last step is imperative.

Related concepts:

Recovery of data to a prior point in time

Obtaining DB2 Diagnosis Guide and Reference ()

Related tasks:

Restoring the table space

Related reference:

REPAIR (DB2 Utilities)

Recovering from DB2 database failure

If a DB2 failure occurs because of an allocation or open problem, you can recover from this situation.

Symptoms

The symptoms vary based on whether the failure was an allocation or an open problem:

Allocation problem

The following message indicates an allocation problem:

```
DSNB207I - DYNAMIC ALLOCATION OF DATA SET FAILED.
           REASON=rrrr DSNAME=dsn
```

The *rrrr* is a z/OS dynamic allocation reason code.

Open problem

The following messages indicate an open problem:

```
IEC161I  rc[(sfi)] - ccc, iii, sss, ddn,
       ddd, ser, xxx, dsn, cat

DSNB204I - OPEN OF DATA SET FAILED. DSNAME = dsn
```

In the IEC161I message:

rc Is a return code.
sfi Is subfunction information, which is displayed only with certain return codes.
ccc Is a function code.
iii Is a job name.
sss Is a step name.
ddn Is a DD name.
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.

Environment

When this type of problem occurs:

- The table space is automatically stopped.
- Programs receive a -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.

Resolving the problem

Operator response:

1. Check reason code, and correct problems.
2. Ensure that drives are available for allocation.
3. Enter the command **START DATABASE**.

Related reference

MVS Authorized Assembler Services Guide

Recovering a DB2 subsystem to a prior point in time

You can recover a DB2 subsystem and data sharing group to a prior point in time by using the BACKUP SYSTEM and RESTORE SYSTEM utilities.

About this task

In this recovery procedure, you create and populate a table that contains data that is both valid and invalid. You need to restore your DB2 subsystem to a point in time before the invalid data was inserted into the table, but after the point in time when the valid data was inserted. Also, you create an additional table space and table that DB2 will re-create during the log-apply phase of the restore process.

Procedure

To insert data into a table, determine the point in time that you want to recover to, and then recover the DB2 subsystem to a prior point in time:

1. Issue the **START DB2** command to start DB2 and all quiesced members of the data sharing group. Quiesced members are ones that you removed from the data sharing group either temporarily or permanently. Quiesced members remain dormant until you restart them.
2. Issue SQL statements to create a database, a table space, and two tables with one index for each table.
3. Issue the BACKUP SYSTEM DATA ONLY utility control statement to create a backup copy of only the database copy pool for a DB2 subsystem or data sharing group.
4. Issue an SQL statement to first insert rows into one of the tables, and then update some of the rows.
5. Use the LOAD utility with the LOG NO attribute to load the second table.
6. Issue SQL statements to create an additional table space, table, and index in an existing database. DB2 will re-create the additional table space and table during the log-apply phase of the restore process.

7. Issue the SET LOG SUSPEND command or the SET LOG RESUME command to obtain a log truncation point, *logpoint1*, which is the point you want to recover to. For a non-data sharing group, use the RBA value. For a data sharing group, use the lowest log record sequence number (LRSN) from the active members.

   The following example shows sample output for the SET LOG SUSPEND command:

```
 09.47.52           @set log suspend
 09.47.52 STC00066  DSN9022I  @ DSNJC001 '-SET LOG' NORMAL COMPLETION
*09.47.55 STC00066 *DSNJ372I  @ DSNJC09A UPDATE ACTIVITY HAS BEEN
*SUSPENDED FOR VA1A AT RBA 00004777B710, LRSN C31E5141E0C4, PRIOR
*CHECKPOINT RBA 000047778090
```

8. Issue an SQL statement to first insert rows into one of the tables and then to update and delete some rows.
9. Issue the **STOP DB2** command to stop DB2 and all active members of the data sharing group.
10. Run the DSNJU003 change log inventory utility to create a SYSPITR CRCR record (`CRESTART CREATE SYSPITR=logpoint1`). The log truncation point is the value that you obtained from issuing either the **SET LOG SUSPEND** command, or the **SET LOG RESUME** command.
11. For a data sharing group, delete all of the coupling facility structures.
12. Issue the **START DB2** command to restart DB2 and all members of the data sharing group.
13. Run the RESTORE SYSTEM utility. For a data sharing group, this utility can be run only on one member. If the utility stops and you must restart it, you can restart the utility only on the member on which it was initially run.
14. After the RESTORE SYSTEM utility completes successfully, issue the **STOP DB2** command to stop DB2 and all active members of the data sharing group. The DB2 subsystem resets to RECOVER-pending status.
15. Issue the **START DB2** command to restart DB2 and all members of the data sharing group.
16. Issue the **DISPLAY** command to identify the utilities that are active and the objects that are restricted. For example:

```
-DIS UTIL(*)
-DIS DB(DSNDB01) SP(*)
-DIS DB(DSNDB06) SP(*) LIMIT(*)
-DIS DB(DSNDB06) SP(*) LIMIT(*)RESTRICT
```

17. Stop all of the active utilities that you identified in the previous step.
18. Recover any objects that are in RECOVER-pending status or REBUILD-pending status from the table that you created in step 6 on page 536.

Recovering from a down-level page set problem

When using a stand-alone utility or a non-DB2 utility, you might inadvertently replace a DB2 page set with an incorrect or outdated copy. This type of copy is called *down-level*. Using a down-level page set can cause complex problems; therefore, you need to recover from this situation.

Symptoms

The following message is issued:

```
DSNB232I csect-name - UNEXPECTED DATA SET LEVEL ID ENCOUNTERED
```

The message also contains the level ID of the data set, the level ID that DB2 expects, and the name of the data set.

Causes

A down-level page set can be caused by:

- A DB2 data set is inadvertently replaced by an incorrect or outdated copy. Usually this happens in conjunction with use of a stand-alone or non-DB2 utility, such as DSN1COPY or DFSMShsm.
- A cold start of DB2 occurs.
- A VSAM high-used RBA of a table space becomes 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 the following operations, which do not use the level ID data:

- LOAD REPLACE
- RECOVER
- REBUILD INDEX
- DSN1COPY
- DSN1PRNT

Environment

- If the error was reported during mainline processing, DB2 sends a "resource unavailable" SQLCODE and a reason code to the application to explain the error.
- If the error was detected while a utility was processing, the utility generates a return code 8.

Diagnosing the problem

Determine whether the message was issued during restart or at some other time during normal operation. This information is important for determining what steps to take below

Resolving the problem

System programmer response: The actions that you need to do to recover depend on when the message was issued:

- If the message was issued during restart, take one of the following actions:
  - Replace the data set with one that is at the proper level, by 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 output identifies the level ID.
  - Recover the data set to the current time, or to a prior time, by using the RECOVER utility.
  - Replace the contents of the data set, by using LOAD REPLACE.
- If the message was issued during normal operation (not during restart):
  1. Take one of the actions that are listed for situations when the message was issued during restart.
  2. Accept the down-level data set by changing its level ID. You can use the REPAIR utility with the LEVELID statement. (You cannot use the LEVELID option in the same job step with any other REPAIR utility control statement.)

     Attention: If you accept a down-level data set or disable down-level detection, your data might be inconsistent.

Related system programmer actions: Consider taking the following actions, which might help you minimize or deal with down-level page set problems in the future:

- To control how often the level ID of a page set or partition is updated, specify a value between 0 and 32767 on the LEVELID UPDATE FREQ field of panel DSNTIPL.
- 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.

Related reference

LEVELID UPDATE FREQ field (DLDFREQ subsystem parameter) (DB2 Installation and Migration)

DSN1COPY (DB2 Utilities)

DSN1PRNT (DB2 Utilities)

LOAD (DB2 Utilities)

RECOVER (DB2 Utilities)

REPAIR (DB2 Utilities)

Recovering from a problem with invalid LOBs

If a LOB table space is defined with LOG NO and you need to recover that table space, you can recover the LOB data to the point at which you made your last image copy of the table space.

About this task

Unless your LOBs are fairly small, specifying LOG NO for LOB objects is recommended for the best performance. However, to maximize recoverability, specifying LOG YES is recommended. The performance cost of logging exceeds the benefits that you can receive from logging such large amounts of data. If no changes are made to LOB data, the logging cost 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.

Procedure

To recover LOB data from a LOB table space that is defined with LOG NO:

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 auxiliary warning status, which indicates that some of your LOBs are invalid. Applications that try to retrieve the values of those LOBs 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.

GUPI

Fix the invalid LOBs, by updating the LOBs or setting them to the null value. For example, suppose that 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 DSN8A10. EMP_PHOTO_RESUME
  SET RESUME = :hvlob
  WHERE EMP_ROWID = ROWID(X'C1BDC4652940D40A81C201AA0A28');
```

GUPI

Recovering from table space I/O errors

You can recover a table space after I/O errors have occurred and caused the table space to fail.

Symptoms

The following message is issued, where *dddddddd* is a table space name:

```
DSNU086I   DSNUCDA1 READ I/O ERRORS ON SPACE=dddddddd.
         DATA SET NUMBER=nnn.
         I/O ERROR PAGE RANGE=aaaaaa, bbbbbb.
```

Any table spaces that are identified in DSNU086I messages must be recovered. Follow the steps later in this topic.

Environment

DB2 remains active.

Resolving the problem

Operator response:

1. Fix the error range:
   a. Use the command **STOP DATABASE** to stop the failing table space.
   b. Use the command **START DATABASE ACCESS (UT)** to start the table space for utility-only access.
   c. Start a RECOVER utility step to recover the error range by using the following statement:

      ```
      DB2 RECOVER (dddddddd) ERROR RANGE
      ```

      If you receive message DSNU086I again to indicate that the error range recovery cannot be performed, continue with step 2.
   d. 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.
2. If error range recovery fails because of a hardware problem:
   a. Use the command **STOP DATABASE** to stop the table space or table space partition that contains the error range. As a result of this command, all in-storage data buffers that are associated with the data set are externalized to ensure data consistency during the subsequent steps.

b. Use the INSPECT function of the IBM Device Support Facility, ICKDSF, to check for track defects and to assign alternate tracks as necessary. Determine the physical location of the defects by analyzing the output of messages DSNB224I, DSNU086I, and IOS000I. These messages are displayed on the system operator's console at the time that the error range was created. If damaged storage media is suspected, request assistance from IBM Hardware Support before proceeding.

c. Use the command **START DATABASE** to start the table space with ACCESS(UT) or ACCESS(RW).

d. Run the RECOVER utility with the ERROR RANGE option. Specify an error range that, from image copies, locates, allocates, and applies the pages within the tracks that are affected by the error ranges.

Related information

Device Support Facilities (ICKDSF) Device Support Facilities (ICKDSF) User's Guide and Reference

Recovering from DB2 catalog or directory I/O errors

When the DB2 catalog or directory fails because of I/O errors, you need to recover from this situation so that processing can return to normal.

Symptoms

The following message is issued, where *dddddddd* is the name of the table space from the catalog or directory that failed (for example, SYSIBM.SYSCOPY):

```
DSNU086I DSNUCDA1 READ I/O ERRORS ON SPACE=dddddddd.
         DATA SET NUMBER=NNN.
         I/O ERROR PAGE RANGE=aaaaaa, bbbbbb
```

This message can indicate either read or write errors. You might also receive a DSNB224I or DSNB225I message, which indicates an input or output error for the catalog or directory.

Environment

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 perform 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 to indicate that the resource is unavailable.

Resolving the problem

Operator response: Recover each table space in the failing DB2 catalog or directory. If multiple table spaces need to be recovered, recover them in the recommended order as defined in the information about the RECOVER utility.

1. Stop the failing table spaces.
2. Determine the name of the data set that failed by using one of the following methods:
   - 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 that is 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:
    DSNC100.DSNDBC.DSNDB06.*dddddddd*.I0001.A001
    The *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:
    DSNC100.DSNDBC.DSNDB01.*dddddddd*.I0001.A001
    The *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 might be different.

3. Use the access method services **DELETE** command to delete the data set, specifying the fully qualified data set name.
4. After the data set is deleted, use the access method services **DEFINE** command with the REUSE parameter 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.
5. Issue the command **START DATABASE ACCESS(UT)**, naming the table space that is 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.

Related tasks

Recovering catalog and directory objects (DB2 Utilities)

Recovering from integrated catalog facility failure

Sometimes VSAM volume data sets might be out of space or destroyed. Also, you might experience problems with other VSAM data sets being out of space or unable to be extended any further.

Symptoms

The symptoms for integrated catalog facility problems vary according to the underlying problems.

Recovering VSAM volume data sets that are out of space or destroyed

If the VSAM volume data set (VVDS) is out of space or destroyed, you can recover from the situation. You can recover by using DB2 commands, DB2 utilities, and access method services.

Symptoms

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
```

VSAM might 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.

Environment

Your program is terminated abnormally, and one or more messages are issued.

Resolving the problem

Operator response: Begin by determining whether the VSAM volume data set is out of space or has been destroyed. Then follow these steps:

1. Determine the names of all table spaces that reside 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.
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 you cannot use the COPY utility, continue with this procedure. Be aware that processing time increases because more information must be 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 that resides on that volume.
5. Use access method services to recover all non-DB2 data sets that resides on that volume.
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 delete and redefine table spaces that are STOGROUP-defined because DB2 takes care of them automatically.
7. Issue the DB2 **START DATABASE** command to restart all the table spaces that were 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.

Related tasks

Backing up and recovering your data

Related information

DFSMS Access Method Services Commands

DFSMS Managing Catalogs

DSNP012I (DB2 Messages)

Recovering from out-of-disk-space or extent limit problems

When a volume on which a data set is stored has insufficient space, or when the data set reaches its maximum size or its maximum number of VSAM extents, you can recover from this situation.

Symptoms

The symptoms vary based on the specific situation. The following messages and codes might be issued:

- DSNP007I
- DSNP001I
- -904 SQL return code (SQLSTATE '57011')

Environment

For a demand request failure during restart, the object that is 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.

Diagnosing the problem

Read the following descriptions of possible problems, and determine which problem you are experiencing.

- Extend request failures: When an insert or update requires additional space, but the space is not available in the current table space or index space, DB2 issues the following message:

  ```
  DSNP007I - DSNPmmmm - EXTEND FAILED FOR
             data-set-name. RC=rrrrrrrr
             CONNECTION-ID=xxxxxxxx,
             CORRELATION-ID=yyyyyyyyyyy
             LUWID-ID=logical-unit-of-work-id=token
  ```

- Look-ahead warning: A look-ahead warning occurs when enough space is available for a few inserts or 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, DB2 issues the following message:

  ```
  DSNP001I - DSNPmmmm - data-set-name IS WITHIN
             nK BYTES OF AVAILABLE SPACE.
             RC=rrrrrrrr
             CONNECTION-ID=xxxxxxxx,
             CORRELATION-ID=yyyyyyyyyyy
             LUWID-ID=logical-unit-of-work-id=token
  ```

Resolving the problem

What you need to do depends on your particular circumstances.

- In most cases, if the data set has not reached its maximum size, you can enlarge it. For the maximum sizes of data sets, see Limits in DB2 for z/OS (DB2 SQL).
- If the data set has reached its maximum size, you need to follow the appropriate procedure, depending on the situation you face.

Extending a data set

If a user-defined data set reaches the maximum number of VSAM extents, you can extend the data set by adding volumes.

Procedure

To extend a user-defined data set:

1. If possible, delete unneeded data on the current volume.

2. If deleting data from the volume does not solve the problem, add volumes to the data set in one of the following ways:
   - 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.
   - If the data set is not defined in a DB2 storage group, add volumes to the data set by using the access method services **ALTER ADDVOLUMES** command.

Enlarging a fully extended user-managed data set

If a user-managed data set reaches the maximum number of VSAM extents, you can enlarge the data set.

Procedure

To enlarge a user-managed data set:

1. To allow for recovery in case of failure during this procedure, ensure that you have a recent full image copy of your table spaces and 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 supported object.
3. Delete the last data set by using access method services.
4. Redefine the data set, and enlarge it as necessary. The object must be a user-defined linear data set. The limit is 32 data sets if the underlying table space is not defined as LARGE or with a DSSIZE parameter, and the limit is 4096 for objects with greater than 254 parts. For a nonpartitioned index on a table space that is defined as LARGE or with a DSSIZE parameter, the maximum is MIN(4096, 2^{32} / (*index piece size*/*index page size*)).
5. Issue the command **START DATABASE ACCESS (UT)** to start the object for utility-only access.
6. To recover the data set that was redefined, use the RECOVER utility 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).

   The RECOVER utility enables you to specify a single data set number for a table space. Therefore, you need to redefine and recover only the last data set (the one that needs extension). This approach can be better than using the REORG utility 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, use the REBUILD INDEX utility.
7. Issue the command **START DATABASE** to start the object for either RO or RW access, whichever is appropriate.

Related reference

-START DATABASE (DB2) (DB2 Commands)

-STOP DATABASE (DB2) (DB2 Commands)

RECOVER (DB2 Utilities)

REBUILD INDEX (DB2 Utilities)

Enlarging a fully extended DB2-managed data set

If a DB2-managed data set reaches the maximum number of VSAM extents, you can enlarge the data set.

Procedure

To enlarge a DB2-managed data set:

1. Use the SQL statement ALTER TABLESPACE or ALTER INDEX with a USING clause. (You do not need to stop the table space before you use ALTER TABLESPACE.) You can give new values of PRIQTY and SECQTY in either the same or a new DB2 storage group.
2. Use one of the following procedures. 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 spaces: Run one of the following utilities on the table space: REORG, RECOVER, or LOAD REPLACE.
   - For LOB table spaces that are defined with LOG YES: Run the RECOVER utility on the table space.
   - For LOB table spaces that are defined with LOG NO:
     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.

Adding a data set

If a user-defined simple data set reaches its maximum size, you can use access method services to define another data set.

Procedure

To add another data set:

1. Use access method services to define another data set. The name of the new data set must follow the naming sequence 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 ends with A001, the next name must end with A002, and so on. Also, be sure to add either the character "I" or the character "J" to 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 to indicate the cause of the problem.
2. If necessary, correct the problem (identified in the access method services messages) to obtain additional space.

Redefining a partition (index-based partitioning)

Sometimes each partition in a partitioned object is restricted to a single data set. If the data set reaches its maximum size, you need to redefine the partitions. Redefining a partition in an index-based partitioning environment is different than in a table-based partitioning environment.

Procedure

To redefine the partitions in an index-based partitioning environment:

1. Use the ALTER INDEX ALTER PARTITION statement to alter the key range values of the partitioning index.
2. Use the REORG utility with inline statistics on the partitions that are affected by the change in key range.

3. Use the RUNSTATS utility on the nonpartitioned indexes.
4. Rebind the dependent packages and plans.

Redefining a partition (table-based partitioning)

Sometimes each partition in a partitioned object is restricted to a single data set. If the data set reaches its maximum size, you need to redefine the partitions. Redefining a partition in a table-based partitioning environment is different than in an index-based partitioning environment.

Procedure

To redefine the partitions in a table-based partitioning environment:

1. Use the SQL statement ALTER TABLE ALTER PARTITION to alter the partition boundaries.
2. Use the REORG utility with inline statistics on the partitions that are affected by the change in partition boundaries.
3. Use the RUNSTATS utility on the indexes.
4. Rebind the dependent packages and plans.

Enlarging a fully extended data set for the work file database

If you have an out-of-disk-space or extent limit problem with the work file database (DSNDB07), you need to add space to the data set.

Procedure

To enlarge a fully extended data set for the work file database:

Add space to the DB2 storage group, choosing one of the following approaches:

- Use SQL to create more table spaces in database DSNDB07.
- 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.

Recovering from referential constraint violation

When a referential constraint is violated, the table space is available for some actions, but you cannot run certain utilities or use SQL to update the data in the table space until you recover from this situation.

Symptoms

One of the following messages is issued at the end of utility processing, depending on whether 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
```

Causes

DB2 detected one or more referential constraint violations.

Environment

The table space is still generally 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.

Resolving the problem

Operator response:

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. Consider these recommendations:
   - If you do not believe that violations exist, specify DELETE NO. If violations do not exist, specifying DELETE NO resets the CHECK-pending status; however, if violations do exist, the status is not reset.
   - If you believe that violations exist, specify the DELETE YES option and an appropriate exception table. Specifying DELETE YES results in deletion of all rows that are 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 that are 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, using ACCESS (FORCE) is not recommended because it does not correct the underlying violations of referential constraints.

Related reference

CHECK DATA (DB2 Utilities)

Recovering from distributed data facility failure

You can recover from various problems that occur for the distributed data facility (DDF).

Symptoms

The symptoms for DDF failures vary based on the precise problems. The symptoms include messages, SQL return codes, and apparent wait states.

Recovering from conversation failure

A VTAM APPC or TCP/IP conversation might fail during or after allocation. The conversation is not available for use until you recover from the situation.

Symptoms

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 that detect a failure from that LU mode or IP address are suppressed until communications to the LU that uses that mode are successful.

DB2 returns messages DSNL501I and DSNL502I. Message DSNL501I usually means that the other subsystem is not operational. When the error is detected, it is reported by a console message, and the application receives an SQL return code.

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.

Environment

The application can choose to request rollback or commit, both of which deallocate 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 deallocation process of the conversation are reported through messages, but they do not stop the commit or rollback processing. If the conversation that is used for the commit or rollback message fails, the error is reported. If the error occurred during a commit process and if the remote database access was read-only, the commit process continues. Otherwise the commit process is rolled back.

Diagnosing the problem

System programmer response: Review the VTAM or TCP/IP return codes, and possibly 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.

Resolving the problem

Operator response: Correct the cause of the unavailable-resource condition by taking the action that is required by the diagnostic messages that are displayed on the console.

Related concepts

SQL codes (DB2 Codes)

Related information

z/OS Communications Server: SNA Messages

Recovering from communications database failure

You need to recover the communications database (CDB) when a failure occurs during an attempt to access the CDB.

Symptoms

A DSNL700I message, which indicates that a resource-unavailable condition exists, is sent to the console. Other messages that describe the cause of the failure are also sent to the console.

Environment

If the distributed data facility (DDF) has already started when an individual CDB table becomes unavailable, DDF does not terminate. Depending on the severity of the failure, threads are affected as follows:

- The threads receive a -904 SQL return code (SQLSTATE '57011') with resource type 1004 (CDB).

- The threads continue using VTAM default values.

The only threads that receive a -904 SQL return code are those that access locations that have not had any prior threads. DB2 and DDF remain operational.

Resolving the problem

Operator response:

1. Examine the messages to determine the source of the error.
2. Correct the error, and then stop and restart DDF.

Recovering from a problem with a communications database that is incorrectly defined

You need to recover from a situation in which the communications database (CDB) is not correctly defined. This problem occurs when distributed data facility (DDF) is started and the DB2 catalog is accessed to verify the CDB definitions.

Symptoms

A DSNL701I, DSNL702I, DSNL703I, DSNL704I, or DSNL705I message is issued to identify the problem. Other messages that describe the cause of the failure are also sent to the console.

Environment

DDF fails to start. DB2 continues to run.

Resolving the problem

Operator response:

1. Examine the messages to determine the source of the error.
2. Correct the error, and then restart DDF.

Recovering from database access thread failure

When a database access thread is deallocated, a conversation failure occurs, and you need to recover from this situation.

Symptoms

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:

- 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).

Environment

Normal DB2 error recovery mechanisms apply, with the following exceptions:

- Errors that are encountered in the functional recovery routine are automatically converted to rollback situations. The allied thread experiences conversation failures.
- Errors that occur during commit, rollback, 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 experiences conversation failures.

Diagnosing the problem

System programmer response: Collect all diagnostic information that is related to the failure at the serving site. For a DB2 database access thread (DBAT), a dump is produced at the server.

Resolving the problem

Operator response: Communicate with the operator at the other site to take the appropriate corrective action, regarding the messages that are displayed at both the requesting and responding sites. Ensure that you and operators at the other sites gather the appropriate diagnostic information and give it to the programmer for diagnosis.

Recovering from VTAM failure

When VTAM terminates or fails, you need to recover from the situation.

Symptoms

VTAM messages and DB2 messages are issued to indicate that distributed data facility (DDF) is terminating and to explain why.

Causes
Environment

DDF terminates. An abnormal VTAM failure or termination causes DDF to issue a **STOP DDF MODE(FORCE)** command. The VTAM commands **Z NET,QUICK** and **Z NET,CANCEL** cause an abnormal VTAM termination. A **Z NET,HALT** causes a **STOP DDF MODE(QUIESCE)** to be issued by DDF.

Resolving the problem

Operator response: Correct the condition that is described in the messages that are received at the console, and restart VTAM and DDF.

Recovering from TCP/IP failure

When TCP/IP terminates or fails, you need to recover from this situation.

Symptoms

TCP/IP messages and DB2 messages are issued to indicate that TCP/IP is unavailable.

Environment

Distributed data facility (DDF) periodically attempts to reconnect to TCP/IP. If the TCP/IP listener fails, DDF automatically tries to re-establish the TCP/IP listener for the DRDA SQL port or the resync port every three minutes. TCP/IP connections cannot be established until the TCP/IP listener is re-established.

Resolving the problem

Operator response:

1. Examine the messages that are received at the console to determine the cause of the problem.
2. Correct the condition.
3. Restart TCP/IP. You do not need to restart DDF after a TCP/IP failure.

Recovering from remote logical unit failure

When a series of conversation or change number of sessions (CNOS) failures occur from a remote logical unit (LU), you need to recover from this situation.

Symptoms

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 DSNL501I is displayed to indicate that the CNOS request cannot be negotiated. Message DSNL500I is issued only once for all the SQL conversations that fail as a result 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.

Environment

Any application communications with a failed LU receives a message to indicate a resource-unavailable condition. Any attempt to establish communication with such an LU fails.

Resolving the problem

System programmer response: Communicate with the other involved sites regarding the unavailable-resource condition, and request that appropriate corrective action be taken. If a DSNL502 message is received, activate the remote LU, or ask another operator to do so.

Recovering from an indefinite wait condition

If a distributed thread is hung, an application might be in an indefinite wait condition. For example, an allied thread might wait indefinitely for a response from a remote server location. Another example is a database access thread that waits for a new request from the remote requester location.

Symptoms

An application is in an indefinitely long wait condition. This can cause other DB2 threads to fail due to resources that are held by the waiting thread. DB2 sends an error message to the console, and the application program receives an SQL return code.

Environment

DB2 does not respond.

Diagnosing the problem

Operator response: To check for very long waits, look to see if the conversation timestamp is changing from the last time it was used. If it is changing, the conversation thread is not hung, but it is taking more time for a long query. Also, look for conversation state changes, and determine what they mean.

Resolving the problem

Operator response:

1. Use the **DISPLAY THREAD** command with the LOCATION and DETAIL options to identify the LUWID and the session allocation for the waiting thread.
2. Use the **CANCEL DDF THREAD** command to cancel the waiting thread.
3. 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* or use the TCP/IP DROP command. For instructions on how to use the VTAM commands and TCP/IP commands, see -CANCEL THREAD (DB2) (DB2 Commands).

Related tasks

Canceling threads

Recovering database access threads after security failure

During database access thread allocation, the remote site might not have the proper security to access DB2 through distributed data facility (DDF). When this happens, you can recover from the situation.

Symptoms

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, and RCSEC=5. These return codes indicate that a security violation has occurred. The server has deallocated the conversation because the user is not allowed to access the server. For conversations that use DRDA access, LU 6.2 communications protocols present specific reasons for why the user access failed, and these reasons are communicated 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.

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 DRDA access, SQLCODE –30082 is returned.

Causes

This problem is caused by a remote user who attempts to access DB2 through DDF without the necessary security authority.

Resolving the problem

Operator response:

1. Read about the DB2 code 00D3103D.
2. Take the appropriate action:
   - If the security failure involves a DB2 database access thread, provide the DSNL030I message to the system programmer.
   - If the security failure does not involve a DB2 server, work with the operator or programmer at the server to get diagnostic information that is needed by the system programmer.

Related information

00D3103D (DB2 Codes)

Performing remote-site disaster recovery

When your local system experiences damage or disruption that prevents recovery from that site, you can recover by using a remote site that you have set up for this purpose.

Symptoms

The specific symptoms of a disaster that affects your local system hardware vary, but when this happens, the affectedDB2 subsystem is not operational.

Causes

Your local system hardware has suffered physical damage.

Resolving the problem

System programmer response: Coordinate the activities that are detailed in "Restoring data from image copies and archive logs."

Operator response: At the remote-site, the disaster-recovery procedures differ from other recovery procedures because you cannot use the hardware at your local DB2 site to recover data. Instead, you use hardware at a remote site to recover after a disaster by using one of a variety of methods.

Recovering from a disaster by using system-level backups

If you have recent system-level backups, you can use those backups along with one of several utilities to recover after a disaster.

Procedure

To recover from a disaster by using system-level backups:

For a remote site recovery procedure where tape volumes that contain system data are sent from the production site, specify the dump class that is available at the remote site by using the following installation options on installation panel DSNTIP6:

- Either RESTORE FROM DUMP or RECOVER FROM DUMP
- DUMP CLASS NAME

Restoring data from image copies and archive logs

Follow the appropriate procedure for restoring from image copies and archive logs, depending on whether you are in a data sharing environment. Both procedures assume that all logs, copies, and reports are available at the recovery site.

Related information

DFSMS Access Method Services Commands

Restoring data in a non-data sharing environment

If you are in a non-data sharing environment, you might need to recover from a disaster by restoring data from image copies and logs. The procedure that you follow assumes that all logs, image copies, and reports are available at the recovery site.

Procedure

To recover from a disaster in a non-data sharing environment by using image copies and archive logs:

1. If an integrated catalog facility catalog does not already exist, run job DSNTIJCA to create a user catalog.
2. Use the access method services **IMPORT** command to import the integrated catalog facility catalog.
3. Restore DB2 libraries. Some examples of libraries that you might need to restore include:
   - DB2 SMP/E libraries
   - User program libraries
   - User DBRM libraries
   - DB2 CLIST libraries
   - DB2 libraries that contain customized installation jobs

- JCL for creating user-defined table spaces

4. Use IDCAMS **DELETE NOSCRATCH** to delete all catalog and user objects. (Because step 2 on page 554 imports a user ICF catalog, the catalog reflects data sets that do not exist on disk.)
5. Obtain a copy of installation job DSNTIJIN, which 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.
6. Recover the BSDS:
   a. Use the access method services **REPRO** command to restore the contents of one BSDS data set (allocated in step 5). You can find the most recent BSDS image in the last file (archive log with the highest number) on the latest archive log tape.
   b. Determine the RBA range for this archive log by using 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.
   c. Delete the oldest archive log from the BSDS.
   d. Register this latest archive log tape data set in the archive log inventory of the BSDS that you just restored by using the change log inventory utility (DSNJU003). This step is necessary because the BSDS image on an archive log tape does not reflect the archive log data set that resides on that tape. After these archive logs are registered, use the print log map utility (DSNJU004) to list the contents of the BSDS.
   e. Adjust the active logs in the BSDS by using the change log inventory utility (DSNJU003), as necessary:
      1) To delete all active logs in the BSDS, use the DELETE option of DSNJU003. Use the BSDS listing that is produced in step 6d to determine the active log data set names.
      2) To add the active log data sets to the BSDS, use the NEWLOG statement of DSNJU003. Do not specify a STARTRBA or ENDRBA in the NEWLOG statement. This specification indicates to DB2 that the new active logs are empty.
   f. If you are using the DB2 distributed data facility, update the LOCATION and the LUNAME values in the BSDS by running the change log inventory utility with the DDF statement.
   g. List the new BSDS contents by using the print log map utility (DSNJU004). 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 are correct).
   h. If you are using dual BSDSs, make a copy of the newly restored BSDS data set to the second BSDS data set.
7. Optional: Restore archive logs to disk. Archive logs are typically stored on tape, but restoring them to disk might 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 by using the integrated catalog and not the unit or VOLSER that is 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 generates a report.

9. Examine the DSN1LOGP output, and identify any utilities that were executing at the end of the last archive log. Determine the appropriate recovery action to take on each table space that is involved in a utility job. If DSN1LOGP output showed that utilities are inflight (PLAN=DSNUTIL), examine SYSUTILX to identify the utility status and determine the appropriate recovery approach.
10. Modify DSNZP*xxx* parameters:
    a. Run the DSNTINST CLIST in UPDATE mode.
    b. To defer processing of all databases, select DATABASES TO START AUTOMATICALLY from panel DSNTIPB. Panel DSNTIPS opens. On panel DSNTIPS, type `DEFER` in the first field and `ALL` in the second field; then press **Enter**. You are returned to panel DSNTIPB.
    c. To specify where you are recovering, select OPERATOR FUNCTIONS from panel DSNTIPB. Panel DSNTIPO opens. From panel DSNTIPO, type `RECOVERYSITE` in the SITE TYPE field. Press **Enter** to continue.
    d. Optional: Specify which archive log to use by selecting OPERATOR FUNCTIONS from panel DSNTIPB. Panel DSNTIPO opens. From panel DSNTIPO, type `YES` in the READ COPY2 ARCHIVE 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* by using job DSNTIJUZ (produced by the CLIST started in the first step of this procedure).

       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 that are in process.
11. Create a conditional restart control record by using the change log inventory utility with one of the following forms of the CRESTART statement:
    - `CRESTART CREATE,ENDRBA=`*nnnnnnnn*`000`

      The *nnnnnnnn*000 equals a value that is one more than the ENDRBA of the latest archive log.
    - `CRESTART CREATE,ENDTIME=`*nnnnnnnnnnnn*

      The *nnnnnnnnnnnn* is the end time of the log record. Log records with a timestamp later than *nnnnnnnnnnnn* are truncated.
12. Enter the command **START DB2 ACCESS(MAINT)**.

You must enter this command, because real-time statistics are active and enabled; otherwise, errors or abends could occur during DB2 restart processing and recovery processing (for example, GRECP recovery, LPL recovery, or the RECOVER utility).

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 is issued for activity against the object that occurs during restart as a result of a unit of recovery or pending writes. In this case, the status that is 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 a state that DB2 does not expect.

00C90095
DB2 cannot access the page, because the table space or index space has not been recovered yet.

00C900A9
An attempt was made to allocate a deferred resource.

13. Resolve the indoubt units of recovery. The RECOVER utility, which you run in a subsequent step, fails 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 386. From an installation SYSADM authorization ID, enter the **RECOVER INDOUBT** command for all affected transactions.
14. Recover the catalog and directory. 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. Determine whether a utility was running at the time the latest archive log was created by entering the **DISPLAY UTILITY(*)** command, and record the name and current phase of any utility that is running. (You cannot restart a utility at the recovery site that was interrupted at the disaster site. You must use the **TERM UTILITY** command to terminate it. Use the **TERM UTILITY** command on a utility that is operating on any object except DSNDB01.SYSUTILX.)
    d. Run the DIAGNOSE utility with the DISPLAY SYSUTIL option. The output consists of information about each active utility, including the table space

name (in most cases). This is the only way to correlate the object name with the utility. Message DSNU866I gives information about the utility, and DSNU867I gives the database and table space name in USUDBNAM and USUSPNAM, respectively.

e. Use the **TERM UTILITY** command to terminate any utilities that are in progress on catalog or directory table spaces.

f. Recover the rest of the catalog and directory objects, starting with DBD01, in the order shown in the description of the RECOVER utility.

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. Use any method that you want to verify the integrity of the DB2 catalog and directory. Use the catalog queries in member DSNTESQ of data set DSN1010.SDSNSAMP after the work file database is defined and initialized.
17. If you use data definition control support, recover the objects in the data definition control support database.
18. If you use the resource limit facility, recover the objects in the resource limit control facility database.
19. Modify DSNZP*xxx* to restart all databases:
    a. Run the DSNTINST CLIST in UPDATE mode.
    b. From panel DSNTIPB, select DATABASES TO START AUTOMATICALLY. Panel DSNTIPS opens. Type `RESTART` in the first field and `ALL` in the second field, and press **Enter**. You are returned to DSNTIPB.
    c. Reassemble DSNZP*xxx* by using job DSNTIJUZ (produced by the CLIST started in step 3 on page 554).
20. Stop DB2.
21. Start DB2.
22. Make a full image copy of the catalog and directory.
23. Recover user table spaces and index spaces. If utilities were running on any table spaces or index spaces, see "What to do about utilities that were in progress at time of failure" on page 566. You cannot restart a utility at the recovery site that was interrupted at the disaster site. Use the **TERM UTILITY** command to terminate any utilities that are 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.DSNDB*x*.*dbname*.*psname*.*y*0001.*znnn*

  The *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.

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 the SQL statement 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-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 queries for which the results are known.

24. Make full image copies of all table spaces and indexes with the COPY YES attribute.
25. Finally, compensate for work that was lost since the last archive was created by rerunning online transactions and batch jobs.

What to do next

Determine what to do about any utilities that were in progress at the time of failure.

Related concepts

Preparations for disaster recovery

What to do about utilities that were in progress at time of failure

Related tasks

Defining data sets

Migration step 1: Actions to complete before migration (DB2 Installation and Migration)

Invoking the CLIST (DB2 Installation and Migration)

Recovering catalog and directory objects (DB2 Utilities)

Related reference

DSN1LOGP (DB2 Utilities)

Restoring data in a data sharing environment

If you are in a data sharing environment, you might need to recover from a disaster by restoring data from image copies and logs. The procedure that you follow assumes that all logs, image copies, and reports are available at the recovery site.

About this task

Additional recovery procedures for data sharing environments are also available.

Procedure

To recover from a disaster by using image copies and archive logs:

1. If you have information in your coupling facility from practice startups, remove old information from the coupling facility. If you do not have old information in your coupling facility, continue with the step 2.
   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, enter the following command to force off the connection of those structures:

      ```
      SETXCF FORCE,CONNECTION,STRNAME=strname,CONNAME=ALL
      ```

      Connections for the SCA are not held at termination; therefore you do not need to force off any SCA connections.
   c. Delete all the DB2 coupling facility structures that have a STATUS of ALLOCATED 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. Some examples of libraries that you might need to restore include:
   - DB2 SMP/E libraries
   - User program libraries
   - User DBRM libraries
   - DB2 CLIST libraries
   - DB2 libraries that contain customized installation jobs
   - JCL for creating user-defined table spaces
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.)
6. Obtain a copy of the installation job DSNTIJIN, which creates DB2 VSAM and non-VSAM data sets, for the first data sharing member. 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 on the first data sharing member. However, do not run DSNTIJID.

   For subsequent members of the data sharing group, run the DSNTIJIN that defines the BSDS and logs.
7. Recover the BSDS by following these steps for each member in the data sharing group:

a. Use the access method services **REPRO** command to restore the contents of one BSDS data set (allocated in step 6 on page 560) on each member. You can find the most recent BSDS image in the last file (archive log with the highest number) on the latest archive log tape.

b. Determine the RBA and LRSN ranges for this archive log by using 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. Use the STARTLRSN and ENDLRSN of this active log data set as the LRSN range (STARTLRSN and ENDLRSN) for this archive log.

c. Delete the oldest archive log from the BSDS.

d. Register this latest archive log tape data set in the archive log inventory of the BSDS that you just restored by using the change log inventory utility (DSNJU003). This step is necessary because the BSDS image on an archive log tape does not reflect the archive log data set that resides on that tape.

   Running DSNJU003 is critical for data sharing groups. Include the group buffer pool checkpoint information that is stored in the BSDS 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 BSDSs. You receive 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 as shown below

   The following sample DSNJU004 output shows the (partial) 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  ........  ..........
```

   The following sample DSNJU004 output shows the (partial) 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  ........  ..........
```

e. Adjust the active logs in the BSDS by using the change log inventory utility (DSNJU003), as necessary:

1) To delete all active logs in the BSDS, use the DELETE option of DSNJU003. Use the BSDS listing that is produced in step 7d on page 561 to determine the active log data set names.
2) To add the active log data sets to the BSDS, use the NEWLOG statement of DSNJU003. Do not specify a STARTRBA or ENDRBA in the NEWLOG statement. This specification indicates to DB2 that the new active logs are empty.

f. If you are using the DB2 distributed data facility, update the LOCATION and the LUNAME values in the BSDS by running the change log inventory utility with the DDF statement.

g. List the new BSDS contents by using the print log map utility (DSNJU004). 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 are correct).

h. If you are using dual BSDSs, make a copy of the newly restored BSDS data set to the second BSDS data set.

8. Optional: Restore archive logs to disk for each member. Archive logs are typically stored on tape, but restoring them to disk might 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 by using the integrated catalog and not the unit or VOLSER that is specified in the BSDS. If you are using dual BSDSs, remember to update both copies.
9. Use the DSN1LOGP utility to determine, for each member of the data sharing group, 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 generates a report.
10. Examine the DSN1LOGP output for each data sharing member, and identify any utilities that were executing at the end of the last archive log. Determine the appropriate recovery action to take on each table space that is involved in a utility job. If DSN1LOGP output showed that utilities are inflight (PLAN=DSNUTIL), examine SYSUTILX to identify the utility status and determine the appropriate recovery approach.
11. Modify DSNZP*xxx* parameters for each member of the data sharing group:

a. Run the DSNTINST CLIST in UPDATE mode.

b. To defer processing of all databases, select DATABASES TO START AUTOMATICALLY from panel DSNTIPB. Panel DSNTIPS opens. On panel DSNTIPS, type `DEFER` in the first field and `ALL` in the second field; then press **Enter**. You are returned to panel DSNTIPB.

c. To specify where you are recovering, select OPERATOR FUNCTIONS from panel DSNTIPB. Panel DSNTIPO opens. From panel DSNTIPO, type `RECOVERYSITE` in the SITE TYPE field. Press **Enter** to continue.

d. Optional: Specify which archive log to use by selecting OPERATOR FUNCTIONS from panel DSNTIPB. Panel DSNTIPO opens. From 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* by using job DSNTIJUZ (produced by the CLIST started in the first step of this procedure).

   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 that are in process.

12. Create a conditional restart control record for each data sharing member by using the change log inventory utility with one of the following forms of the CRESTART statement:

    - `CRESTART CREATE,ENDLRSN=`*nnnnnnnnnnnn*

      The *nnnnnnnnnnnn* is the LRSN of the last log record that is to be used during restart.
    - `CRESTART CREATE,ENDTIME=`*nnnnnnnnnnnn*

      The *nnnnnnnnnnnn* is the end time of the log record. Log records with a timestamp later than *nnnnnnnnnnnn* are truncated.

    Use the same LRSN or system time-of-day clock timestamp value for all members in a data sharing group. Determine the ENDLRSN value by using one of the following methods:

    - Use the DSN1LOGP summary utility. In the "Summary of Completed Events" section, find the lowest LRSN value that is 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 that is 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 the sample output that is shown in step 7d on page 561, the value 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 the archive logs of all members. Subtract 1 from the lowest LRSN in the data sharing group. Use this value for the ENDLRSN in the CRESTART statement. (In the sample output that is shown in step 7d on page 561, the value 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.

13. Enter the command **`START DB2 ACCESS(MAINT)`**.

    You must enter this command, because real-time statistics are active and enabled; otherwise, errors or abends could occur during DB2 restart processing and recovery processing (for example, GRECP recovery, LPL recovery, or the RECOVER utility).

If a discrepancy exists among the print log map reports as to the number of members in the group, which would be an unlikely occurrence, record the one that shows the highest number of members. 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 except one of the DB2 subsystems 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 mode members.

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 canceled 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 is issued for activity against the object that occurs during restart as a result of a unit of recovery or pending writes. In this case, the status that is 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 a state that DB2 does not expect.

00C900A9
: An attempt was made to allocate a deferred resource.

14. Resolve the indoubt units of recovery. The RECOVER utility, which you run in a subsequent step, fails 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. From an installation SYSADM authorization ID, enter the **RECOVER INDOUBT** command for all affected transactions.
15. Recover the catalog and directory. 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. Determine whether a utility was running at the time the latest archive log was created by entering the **DISPLAY UTILITY(*)** command, and record the name and current phase of any utility that is running. (You cannot restart a utility at the recovery site that was interrupted at the disaster site. To terminate a utility at the recovery site that was interrupted at the disaster site, you must use the TERM UTILITY command.)

d. Run the DIAGNOSE utility with the DISPLAY SYSUTIL option. The output consists of information about each active utility, including the table space name (in most cases). This is the only way to correlate the object name with the utility. Message DSNU866I gives information about the utility, and DSNU867I gives the database and table space name in USUDBNAM and USUSPNAM, respectively.

e. Use the **TERM UTILITY** command to terminate any utilities that are in progress on catalog or directory table spaces.

f. Recover the rest of the catalog and directory objects, starting with DBD01, in the order shown in the description of the RECOVER utility.

16. 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.
17. Use any method that you want to verify the integrity of the DB2 catalog and directory. Use the catalog queries in member DSNTESQ of data set DSN1010.SDSNSAMP after the work file database is defined and initialized.
18. If you use data definition control support, recover the objects in the data definition control support database.
19. If you use the resource limit facility, recover the objects in the resource limit control facility database.
20. Modify DSNZP*xxx* to restart all databases on each member of the data sharing group:
    a. Run the DSNTINST CLIST in UPDATE mode.
    b. From panel DSNTIPB, select DATABASES TO START AUTOMATICALLY. Panel DSNTIPS opens. Type RESTART in the first field and ALL in the second field, and press **Enter**. You are returned to DSNTIPB.
    c. Reassemble DSNZP*xxx* by using job DSNTIJUZ (produced by the CLIST started in step 4 on page 560).
21. Stop DB2.
22. Start DB2.
23. Make a full image copy of the catalog and directory.
24. Recover user table spaces and index spaces. If utilities were running on any table spaces or index spaces, see "What to do about utilities that were in progress at time of failure" on page 566. You cannot restart a utility at the recovery site that was interrupted at the disaster site. Use the **TERM UTILITY** command to terminate any utilities that are 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.DSNDB*x*.*dbname*.*psname*.*y*0001.*znnn*

The *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.

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 the SQL statement 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-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 queries for which the results are known.

25. Make full image copies of all table spaces and indexes with the COPY YES attribute.

26. Finally, compensate for work that was lost since the last archive was created by rerunning online transactions and batch jobs.

What to do next

Determine what to do about any utilities that were in progress at the time of failure.

Related concepts

Preparations for disaster recovery

What to do about utilities that were in progress at time of failure

Recovering data (DB2 Data Sharing Planning and Administration)

Related tasks

Migration step 1: Actions to complete before migration (DB2 Installation and Migration)

Invoking the CLIST (DB2 Installation and Migration)

Recovering catalog and directory objects (DB2 Utilities)

Related reference

DSN1LOGP (DB2 Utilities)

What to do about utilities that were in progress at time of failure

After you restore data from image copies and archives, you might need to take some additional steps. For example, you need to determine what to do about any utilities that were in progress at the time of the failure.

You might need to take additional steps if any utility jobs were running after the last time that the log was offloaded before the disaster.

After restarting DB2, only certain utilities need to be terminated with the **TERM UTILITY** command.

Allowing the RECOVER utility to reset pending states is preferable. However, you might occasionally need 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 controls."

For the following utility jobs, perform the indicated actions:

CHECK DATA
Terminate the utility, and run it again after recovery is complete.

COPY After you enter the **TERM UTILITY** command, DB2 places a record in the SYSCOPY catalog table to indicate that the COPY utility job 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 that you specified in the following table, and perform the specified actions. For the SORTKEYS option, you must specify a value that is greater than zero for *integer*. If you specify zero for *integer*, the SORTKEYS option does not apply.

Table 38. Actions to take when a LOAD utility job 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 that is 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, recover the table space to a prior point in time. Recover the indexes.
LOG NO, *copy-spec*, and SORTKEYS *integer*	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 the TOCOPY option of the RECOVER utility to do this.

To avoid extra loss of data in a future disaster situation, run the QUIESCE utility on table spaces before invoking the LOAD utility. This enables you to recover a table space by using the TOLOGPOINT option instead of TOCOPY.

REORG
For a user table space, find the options that you specified in the following table, and perform the specified actions.

Recommendation: Make full image copies of the catalog and directory before you run REORG on them.

Table 39. Actions to take when the REORG utility 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 the REORG job 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 the REORG job began. Recover the indexes.
SHRLEVEL CHANGE or SHRLEVEL REFERENCE	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.

For a catalog or directory table space, the instructions are somewhat different. For those table spaces that were using online REORG, find the options that you specified in the preceding table, and perform the specified actions.

If you have no image copies from immediately before REORG failed, use this procedure:

1. From your **DISPLAY UTILITY** command and DIAGNOSE utility output, determine what phase the REORG job was in and which table space it was reorganizing when the disaster occurred.
2. Run the RECOVER utility on the catalog and directory in the correct order. Recover all table spaces to the current time, except the table space that was being reorganized at the time of the disaster. If the RELOAD phase of the REORG job 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, a recovery to the current time restores the data to the state that it was in before the REORG job. If the RELOAD phase completed, perform the following actions:
   a. Run the DSN1LOGP utility against the archive log data sets from the disaster site.
   b. Find the begin-UR log record for the REORG job that failed in the DSN1LOGP output.
   c. Run the RECOVER utility 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 the REORG job failed, run the RECOVER utility with the TOCOPY option to recover the catalog and directory, in the correct order.

Related tasks

Recovering catalog and directory objects (DB2 Utilities)

Recovering from disasters by using a tracker site

You can use a tracker site for disaster recovery. A DB2 *tracker site* is a separate DB2 subsystem or data sharing group that exists solely to keep shadow copies of the data at your primary site.

About this task

Using a tracker site for disaster recovery is somewhat similar to other methods.

Recommendation: Test and document a disaster procedure that is customized for your location.

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 need to constantly ship image copies; the takeover time for the tracker site might be faster because DB2 recovery does not need to use image copies.

Characteristics of a tracker site

A *tracker site* is a separate DB2 subsystem or data sharing group that exists solely for the purpose of keeping shadow copies of the data at your primary site.

Because the tracker site must use only the primary site 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.

- 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 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. Use of the **START DATABASE** command is not necessary to clear LPL or GRECP conditions because you are going to be running RECOVER 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

For disaster recovery purposes, you might want to set up a tracker site. To set up a tracker site, you create a mirror image of your primary DB2 subsystem, and then ensure that the tracker site is synchronized with the primary site.

Procedure

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 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 that this site is to receive 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 plan to send over from the primary site.
4. Optional: Allocate data sets for user-managed indexes that you want to rebuild during recovery cycles. The main reason that you rebuild indexes during recovery cycles is for running efficient queries on the tracker site. If you do not require indexes, you do not need 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 that it takes to bring up the disaster site. Be sure that you define data sets with the proper prefix (either I or J) for both indexes and table spaces.
5. Send full image copies of all DB2 data at the primary site 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 the prior, tailor installation job DSNTIJIN to create DB2 catalog data sets.

What to do next

Important: Do not attempt to start the tracker site when you are setting it up. You must follow the procedure described in "Establishing a recovery cycle by using RESTORE SYSTEM LOGONLY."

Related reference

BACKUP SYSTEM (DB2 Utilities)

Establishing a recovery cycle by using RESTORE SYSTEM LOGONLY

Each time that you restore the logs and the BSDS from the primary site at your tracker site, you establish a new recovery cycle. One way to establish a recovery cycle is to use the RESTORE SYSTEM utility with the LOGONLY option.

Before you begin

Full image copies of all the data at the primary site must be available at the tracker site.

About this task

Using the LOGONLY option of the RESTORE SYSTEM utility enables you to periodically apply the active log, archive logs, and the BSDS from the primary site at the tracker site.

Procedure

To establish a recovery cycle at your tracker site by using the RESTORE SYSTEM utility:

1. While your primary site continues its usual workload, send a copy of the primary site 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 by following these steps:
   a. Locate the BSDS in the latest archive log that is now at the tracker site.
   b. Register this archive log in the archive log inventory of the new BSDS by using the change log inventory utility (DSNJU003).
   c. Register the primary site active log in the new BSDS by using the change log inventory utility (DSNJU003).
3. Use the change log inventory utility (DSNJU003) with the following CRESTART control statement:

   ```
   CRESTART CREATE,ENDRBA=nnnnnnnn000, FORWARD=NO,BACKOUT=NO
   ```

   In this control statement, *nnnnnnnn* equals the RBA at which the latest archive log record ends +1. Do not specify the RBA at which the 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,ENDLRSN=nnnnnnnnnnnn,FORWARD=NO,BACKOUT=NO
   ```

   In this control statement, *nnnnnnnnnnnn* is the lowest LRSN of all the members that are to be read during restart. 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 (message DSN1213I), you can use the displayed value.

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 the DSN1COPY utility to create a copy of SYSUTILX during the last tracker cycle, restore this copy with DSN1COPY.

   Data sharing
   For data sharing, restart every member of the data sharing group.
6. At the tracker site, restart DB2 to begin a tracker site recovery cycle.
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.
8. If the RESTORE SYSTEM utility issues a return code of 4, use the DSN1COPY utility to make a copy of SYSUTILX and of indexes that are associated with SYSUTILX before you recover or rebuild those objects. DSN1COPY issues a return code of 4 if application of the log results in one or more DB2 objects being marked 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 to 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 are not retained in the next recovery cycle.
11. After all recovery activity complete at the tracker site, shut down the DB2 tracker site.
12. Optional: Stop and start the DB2 tracker site several times before completing a recovery cycle.

Related concepts

Media failures during LOGONLY recovery

Related tasks

Establishing a recovery cycle by using the RECOVER utility

Restoring data from image copies and archive logs

Establishing a recovery cycle by using the RECOVER utility

Each time that you restore the logs and the BSDS from the primary site at your tracker site, you establish a new recovery cycle. One way to establish a recovery cycle is to use the RECOVER utility.

Procedure

To establish a recovery cycle by using the RECOVER utility:

1. While your primary site continues its usual workload, send a copy of the primary site 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 copy of DSNDB01.SYSUTILX by using the DSN1COPY utility.

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, discard the tracker cycle updates 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 by using one of the following methods:
   - Locate the BSDS in the latest archive log that is now at the tracker site.
   - Register this archive log in the archive log inventory of the new BSDS by using the change log inventory utility (DSNJU003).
   - Register the primary site active log in the new BSDS by using the change log inventory utility (DSNJU003).
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 value of the ENDRBA of the latest archive log plus 1. Do 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 that are to be read during restart. 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 message DSNJ003I at the console logs use ENDLRSN -1 as the input to the conditional restart.
   - If you receive the ENDLRSN from the output of the DSN1LOGP utility (DSN1213I message), use the displayed value.

   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 utility 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.

   You must recover database objects as the following procedure specifies:

   a. Restore the full image copy or DSN1COPY of 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 you recover other system and user table spaces, determine what utilities were running at the primary site.

   b. Recover the catalog and directory in the correct order.

      If you have user-defined catalog indexes, rebuilding them is optional until the tracker DB2 site becomes the takeover DB2 site. (You might want to rebuild them sooner if you require them for catalog query performance.) However, if you are recovering user-defined catalog indexes, do the recovery in this step.

      Exception: If you have any user-defined, STOGROUP-managed indexes on the DB2 catalog and directory, you must rebuild IBM-defined indexes by name.

   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.

      You do not need to rebuild indexes unless you intend to run dynamic queries on the data at the tracker site.

   For 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. Issue the **DISPLAY UTILITY(*)** command for a list of currently running utilities.
8. Run the DIAGNOSE utility with the DISPLAY SYSUTIL statement to determine 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 because DB2 prevents you from using the **TERM UTILITY** command to remove pending statuses at a tracker site.
   - If a LOAD, REORG, REPAIR, or COPY utility job is in progress on any catalog or directory object at the primary site, shut down DB2 subsystem. You cannot continue recovering by using 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 job 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 a restart-pending state.

Data sharing

If read/write shared data (GPB-dependent data) is in the advisory recovery pending state, the tracker DB2 site performs recovery processing. Because the tracker DB2 site always performs a conditional restart, the postponed indoubt units of recovery are not recognized after the tracker DB2 site restarts.

10. After all recovery has completed at the tracker site, shut down the tracker DB2 site. This is the end of the tracker site recovery cycle.
11. Optional: Stop and start the tracker DB2 site several times before completing a recovery cycle.

Related concepts

Media failures during LOGONLY recovery

Related tasks

Establishing a recovery cycle by using RESTORE SYSTEM LOGONLY

Restoring data from image copies and archive logs

Media failures during LOGONLY recovery

If an I/O error occurs during a LOGONLY recovery, you can recover the object by 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. (This is possible, however, for a non-tracker system.) Instead, you must remove the corrupted volume and re-initialize another volume with the same volume serial number before you invoke the RECOVER utility for all table spaces and indexes on that volume.

Maintaining a tracker site

If you want to have a tracker site for possible disaster recovery needs, you need to maintain it so that it can operate as required.

Procedure

To maintain a tracker site:

1. Keep the tracker site and primary site be at the same maintenance level to avoid unexpected problems.
2. Between recovery cycles, apply maintenance as you normally do, by stopping and restarting the DB2 subsystem or a DB2 data sharing member.
3. If a tracker site fails, restart it as you normally do.
4. Save your complete tracker site prior to testing a takeover site. This step is necessary because bringing up a tracker site as the takeover site destroys the tracker site environment. After testing the takeover site, you can restore the tracker site and resume the recovery cycles.

Results

When restarting a data sharing group, the first member that starts during a recovery cycle 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 be the takeover site

If a disaster occurs at the primary site, the tracker site must become the takeover site.

Before you begin

Save your complete tracker site prior to testing a takeover site.

Procedure

To make the tracker site be the takeover site:

1. Restart the takeover site.
2. Apply log data or image copies that were en route when the disaster occurred.
3. Follow the appropriate procedure for making the tracker site a takeover site, depending on whether you use RESTORE SYSTEM LOGONLY or the RECOVER utility in your tracker site recovery cycles.

   Related tasks

   Maintaining a tracker site

Recovering at a tracker site that uses the RESTORE SYSTEM utility:

One way that you can make the tracker site be the takeover site is by using the RESTORE SYSTEM utility with the LOGONLY option in the recovery cycles at the tracker site.

Procedure

To make the tracker site be the takeover site by using the RESTORE SYSTEM utility with the LOGONLY option:

1. If log data for a recovery cycle is en route or is available but has not yet been used in a recovery cycle, perform the procedure in "Establishing a recovery cycle by using RESTORE SYSTEM LOGONLY" on page 570.
2. Ensure that the TRKSITE NO subsystem parameter is specified.
3. For scenarios other than data sharing, continue with step 4.

   Data sharing

   If this is a data sharing system, delete the coupling facility structures.
4. Start DB2 at the same RBA or ENDLRSN 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 by issuing the `START DATABASE(*) SPACENAM(*)` command.
6. If you used the DSN1COPY utility 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 **DISPLAY UTILITY(*)** command .
   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 in the correct order by using the **TERM UTILITY(*)** command.
8. Rebuild indexes, including IBM and user-defined indexes on the DB2 catalog and user-defined indexes on table spaces.

Related tasks

Restoring data from image copies and archive logs

Recovering at a tracker site that uses the RECOVER utility

Recovering at a tracker site that uses the RECOVER utility:

One way that you can make the tracker site be the takeover site is by using the RECOVER utility in the recovery cycles at your tracker site.

Procedure

To make the tracker site be the takeover site by using the RECOVER utility:

1. Restore the BSDS, and register the archive log from the last archive log that you received from the primary site.
2. For environments that do not use data sharing, continue with step 3.

   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. Take the appropriate action, which depends on whether you received more logs from the primary site. If this is a non-data-sharing DB2 subsystem, the log truncation point varies depending on whether you have received more logs from the primary site since the last recovery cycle:
   - If you did not receive 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, they are current except for any objects that had outstanding units of recovery during restart. Because the previous cycle specified NO for both FORWARD and BACKOUT and you have now specified YES, affected data sets are placed in the LPL. Restart the objects that are in LPL status by using the following 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 equals the value of the ENDRBA of the latest archive log plus 1. 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; use the following CRESTART statement:

   ```
   CRESTART CREATE,ENDLRSN=nnnnnnnnnnnn,FORWARD=YES,BACKOUT=YES
   ```

   In this statement, *nnnnnnnnnnnn* is the LRSN of the last log record that is to be used during restart. 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 message DSNJ003I at the console logs use ENDLRSN -1 as the input to the conditional restart.
- If you receive the ENDLRSN from the output of the DSN1LOGP utility (DSN1213I message), use the displayed value.

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.

The takeover DB2 sites 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 copy that was taken at the tracker site.
6. Terminate any in-progress utilities by using the following procedure:
   a. Enter the command **DISPLAY UTILITY(*)**.
   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 by using the command **TERM UTILITY(*)**.
7. Continue with your recoveries either with the LOGONLY option or with image copies. Remember to rebuild indexes, including IBM and user-defined indexes on the DB2 catalog and user-defined indexes on table spaces.

Related tasks

Restoring data from image copies and archive logs

Recovering at a tracker site that uses the RESTORE SYSTEM utility

Using data mirroring for disaster recovery

Data mirroring is the automatic replication of current data from your primary site to a secondary site. To recover after a disaster, you can use this secondary site for your recovery site without the need to restore DB2 image copies. You also do not need to apply DB2 logs to bring DB2 data to the current point in time.

About this task

The 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 by using consistency groups.

Follow the appropriate procedure for recovering from a disaster by using data mirroring.

Role of data mirroring in recovery from a rolling disaster

In a real disaster, your local site gradually and intermittently fails for a duration of several seconds. This kind of DB2 failure is known as a *rolling disaster*. You can recover from a rolling disaster by using data mirroring.

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 a DB2 subsystem and data 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.

The following figure shows how a rolling disaster can cause data to become inconsistent between two subsystems.

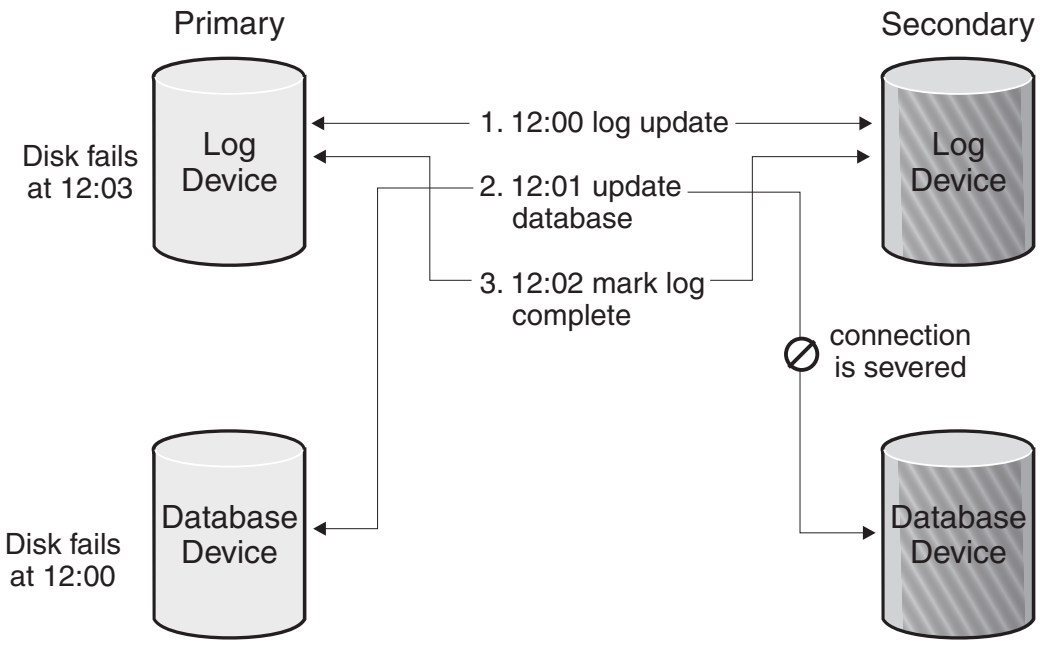


Figure 51. Data inconsistency caused by a rolling disaster

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 the preceding figure depicts.

1. Some time prior to 12:00: A table space is updated in the buffer pool.
2. 12:00 The log record is written to disk on logical storage subsystem 1.
3. 12:01: Logical storage subsystem 2 fails.
4. 12:02: The update to the table space is externalized to logical storage subsystem 2 but is not written because subsystem 2 failed.
5. 12:03: The log record that indicates that the table space update was made is written to disk on logical storage subsystem 1.
6. 12:03: Logical storage subsystem 1 fails.

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.

Important: 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.

Role of consistency groups in recovery

Generally a *consistency group* is a collection of volumes that contain consistent, related data. Consistency groups play an important role in DB2 recovery.

A consistency group, which is a collection of related data, can span logical storage subsystems and disk subsystems. For DB2 specifically, a consistency group contains an entire DB2 subsystem or an entire 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 the following methods to create consistency groups:

- XRC I/O timestamping and system data mover
- FlashCopy consistency groups
- GDPSfreeze policies
- The DB2 SET LOG SUSPEND command

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 by taking 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 define. 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 recover data at your secondary site from a disaster at your primary site.

About this task

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, UNLOAD, or RECOVER utility was active at the time of the disaster.

Procedure

To recover at the secondary site after a disaster:

1. At your recovery site, IPL all z/OS images that correspond to the z/OS images that you lost at your primary site.
2. For environments that do not use data sharing, continue with step 3.

   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 LOCATION and LUNAME in the BSDS to values that are specific to your new primary site. To set LOCATION and LUNAME, run the stand-alone change log inventory utility (DSNJU003) with the following control statement:
   `DDF LOCATION=locname, LUNAME=luname`
4. Start all DB2 members by 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 step 6 on page 582.

   Data sharing
   For objects that are in GRECP status, DB2 automatically recovers the

objects during restart. Message DSNI049I is issued when the recovery for all objects that are in GRECP status is complete. A message is issued for each member, even if the member did not perform GRECP recovery.

After message DSNI049I is issued:

a. Display all data sets with GRECP or LPL status by issuing the following DB2 command:

```
-DISPLAY DATABASE(*) SPACENAM(*) RESTRICT(GRECP, LPL) LIMIT(*)
```

Record the output that this command generates.

6. Use the following DB2 command to display all utilities that the failure interrupted:

```
-DISPLAY UTILITY(*)
```

If utilities are pending, record the output from this command, and continue to the next step. You cannot restart utilities at a recovery site. You will terminate these utilities in step 8. If no utilities are pending, continue to step number 9.

7. Use the DIAGNOSE utility to access the SYSUTIL directory table. You cannot access this directory table by using normal SQL statements (as you can with most other directory tables). You can access SYSUTIL only by using the DIAGNOSE utility, which is normally intended to be used under the direction of IBM Software Support.

Use the following control statement to run the DIAGNOSE utility job:

```
DIAGNOSE DISPLAY SYSUTIL
```

To stop the utility, issue this control statement:

```
END DIAGNOSE
```

Examine the output. 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 environments that do not use data sharing, continue to step 10.

Data sharing

For data sharing groups, use the following **START DATABASE** command on each database that contains objects that are in 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 places 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 NONE 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 NONE and the LOAD job was interrupted during or after the RELOAD phase, recover this object to a point in time that is before this utility ran.
- Otherwise, recover the object to a point in time that is before the LOAD job ran.

12. For each object that the REORG utility places 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, recover this object to a point in time that is before this utility ran.
      - 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 SHRLEVEL 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.

Managing DFSMShsm default settings when using the BACKUP SYSTEM, RESTORE SYSTEM, and RECOVER utilities

In some data mirroring situations, you might need to set or override the DFSMShsm default settings for the BACKUP SYSTEM, RESTORE SYSTEM, and RECOVER utilities.

Before you begin

The following prerequisites apply:

- You must be running on z/OS Version 1 Release 8 or later.
- You must apply APAR OA23849 to enable the DFSMShsm FRBACKUP and FRRECOV functions to support PPRC primary volumes.
- You must apply APAR OA24814 so that DFSMShsm supports IBM Remote Pair FlashCopy in FRBACKUP and FRRECOV operations in an SMS copy pool environment.

About this task

For example, consider that the source volumes in the SMS storage groups for your database or log copy pools are mirrored, or that the target volumes in the SMS backup storage groups for your copy pools are mirrored. You can use IBM Remote Pair FlashCopy (Preserve Mirror) for Peer-to-Peer Remote Copy (PPRC). Also, you

can allow FlashCopy to PPRC primary volumes. However, you might need to set or override the DFSMShsm default settings for the BACKUP SYSTEM, RESTORE SYSTEM, and RECOVER utilities.

Procedure

To manage the DFSMShsm default settings for these utilities:

Issue the DFSMShsm FRBACKUP PREPARE command.

- To set the DFSMShsm defaults for the BACKUP SYSTEM utility, the RESTORE SYSTEM utility, and the RECOVER utility, issue the following command:
  ```
  FRBACKUP CP cp-name PREPARE ALLOWPPRCP (FRBACKUP (x) FRRECOV (x))
  ```
- To override the DFSMShsm defaults for the RESTORE SYSTEM utility or the RECOVER utility, issue the following command:
  ```
  FRBACKUP CP cp-name PREPARE ALLOWPPRCP (FRRECOV (x))
  ```

Related concepts

Considerations for using the BACKUP SYSTEM utility and DFSMShsm

Related reference

FRBACKUP command: Requesting a fast replication backup or dump version DFSMShsm Storage Administration Reference

Syntax and options of the RECOVER control statement (DB2 Utilities)

Syntax and options of the RESTORE SYSTEM control statement (DB2 Utilities)

Recovering with Extended Remote Copy

One method that ensures that data volumes remain consistent at your recovery site involves 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.

Before you begin

This procedure assumes that you are familiar with basic use of XRC.

Procedure

To recover at an XRC secondary site after a disaster:

1. Issue the TSO command **XEND XRC** to end the XRC session.
2. Issue the TSO command **XRECOVER XRC**. 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 581.

Related information

Extended Remote Copy (DFSMS Advanced Copy Services)

Scenarios for resolving problems with indoubt threads

Indoubt threads can cause a variety of problems, but you can recover from these problems.

The recovery scenarios for indoubt threads are based on a sample environment, which this topic describes. 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.

Configuration

The configuration includes 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 run at Seattle and access both local and remote data.

- IMS application, IMSAPP01, at Seattle, accesses local data and remote data by DRDA access at San Jose, which accesses remote data on behalf of Seattle by DB2 private protocol access at Los Angeles.
- TSO application, TSOAPP01, at Seattle, accesses data by DRDA access at San Jose and at Los Angeles.

Threads

The following threads are described and keyed to Figure 52 on page 586. Database access threads (DBAT) access data on behalf of a thread (either allied or DBAT) at a remote requester.

- Allied IMS thread **A** at Seattle accesses data at San Jose by DRDA access.
  - DBAT at San Jose accesses data for Seattle by DRDA access **1** and requests data at Los Angeles by DB2 private protocol access **2**.
  - DBAT at Los Angeles accesses data for San Jose by DB2 private protocol access **2**.
- Allied TSO thread **B** at Seattle accesses local data and remote data at San Jose and Los Angeles, by DRDA access.
  - DBAT at San Jose accesses data for Seattle by DRDA access **3**.
  - DBAT at Los Angeles accesses data for Seattle by DRDA access **4**.

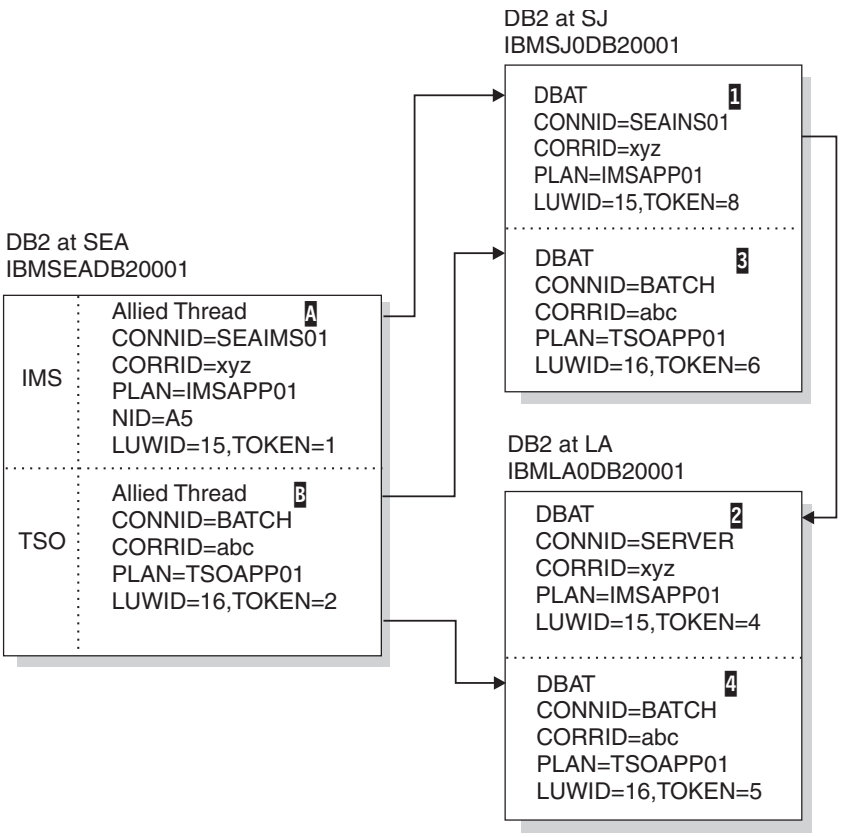


Figure 52. Resolution of indoubt threads. Results of issuing **DISPLAY THREAD TYPE(ACTIVE)** at each DB2 subsystem.

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 the preceding figure. The logical unit of work IDs (LUWIDs) have been shortened for readability, as follows:

- LUWID=15 is IBM.SEADB21.15A86A876789.0010.
- LUWID=16 is IBM.SEADB21.16B57B954427.0003.

For the purposes of procedures that are based on this configuration, 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.

Read one or more of the scenarios to learn how best to handle problems with indoubt threads in your own environment.

Scenario: Recovering from communication failure

A communication failure can cause an indoubt thread.

Symptoms

A communication failure occurred between Seattle (SEA) and Los Angeles (LA) after the database access thread (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**.

At SEA, NetView alert A006 is generated, and message DSNL406 is displayed, indicating that an indoubt thread at LA because of a communication failure. At LA, alert A006 is generated, and message DSNL405 is displayed, to indicate that a thread is in an indoubt state because of a communication failure with SEA.

Causes

A communication failure caused the indoubt thread.

Environment

The following figure illustrates the environment for this scenario.

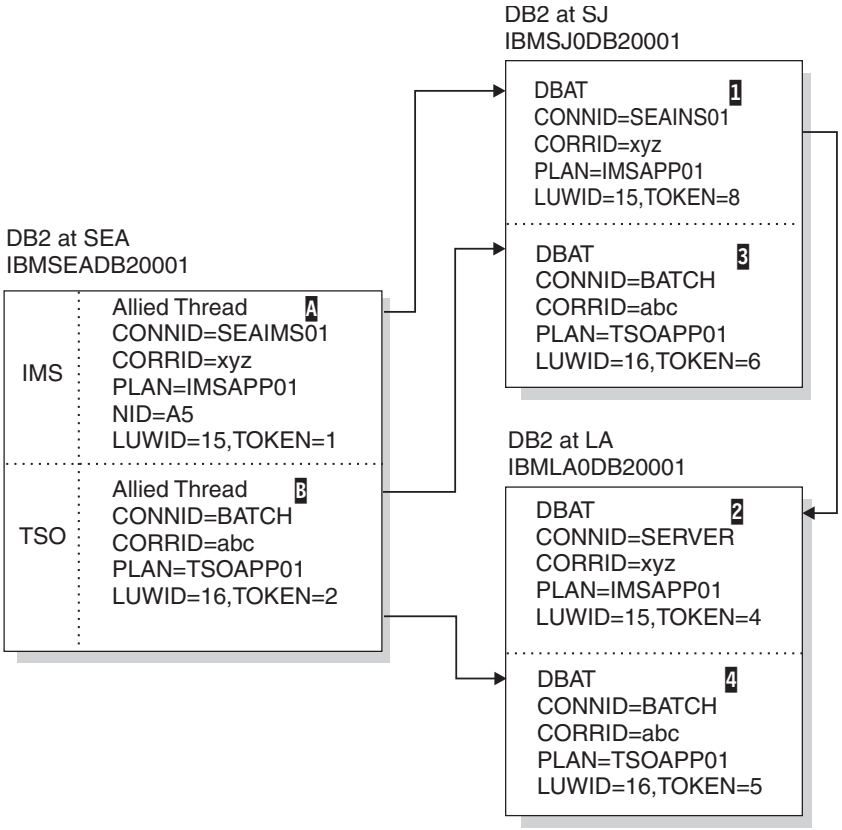


Figure 53. Resolution of indoubt threads. "Scenarios for resolving problems with indoubt threads" on page 584 contains a detailed description of the scenario depicted in this figure.

At SEA, an IFCID 209 trace record is written. After the alert is generated and the message is 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 subsystem has an indoubt resolution responsibility. The thread shows up in a **DISPLAY THREAD** report for indoubt threads. The thread also shows up in a **DISPLAY THREAD** report for active threads until the application terminates.

The TSO application is informed that the commit succeeded. If the application continues and processes another SQL request, it is rejected with an SQL code to indicate that it must roll back before any more SQL requests can be processed. This is to ensure that the application does not proceed with an assumption based on data that is 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 in the indoubt state. All locks remain held until resolution occurs. The thread shows up in a **DISPLAY THREAD** report for indoubt threads.

The DB2 subsystems, at both SEA and LA, periodically attempt to reconnect and automatically resolve the indoubt thread. If the communication failure affects only the session that is 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.

Resolving the problem

Operator response: If message DSNL407 or DSNL415 for the thread that is identified in message DSNL405 is not issued in a reasonable period of time, contact the system programmer. A communication failure is making database resources unavailable.

System programmer response: Determine and correct the cause of the communication failure. When the problem is 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 that are held for the indoubt thread.

Scenario: Making a heuristic decision about whether to commit or abort an indoubt thread

An organization might need to make a heuristic decision about whether to commit or abort an indoubt thread.

Symptoms

In this scenario, an indoubt thread at Los Angeles (LA) holds database resources that are needed by other applications. The organization makes a heuristic decision about whether to commit or abort an indoubt thread.

Many symptoms are possible, including:

- Message DSNL405 to indicate 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 LOCKS** report that shows a large number of threads that are waiting for the locks that are held by the indoubt thread
- Some threads terminating due to timeout
- IMS and CICS transactions not completing

Environment

The following figure illustrates the environment for this scenario.

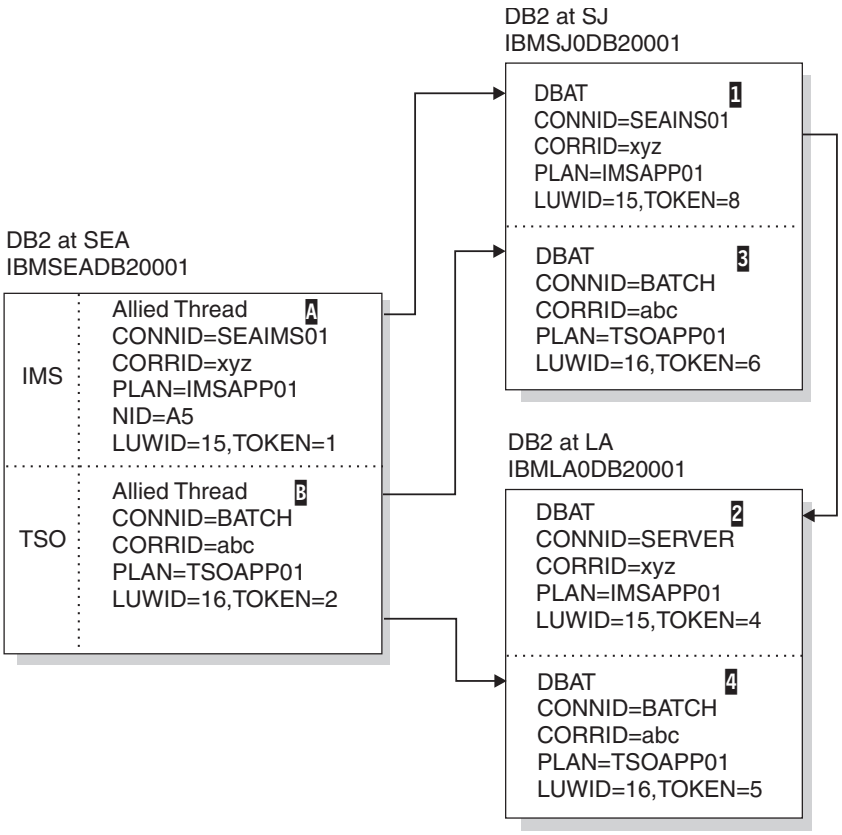


Figure 54. Resolution of indoubt threads. "Scenarios for resolving problems with indoubt threads" on page 584 contains a detailed description of the scenario depicted in this figure.

Resolving the problem

Database administrator response: Determine whether to commit or abort the indoubt thread. First, determine the name of the commit coordinator for the indoubt thread. This name matches the location name of the DB2 subsystem at SEA, and it is included in the DB2 indoubt thread **DISPLAY THREAD** report at LA. Then, have an authorized person at SEA perform one of the following actions:

- If the coordinator DB2 subsystem is active, or if it can be started, request a **DISPLAY THREAD** report for indoubt threads, specifying the LUWID of the thread. (Remember that the token that is used at LA is different than the token that is used at SEA). If no report entry exists for the LUWID, the proper action is to abort. If a report entry for the LUWID exists, 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 that contains the indoubt LUWID. This entry indicates whether the commit decision was commit or abort.
- If statistics class 4 is not available, run the DSN1LOGP utility, and request a summary report. The volume of log data that is to be searched can be restricted if you can determine the approximate SEA log RBA value that was 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 heuristic decision. An IFCID 203 trace record is written, recording the heuristic action.

Scenario: Recovering from an IMS outage that results in an IMS cold start

An IMS outage can result in an IMS cold start. An organization that experiences this situation can recover.

Symptoms

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.

Environment

The following figure illustrates the environment for this scenario.

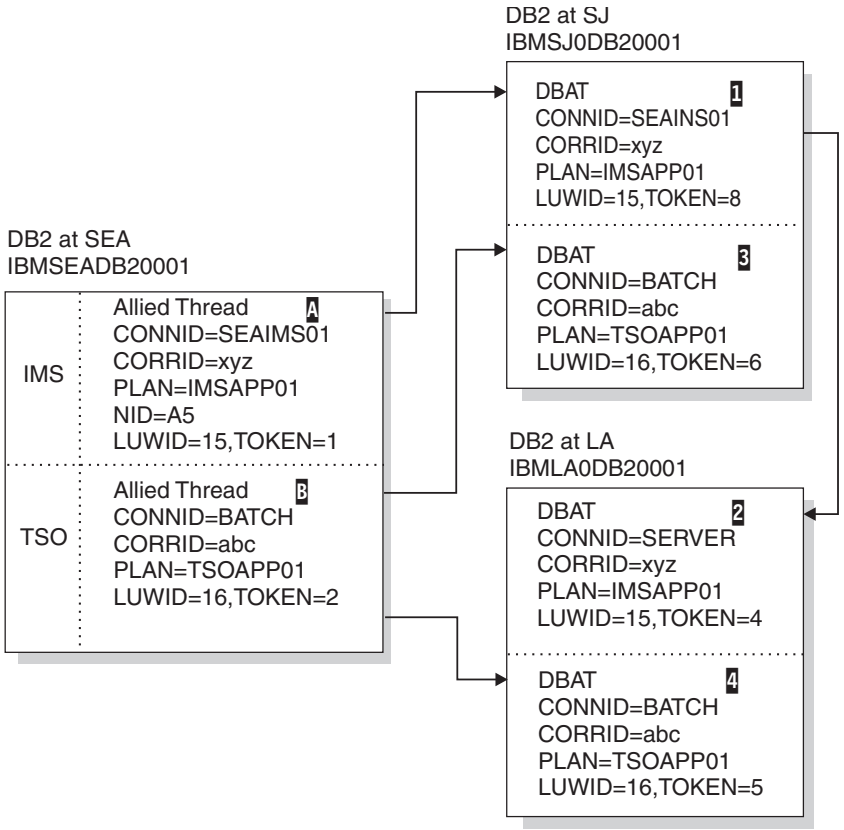


Figure 55. Resolution of indoubt threads. "Scenarios for resolving problems with indoubt threads" on page 584 contains a detailed description of the scenario depicted in this figure.

The abnormal termination of IMS has left one allied thread **A** at the SEA DB2 subsystem indoubt. This is the thread whose LUWID=15. Because the SEA DB2 subsystem still has effective communication with the DB2 subsystem at SJ, the LUWID=15 DBAT **1** at this subsystem 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.
- 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.

Resolving the problem

System programmer response: Issue the **RECOVER INDOUBT** command to resolve the indoubt thread at the SEA DB2 subsystem. This completes the two-phase commit process with the DB2 subsystems at SJ and LA, and the unit of work either commits or aborts.

1. Force the IMS log closed by using the **/DBR FEOV** command, and then archive the IMS log. Use the command **DFSERA10** to print the records from the previous IMS log tape for the last transaction that was processed in each dependent region. Record the PSB and the commit status from the X'37' log that contains the recovery ID.
2. Run the DL/I batch job to back out each PSB that is involved and that has not reached a commit point. The process might take some time because transactions are still being processed. The process might also lock up a number of records, which could affect 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) that is displayed in the **DISPLAY THREAD** messages with the OASNs (4 bytes decimal) as 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, for use during the next step.
6. Enter the following DB2 command, choosing to commit or roll back, and specify the correlation ID:

   ```
   -RECOVER INDOUBT (imsid) ACTION(COMMIT|ABORT) NID (nid)
   ```

   If the command is rejected because network IDs are associated, use the same command again, substituting the recovery ID for the network ID.

   Related concepts

   Duplicate IMS correlation IDs

Scenario: Recovering from a DB2 outage at a requester that results in a DB2 cold start

When an outage at a DB2 requester results in a cold start, the organization that has this situation can recover.

Symptoms

The DB2 subsystem at SEA is started with a conditional restart record in the BSDS to indicate a cold start:

- When the IMS subsystem reconnects, it attempts to resolve the indoubt thread that is identified in IMS as NID=A5. IMS has a resource recovery element (RRE)

for this thread. The SEA DB2 subsystem informs IMS that it has no knowledge of this thread. IMS does not delete the RRE, and the RRE can be displayed by using the IMS **DISPLAY OASN** command. The SEA DB2 subsystem 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 subsystem has cold started. Both the SJ DB2 and the LA DB2 subsystem:
  - 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.

Causes

An abnormal termination of the SEA DB2 subsystem caused the outage.

Environment

The following figure illustrates the environment for this scenario.

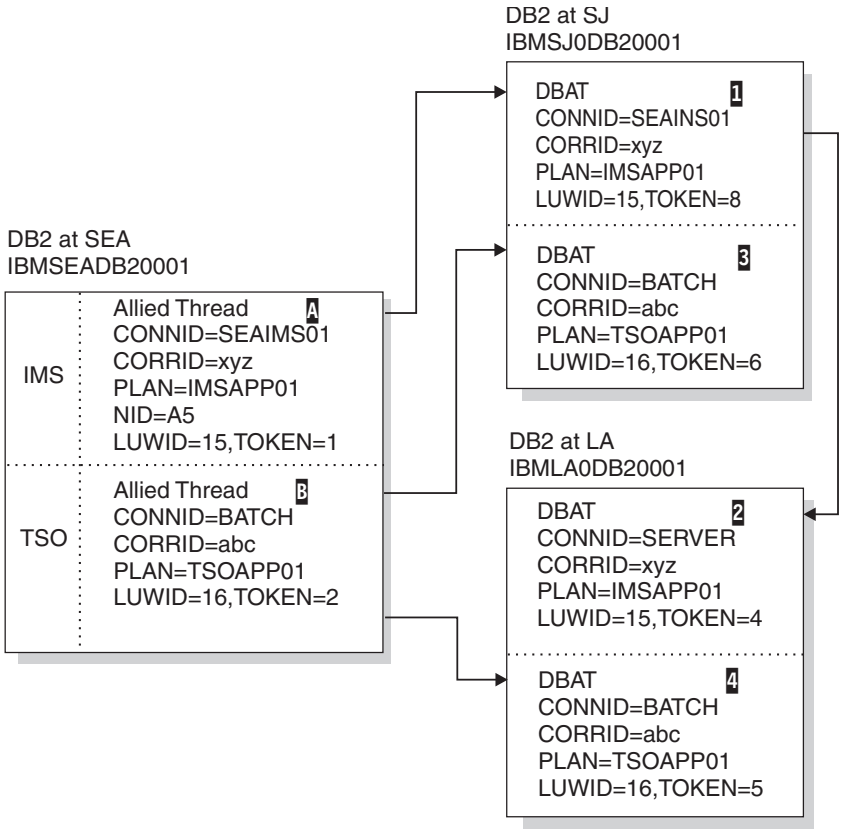


Figure 56. Resolution of indoubt threads. "Scenarios for resolving problems with indoubt threads" on page 584 contains a detailed description of the scenario depicted in this figure.

The abnormal termination of the SEA DB2 subsystem 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 subsystem to communicate the final decision.

The IMS subsystem at SEA is operational and has the responsibility of resolving indoubt units with the SEA DB2 subsystem.

The DB2 subsystems at both SJ and LA accept the cold start connection from SEA. Processing continues, waiting for the heuristic decision to resolve the indoubt threads.

Resolving the problem

Database administrator response: 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, the SEA DB2 subsystem was a participant for LUWID=15, which was being coordinated by IMS.
- The administrator at LA also does not know 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. Both SJ and LA need to make the same heuristic decision. The administrators at SJ and LA also need to 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 subsystem cold started, you might be able to determine the decision from its recovery log. Alternatively, if the failure occurred before the decision was recorded, you might be able to determine the name of the coordinator, if the SEA DB2 subsystem was a participant. You can obtain a summary report of the SEA DB2 recovery log by running 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 the site that is identified by the LUWID.
- The administrator must determine if the LU name that is 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), the SEA DB2 subsystem is a participant in the logical unit of work, and it is being coordinated by a remote system. The DBA 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, the logical unit of work originated at SEA and can be an IMS, CICS, TSO, or batch allied thread of the SEA DB2 subsystem. The **DISPLAY THREAD** report for indoubt threads at a DB2 participant includes message DSNV458 if the coordinator is remote. This line provides external information that is 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 by using this connection name.
- In this example, the administrator at SJ sees that both indoubt threads have an LUWID with the LU name that match the SEA DB2 subsystem 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 the SEA DB2 subsystem 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 possible. Run the DSN1LOGP utility against the SEA DB2 recovery log, and look for the LUWID=16 entry. Three possibilities exist:
  1. No entry is found. That portion of the DB2 recovery log is 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 that is indicated in the SEA DB2 recovery log. In the other two cases, the recovery procedure that is 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 the administrator determines that updates were performed at SJ, LA, or both (but not SEA), and if both SJ and LA make the same heuristic action, data inconsistency probably exists. If updates were also performed at SEA, the administrator can look at the SEA data to 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), several alternative paths to recovery are available. The SEA DB2 subsystem 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, either commit or abort, is included in the message. Trace event IFCID=234 is also written to statistics class 4, which contains the same information.
- If only one such message exists, or if one such entry is in statistics class 4, the decision for indoubt thread LUWID=15 is known and can be communicated to the administrator at SJ. If multiple such messages exist, or if multiple such trace events exist, the administrator must match the IMS NID with the network LUWID. Again, the administrator should use **DSN1LOGP** to analyze the SEA DB2 recovery log if possible. Now four possibilities exist:
  1. No entry is found. That portion of the DB2 recovery log was not available.
  2. An entry is found but is incomplete because of lost recovery log data.
  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 that is indicated in the SEA DB2 recovery log. If an entry is found and its 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 enables correlation to the DSN3005 message, or to the 234 trace event, either of which provides the correct decision.

- If an incomplete entry is found, the NID might have been reported by DSN1LOGP. If it was reported, use it as previously discussed.
- Determine if any of the following conditions exists:
  - No NID is found.
  - The SEA DB2 subsystem has not been started.
  - Reconnecting to IMS has not occurred.

  If any of these conditions exists, the administrator must use the *correlation-id* that is used by IMS to correlate the IMS logical unit of work to the DB2 thread in a search of the IMS recovery log. The SEA DB2 site provided this value to the SJ DB2 subsystem when distributing the thread to SJ. The SJ DB2 site displays this value in the report that is generated by the **DISPLAY THREAD TYPE(INDOUBT)** command.
- 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
  ```

Related concepts

Scenario: What happens when the wrong DB2 subsystem is cold started

Scenario: What happens when the wrong DB2 subsystem is cold started

When one DB2 subsystem, instead of another DB2 subsystem, is cold started, threads are left indoubt. An organization that faces this situation can recover.

The following figure illustrates the environment for this scenario.

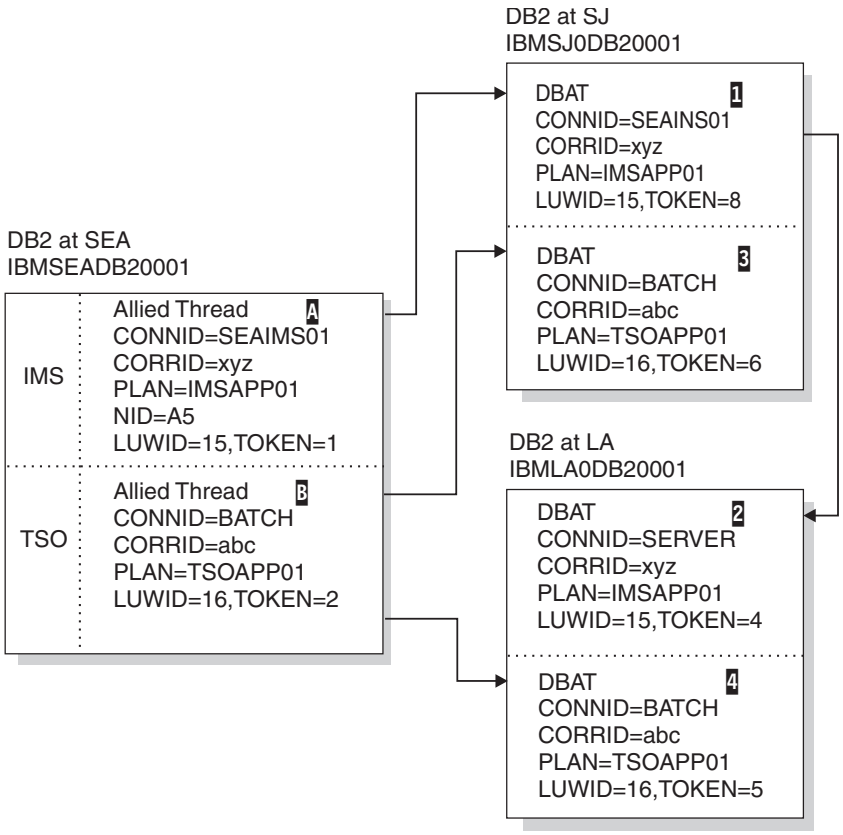


Figure 57. Resolution of indoubt threads. "Scenarios for resolving problems with indoubt threads" on page 584 contains a detailed description of the scenario depicted in this figure.

If the DB2 subsystem at SJ is cold started instead of the DB2 at SEA, the LA DB2 subsystem has the LUWID=15 **2** thread indoubt. The administrator can see that this thread did not originate at SJ, but that it did originate at SEA. To determine the commit or abort action, the LA administrator requests that **DISPLAY THREAD TYPE(INDOUBT)** be issued at the SEA DB2 subsystem, specifying LUWID=15. IMS does not have any indoubt status for this thread because it completes the two-phase commit process with the SEA DB2 subsystem.

The DB2 subsystem at SEA tells the application that the commit succeeded.

When a participant cold starts, a DB2 coordinator continues to include in the display of information about 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, the administrator can request a display of indoubt threads at the DB2 coordinator to determine the correct action.

Related information:

Scenario: Recovering from communication failure

Scenario: Recovering from a DB2 outage at a requester that results in a DB2 cold start

Scenario: Correcting damage from an incorrect heuristic decision about an indoubt thread

If an incorrect heuristic decision is made regarding an indoubt thread, an organization can recover from this incorrect decision.

Symptoms

When the DB2 subsystem 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.

Causes

This scenario is based on the conditions described in "Scenario: Recovering from communication failure" on page 586.

The LA administrator is called to make an 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, whereas the threads at SEA and SJ are committed. Processing continues at all systems. The DB2 subsystem at SEA has indoubt resolution responsibility with LA for LUWID=16.

Environment

The following figure illustrates the environment for this scenario.

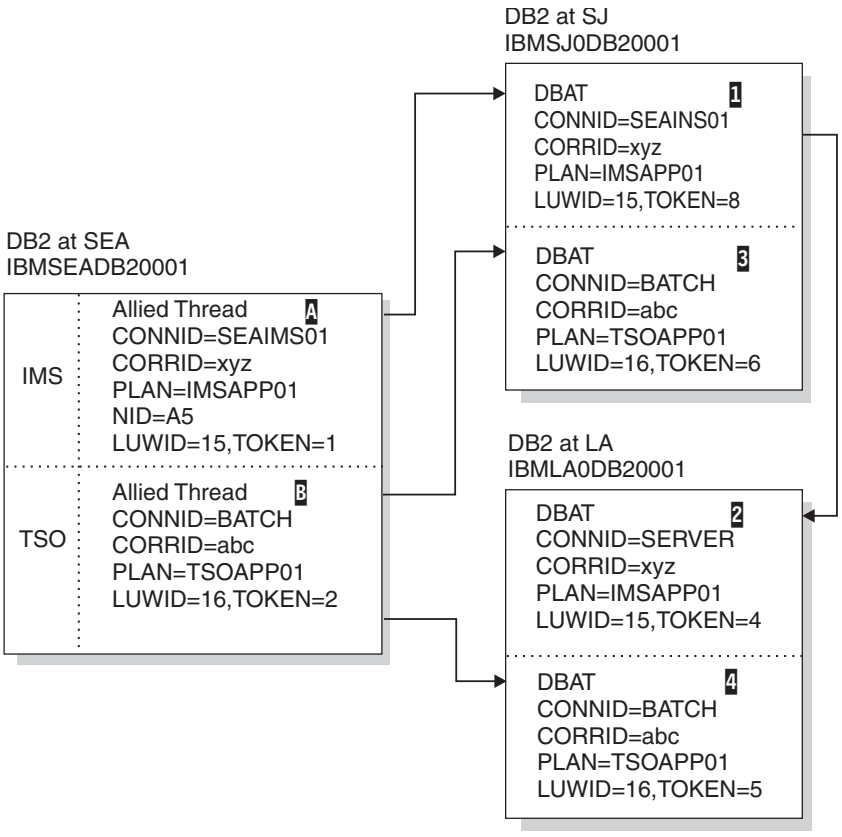


Figure 58. Resolution of indoubt threads. "Scenarios for resolving problems with indoubt threads" on page 584 contains a detailed description of the scenario depicted in this figure.

In this scenario, processing continues. Indoubt thread resolution responsibilities have been fulfilled, and the thread completes at both SJ and LA.

Resolving the problem

Database administrator response: 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, also. The available tools are:

- DSN1LOGP utility, which generates a summary report that identifies the table spaces that were modified by the LUWID=16 thread.
- The statistics trace class 4, which contains an IFCID 207 entry. This entry identifies the recovery log RBA for the LUWID=16 thread.

Notify IBM Software Support about the problem.

Chapter 14. Reading log records

Reading DB2 log records is useful for diagnostic and recovery purposes.

This information discusses three approaches to writing programs that read log records.

Contents of the log

The log contains the information that is needed to recover the results of program execution, the contents of the database, and the DB2 subsystem. The log does not contain information for accounting, statistics, traces, or performance evaluation.

PSPI

The three main types of log records are unit of recovery, checkpoint, and database page set control records.

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 utility.

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).

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)* identifies the log records of a data sharing member. The LRSN might not be unique on a data sharing member. The LRSN is a hexadecimal value derived from a store clock timestamp. DB2 uses the LRSN for recovery in the data sharing environment.

Effects of DB2 data compression

Log records can contain compressed data if a table contains compressed data. For example, if the data in a DB2 row is compressed, all data logged because of changes to that row is compressed. If logged, the record prefix is not compressed, but all of the data in the record is in compressed format. Reading compressed data requires access to the dictionary that was in use when the data was compressed.

PSPI

Data Replication's Q Capture program requires access to the dictionary for a source compressed table space when it processes an IFI 306 READS request. In this case, if the compressed table space is in a data sharing environment, a lock on the table

© Copyright IBM Corp. 1982, 2016

space may cause a Group Buffer Pool dependency (GBPDEP).

Related reference:

DSN1LOGP (DB2 Utilities)

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.

PSPI

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.

PSPI

Database exception table records

Database exception table (DBET) log records register different types of information: exception states and image copies of special table spaces.

PSPI

DBET log records also register exception information that is not related to units of recovery.

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.

Image copies of special table spaces

Image copies of DSNDB01.SYSUTILX, DSNDB01.DBD01, DSNDB06.SYSCOPY, and DSNDB01.SYSDBDXA 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.

PSPI

Related reference:

"Other exception information" on page 605

Related information:

DSNT361I (DB2 Messages)

Typical unit of recovery log records

A typical sequence of log records is written for an insert of one row through TSO.

PSPI

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 40. 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.

Table 40. Example of a log record sequence for an INSERT of one row using TSO (continued)

Type of record	Information recorded
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 41 shows the log records for processing and rolling back an insertion.

Table 41. 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.

PSPI

Types of changes to data

The three basic types of changes to a data page are changes to control information, changes to database pointers, and changes to the data.

PSPI

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. (A page in a catalog table space that has links can contain up to 127 rows.) 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.

The following table summarizes the information logged for data and index changes.

Table 42. 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[1]	The old and new values of the changed data. • On redo, the new data is replaced. • On undo, the old data is replaced.
Insert index entry	The new key value and the data RID.
Delete index entry	The deleted key value and the data RID.
Add column	The information about the column being added, if the table was defined with DATA CAPTURE(CHANGES).
Alter column	The information about the column being altered, if the table was defined with DATA CAPTURE(CHANGES).
Roll back to a savepoint	Information about the savepoint.
Modify table space	Information about the table space version.
LOAD SHRLEVEL NONE RESUME YES	The database ID (DBID) and the page set ID (PSID) of the table space on which the operation was run.
LOAD SHRLEVEL NONE RESUME NO REPLACE	The database ID (DBID) and the page set ID (PSID) of the table space on which the operation was run.
REORG TABLESPACE DISCARD	The database ID (DBID) and the page set ID (PSID) of the table space on which the operation was run.
CHECK DATA DELETE YES	The database ID (DBID) and the page set ID (PSID) of the table space on which the operation was run.

Table 42. Information logged for database changes (continued)

Operation	Information logged
Point-in-time recovery by using the RECOVER utility with the following options: • TOCOPY • TOLASTCOPY • TOLASTFULLCOPY • TORBA • TOLOGPOINT	The database ID (DBID) and the page set ID (PSID) of the table space on which the operation was run.
EXCHANGE DATA on a clone table space	The database ID (DBID) and the page set ID (PSID) of the table space on which the operation was run.
REPAIR SET DELETE	The database ID (DBID) and the page set ID (PSID) of the table space on which the operation was run.

Note:

1. If an update occurs to a table defined with DATA CAPTURE(CHANGES), the entire before-image of the data row is logged.

PSPI

Checkpoint log records

DB2 takes periodic checkpoints during normal operation in order to reduce restart time.

PSPI

DB2 takes checkpoints 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.
- 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. The following table summarizes the information logged.

Table 43. 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).

Table 43. Contents of checkpoint log records (continued)

Type of log record	Information logged
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. 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.
End_Checkpoint	Marks the end of the summary information about a checkpoint.

PSPI

Related reference:

DSNTIPL: Active log data set parameters (DB2 Installation and Migration)

"Database page set control records"

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).

PSPI

The same information is in the DB2 directory (SYSIBM.SYSLGRNX). It is also registered in the log so that it is available at restart.

PSPI

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 database exception table (DBET) log record.

The physical structure of the log

The active log consists of VSAM data sets with certain required characteristics.

PSPI

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.

PSPI

Physical and logical log records

The VSAM control interval (CI) provides 4089 bytes to hold DB2 information. That space is called a *physical record*. The information that is to be logged at a particular time forms a *logical record*, whose length varies independently of the space that is available in the CI.

PSPI

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).

Figure 59 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').

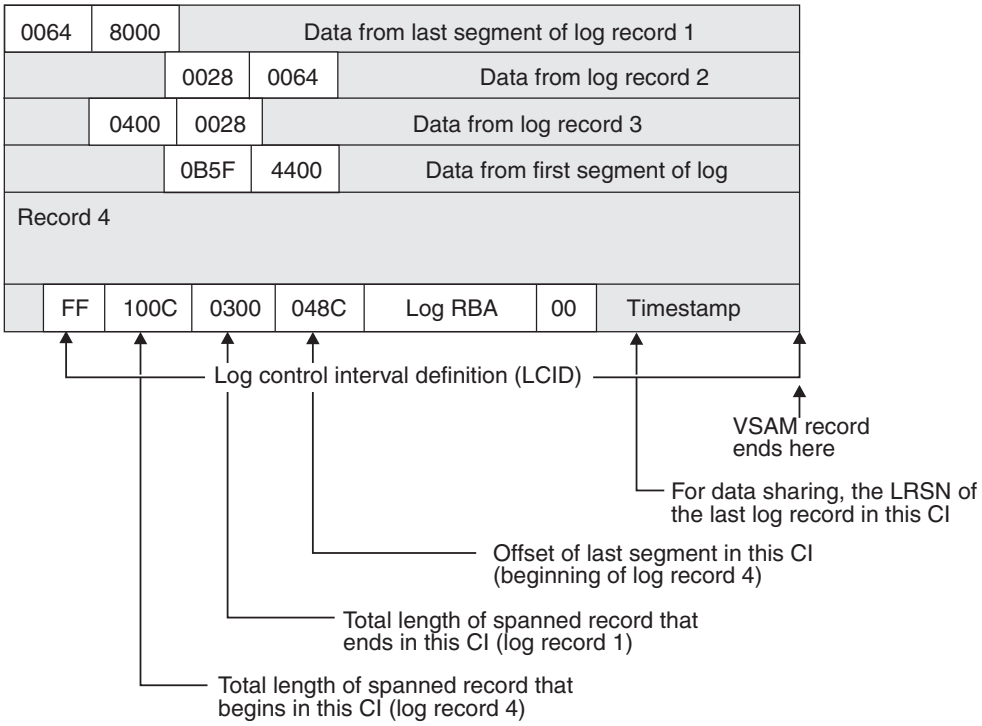


Figure 59. A VSAM CI and its contents

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*.

PSPI

Related reference:

"The log control interval definition (LCID)" on page 608

The log record header

Each logical record includes a prefix, called a *log record header* (LRH), which contains control information.

PSPI

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 44. Contents of the log record header

Hex offset	Length	Information
00	2	Length of this record or segment
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
06	2	Subtype of the log record
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; otherwise, 0
10	6	Log RBA of the previous log record, if this record relates to a unit of recovery; otherwise, 0
16	1	Release identifier
17	1	Length of header
18	6	Undo next LSN
1E	8	LRHTIME

PSPI

Related concepts:

"Unit of recovery log records" on page 600

Related reference:

"Log record type codes" on page 610
"Log record subtype codes" on page 611

The log control interval definition (LCID)

Each physical log block, also referred to as a control interval, includes a suffix called the *log control interval definition* (LCID), which tells how record segments are placed in the physical control interval.

PSPI

The following table describes the contents of the LCID.

Table 45. 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. The following illustration shows the format schematically; the following list describes each field.

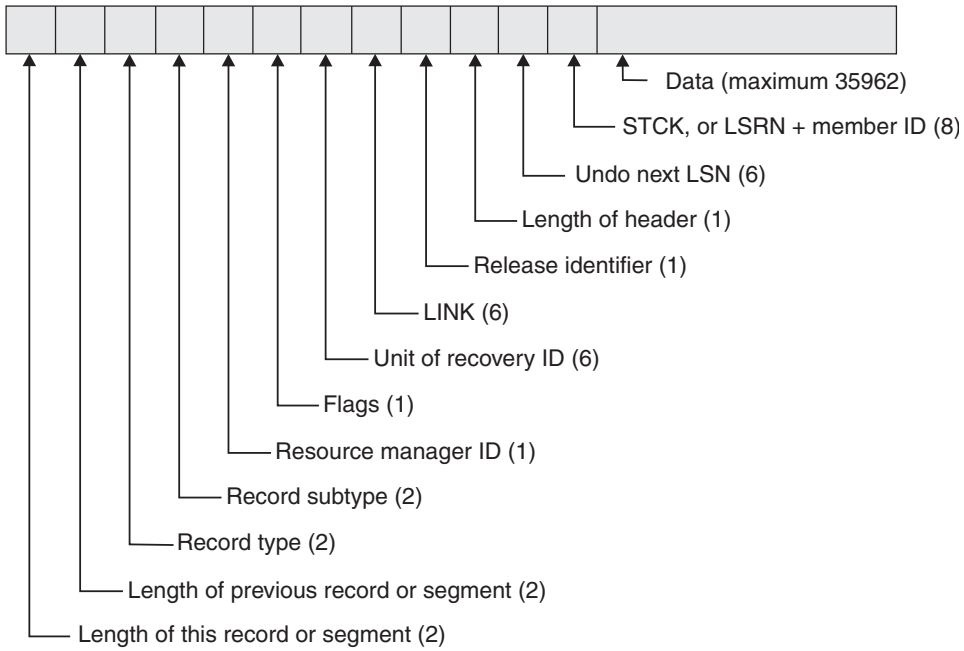


Figure 60. Format of a DB2 recovery log record

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.

Subtype
Some types of recovery log records are further divided into subtypes.

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.

The fields are:

PSPI

Related reference:
"Log record type codes"
"Log record subtype codes" on page 611

Log record type codes

The type code of a log record tells what kind of DB2 event the record describes.

PSPI

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
4200	End of rollback to savepoint
4400	Alter or modify recovery log record

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. A diagnostic log record for the TRUNCATE IMMEDIATE statement is type code 4200, which is a combination of a diagnostic log record (4000) and an UNDO record (0200). PSPI

Log record subtype codes

The log record subtype code provides a more granular definition of the event that occurred and that generated the log record. Log record subtype codes are unique only within the scope of the corresponding log record type code.

PSPI

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, DSNDB01.DBD01, and DSNDB01.SYSDBDXA).

Log record type 0800 (quiesce) does not have subtype codes.

Some log record types (1000 - 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	Roll back to savepoint
000E	Release to savepoint

Subtypes for type 4200 (end of rollback to savepoint)

Code	Type of event
0084	End of savepoint
0085	End of savepoint

Subtypes for type 4200 (diagnostic log record for TRUNCATE IMMEDIATE)

Code	Type of event
0085	Special begin for TRUNCATE IMMEDIATE
0086	Special commit for TRUNCATE IMMEDIATE

Subtypes for type 4400 (alter or modify log record)

Code	Type of event
0083	Alter or modify log record used for DB2 replication

PSPI

Related reference:

DSN1LOGP (DB2 Utilities)

Interpreting data change log records

For specific log record types, DB2 provides the mapping and description that you can use to interpret data changes that are made to DB2 tables from the log.

PSPI

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 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.

PSPI

Related reference:

"Log record subtype codes" on page 611

Reading log records with IFI

You can use the READA (read asynchronously) request of the instrumentation facility interface (IFI) to read log records into a buffer. Use the READS (read synchronously) request to read specific log control intervals from a buffer. You can use these requests online while DB2 is running.

About this task

PSPI

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.

Restriction: Either the primary authorization ID or one of the secondary authorization IDs must have the MONITOR2 privilege.

PSPI

Related concepts:

Programming for the instrumentation facility interface (IFI) (DB2 Performance)

Related tasks:

Requesting data synchronously from a monitor program (DB2 Performance)

Requesting data asynchronously from a monitor program (DB2 Performance)

Related reference:

READA (DB2 Performance)

READS (DB2 Performance)

Gathering active log records into a buffer

Use the START TRACE command to begin gathering active log records into a buffer.

Procedure

PSPI

To gather active log records:

Issue the following START TRACE command in an instrumentation facility interface (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.

PSPI

Reading specific log records (IFCID 0129)

You can use IFCID 129 with an IFI READS (read synchronously) request to return a specific range of log records from the active log into the return area that is initialized by your program.

About this task

PSPI

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 routine. 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.

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.

PSPI

Related concepts:

"Log capture routines" on page 661

DB2 trace output (DB2 Performance)

Related reference:

Qualification fields for READS requests (DB2 Performance)

Reading complete log data (IFCID 0306)

Several benefits are associated with use of IFCID 0306 to read log data.

About this task

PSPI The benefits of 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.

Procedure

To use IFCID 0306:

Use the same call that you use for IFCID 0129. IFCID 0306 must appear in the IFCID area. IFCID 0306 returns complete log records and the spanned record indicators in bytes 2 have no meaning, if present. Multi-segmented control interval log records are combined for a complete log record.
Generally, catalog and directory objects cannot be in group buffer pool RECOVER-pending (GRECP) status when an IFCID 0306 request accesses the compression dictionary. Only log entries for tables that are defined with DATA CAPTURE CHANGES enabled are decompressed.

PSPI

Related tasks:

"Reading specific log records (IFCID 0129)" on page 614

Specifying the return area

You must specify the return area for IFCID 0306 requests.

About this task

PSPI

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. An additional 60-byte area 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 IFI application program must run in supervisor state to request the key 7 return area. The storage size of the return area must 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.

Important: The recommended size of the ECSA return area for IFCID 306 requests is between 66 KB and 200 KB. Specifying more than 200 KB for this return area might cause performance problems or even failures for other critical application programs that depend on ECSA storage. Because the log record counter is a 2 byte value, IFCID 306 might also return an incorrect count of log records if too much space is specified for the ECSA return area.

IFCID 0306 has a unique return area format. The first section is mapped by QW0306OF instead of the write header DSNDQWIN.

PSPI

Related concepts:

Programming for the instrumentation facility interface (IFI) (DB2 Performance)

Related reference:

Instrumentation facility communications area (IFCA) (DB2 Performance)

Return area (DB2 Performance)

Trace fields for READS requests (DB2 Performance)

Qualifying log records

To retrieve IFCID 0306 log records, your program must initialize certain fields in the qualification area that is mapped by DSNDWQAL.

PSPI

These qualification fields are:

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 that is 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. If previous READS requests returned 00E60812 (successful end of READS), for a data sharing environment you can use either QW0306EOS and QW0306EOS+1 in the next F call. For a non-data sharing environment, use QW0306EOS+1 as the start of the log read in the next F call. The F call returns the next range from the last RBA that was provided. Consider the following example:

First, the READS request ends with 00E60812:

```
                  WQALLMOD  WQALLRBA
                  --------  ------------
READS input:            C6  CAC5B606C843

                  QW0306ES     QW0306CT  IFCARC1 IFCARC2  IFCABM
                  ------------ --------  ------- -------- --------
READS output:     CAC5B606CB6C 0060           04 00E60812 00011F3F
```

In the next F call for a data sharing environment, you specify either QW0306EOS and QW0306EOS+1 as the input for WQALLRBA:

```
WQALLMOD  WQALLRBA
                  --------  ------------
READS input:            C6  CAC5B606CB6C

                  QW0306ES     QW0306CT  IFCARC1 IFCARC2  IFCABM
                  ------------ --------  ------- -------- --------
READS output:     CAC5B606CE91 008E           04 00E60812 00004B7B
```

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.

WQALWQLS
In this 64-bit pointer, specify the address of an optional qualification block named WQLS, and mapped by DSNDWQAL, to filter log records returned for IFCID 0306. The qualification block consists of a header of type 'DBPS', count of qualification items up to 50,000, and a list containing DBID and PSID pairs or DBID and OBID pairs. For table space log records, include DBID and PSID pairs. For table log records, include DBID and OBID pairs. WQALLCRI must be set to X'00' for this additional qualification.

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.

PSPI

Reading log records with OPEN, GET, and CLOSE

You can use the assembler language DSNJSLR macro to submit OPEN, GET, and CLOSE functions. Use this stand-alone method to capture log records that you cannot read with the instrumentation facility interface (IFI) when DB2 stops running.

About this task

PSPI

DB2 provides the following 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:

- The OPEN function initializes stand-alone log services.
- The GET function returns a pointer to the next log record or log record control interval.
- The CLOSE function deallocates data sets and frees storage.

To invoke these services, use the assembler language DSNJSLR macro and specify 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.

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.

Important: Use one of the following methods to read active logs while the DB2 subsystem that owns the logs is active:

- IFCID 0129
- IFCID 0306
- Log capture exit

There are no restrictions on reading archive logs.

PSPI

JCL DD statements for DB2 stand-alone log services

Stand-alone services, such as OPEN, GET, and CLOSE, use a variety of JCL DD statements as they operate.

PSPI

The following tables list and describe the JCL DD statements that are used by stand-alone services.

Table 46. JCL DD statements for DB2 stand-alone log services

JCL DD statement	Explanation
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. 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.
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.

Table 47. JCL DD statements for DB2 stand-alone log services in a data-sharing environment

JCL DD statement	Explanation
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.
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:

- An 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.
- An 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".

PSPI

Related reference:

"Stand-alone log CLOSE request" on page 627

"Stand-alone log OPEN request" on page 623

"Stand-alone log GET request" on page 625

Data sharing members that participate in a read

The number of data sharing members whose logs participate in a particular read request varies based on what statements are used.

PSPI

If you use the GROUP DD statement, then the determinant is the number of members in the group. 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.

You then 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.

PSPI

Registers and return codes

DB2 uses registers to store important information and return codes to help you determine the status of stand-alone log activity.

PSPI

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 the following table.

Table 48. 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).

PSPI

Stand-alone log OPEN request

A stand-alone log OPEN request initializes the stand-alone log services.

PSPI

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.

GPR 0 General-purpose register 0 contains a reason code associated with a nonzero return code in register 15.

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

Log control intervals are returned in their actual format and the caller is responsible for using the correct mapping. The format of the control interval can be determined by examining the first bit of the second to last byte (offset 4094).

PSPI

Related reference:

"JCL DD statements for DB2 stand-alone log services" on page 620

"Registers and return codes" on page 623

Stand-alone log GET request

A stand-alone log GET request returns a pointer to a buffer that contains the next log record, based on position information in the request block.

PSPI

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.

GPR 0 General-purpose register 0 contains a reason code associated with a nonzero return code in register 15.

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.

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 the following table. 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 49. 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

PSPI

Related concepts:

X'D1.....' codes (DB2 Codes)

Related tasks:

Chapter 13, "Recovering from different DB2 for z/OS problems," on page 471

Related reference:

"JCL DD statements for DB2 stand-alone log services" on page 620

"Registers and return codes" on page 623

Stand-alone log CLOSE request

A stand-alone log CLOSE request deallocates any log data sets that were dynamically allocated by previous processing. In addition, all storage that was obtained by previous functions, including the request block that is specified on the request, is freed.

PSPI

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.

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.

PSPI

Related reference:

"JCL DD statements for DB2 stand-alone log services" on page 620

"Registers and return codes" on page 623

Related information:

00D10030 (DB2 Codes)

Sample application that uses stand-alone log services

Sample segments of an assembler program use three stand-alone log services (OPEN, GET, and CLOSE) to process one log record.

PSPI

For example:

```
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 61. Excerpts from a sample program using stand-alone log services

PSPI

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. You can use this exit routine online while DB2 is running.

About this task

PSPI This installation exit routine presents log data to a log capture exit routine when the data is written to the DB2 active log. Do not use this exit routine for general purpose log auditing or tracking. The IFI interface is designed 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.

Procedure

To capture log records with the log capture exit routine:

You must write an exit routine (or use the one that is provided by the preceding program offering) that can be loaded and called under the various processing conditions and restrictions that are required by this exit routine. PSPI

Related concepts:

"Contents of the log" on page 599

"Log capture routines" on page 661

Related tasks:

"Reading log records with IFI" on page 613

Related reference:

"The physical structure of the log" on page 605

Part 3. Appendixes

© Copyright IBM Corp. 1982, 2016

Appendix A. Exit routines

DB2 provides installation-wide exit points to the routines that you provide. These exit routines are described in the following sections:

- "Edit procedures"
- "Validation routines" on page 637
- "Date and time routines" on page 641
- "Conversion procedures" on page 645
- "Field procedures" on page 648
- "Log capture routines" on page 661
- "Routines for dynamic plan selection in CICS" on page 664.

Related information:

Managing access through exit routines (Managing Security)

Edit procedures

An edit procedure is assigned to a table by the EDITPROC clause of the CREATE TABLE statement. An edit procedure receives the entire row of a base table in internal DB2 format. It can transform the row when it is stored by an INSERT or UPDATE SQL statement or by the LOAD utility.

PSPI

An edit procedure can be defined as WITH ROW ATTRIBUTES or WITHOUT ROW ATTRIBUTES in a CREATE TABLE statement. An edit procedure that is defined as WITH ROW ATTRIBUTES uses information about the description of the rows in the associated table. You cannot define an edit routine as WITH ROW ATTRIBUTES on a table that has the following characteristics:

- The table contains a LOB, ROWID, or XML column.
- The table contains an identity column.
- The table contains a security label column.
- The table contains a column name that is longer than 18 EBCDIC bytes.

You cannot define an edit procedure as WITHOUT ROW ATTRIBUTES on a table that has LOB columns.

The transformation your edit procedure 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*.

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 procedure 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,

© Copyright IBM Corp. 1982, 2016

therefore, index keys are stored in the index in *edit-decoded* form. Hence, for a table with an edit procedure, index keys in the table **are** edit-coded; index keys in the index are **not** edit-coded.

The sample application contains a sample edit procedure, 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.

PSPI

Related concepts:

General guidelines for writing exit routines

Specifying edit procedures

If you plan to use an edit procedure, you can specify it when you create the table. However, you cannot add an edit procedure to an existing table, or alter a table with an edit procedure to add a column; you must drop and re-create the table.

Procedure

PSPI

To specify an edit procedure for a table:

Specify the EDITPROC clause of the CREATE TABLE statement, followed by the name of the procedure. The procedure is loaded on demand during operation. You can specify the EDITPROC clause on a table that is activated with row and column access control. The rows of the table are passed to these procedures if your security administrator determines that these procedures are allowed to access sensitive data.

PSPI

When edit routines are taken

An edit routine is invoked to edit-encode a row whenever an SQL statement or LOAD utility job inserts or updates the row.

PSPI

An edit routine 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.

PSPI

Parameter list for edit procedures

The parameter list for edit procedures contains pointers to other information, including the authorization ID list.

PSPI

At invocation, registers are set, and the edit procedure uses the standard exit parameter list (EXPL). The following table shows the exit-specific parameter list, as described by macro DSNDEDIT.

Table 50. Parameter list for an edit procedure

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. The value of this parameter is 0 (zero) if both of the following conditions are true: • The edit procedure is insensitive to the format in which DB2 stores the rows of the table. • The edit procedure is defined as WITHOUT ROW ATTRIBUTES in CREATE TABLE statements.
	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

PSPI

Incomplete rows and edit routines

DB2 passes input rows to an edit routine. If an input row has fewer fields than the number of columns in the table, the edit routine must stop processing the row after the last input field.

PSPI

Columns for which no input field is provided and that are not in reordered row format are always at the end of the row and are never defined as NOT NULL. In this case, the columns allow nulls, they are defined as NOT NULL WITH DEFAULT, or the columns are ROWID or DOCID columns.

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.

The following diagram shows how the parameter list points to other row information. The address of the *n*th column description is given by: RFMTAFLD + (*n*-1)*(FFMTE-FFMT).

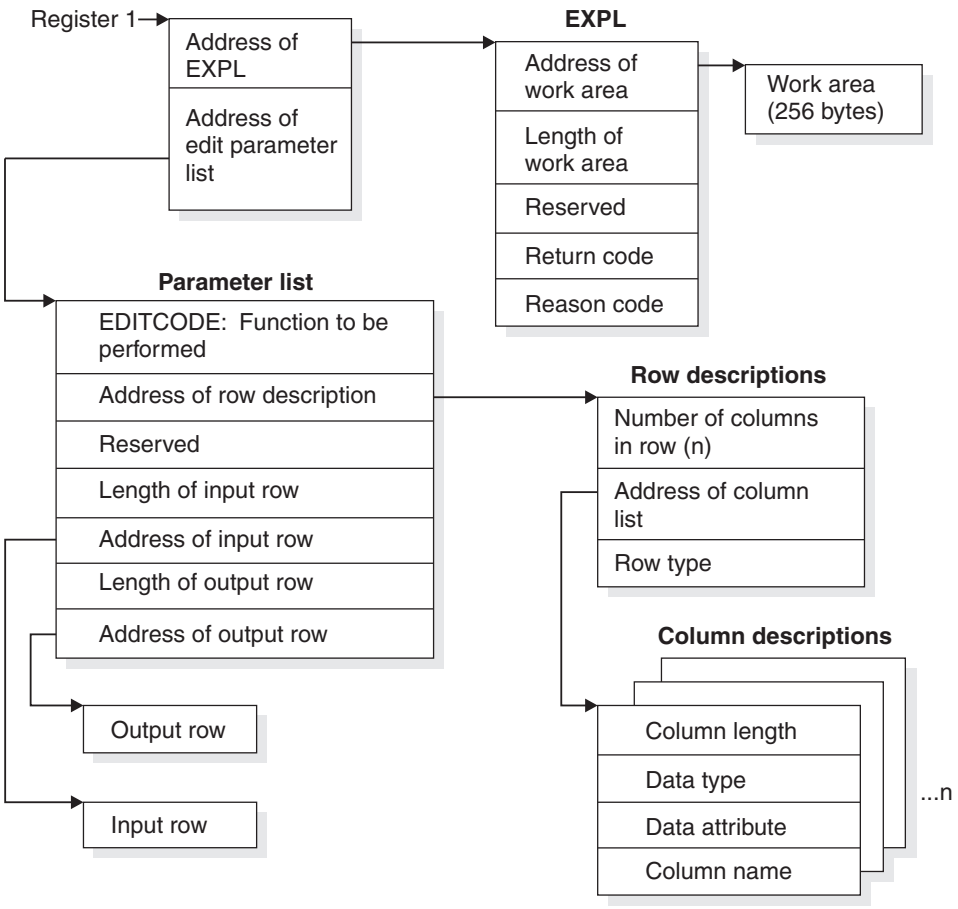


Figure 62. How the edit exit parameter list points to row information

PSPI

Expected output for edit routines

The edit routines return different output depending on whether the input row is in the coded or decoded form.

PSPI

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 for a row in basic row format, "record" includes fields for the lengths of varying-length columns and for null indicators. In counting the maximum length for a row in reordered row format, "record" includes fields for the offsets to the varying length columns and for null indicators. The maximum length of the record 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. In counting the maximum length for a row in basic row format, "record" includes fields for the lengths of varying length columns and for null indicators. In counting the maximum length for a row in reordered row format, "record" includes fields for the offsets to the varying-length columns and for null indicators. The maximum length of the record does not include 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 following meanings:

Table 51. 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).

PSPI

Validation routines

Validation routines are assigned to a table by the VALIDPROC clause of the CREATE TABLE and ALTER TABLE statement. A validation routine receives an entire row of a base table as input. The routine can return an indication of whether to allow a subsequent INSERT, UPDATE, DELETE, FETCH, or SELECT operation.

PSPI

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 or XML column, the LOB or XML values are not passed to the validation routine. The LOB indicator column takes the place of the LOB column, and the XML indicator column takes the place of the XML column. You cannot use VALIDPROC on a table if the table contains a column name that is longer than 18 EBCDIC bytes.

The return code from a validation routine is checked for a 0 value before any insert, update, or delete is allowed.

PSPI

Related concepts:

General guidelines for writing exit routines

Specifying validation routines

When you specify a validation routine for a table, the routine is loaded on demand during operation.

About this task

You can add a validation routine to an existing table, but the routine is not invoked to validate data that was already in the table.

Procedure

PSPI

To specify a validation routine for a table:

Issue the CREATE TABLE or ALTER TABLE statement with the VALIDPROC clause.
You can specify the VALIDPROC clause on a table that is activated with row and column access control. The rows of the table are passed to these routines if your security administrator determines that these routines are allowed to access sensitive data.
You can cancel a validation routine for a table by specifying the VALIDPROC NULL clause in an ALTER TABLE statement.

PSPI

When validation routines are taken

A validation routine for a table is invoked when DB2 inserts or updates a row, even for inserts that the LOAD utility makes.

PSPI

The routine is invoked for most delete operations, including a mass delete of all the rows of a table. 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.

PSPI

Parameter list for validation routines

At invocation, registers are set, and the validation routine uses the standard exit parameter list (EXPL).

PSPI

The following diagram shows how the parameter list points to other information.

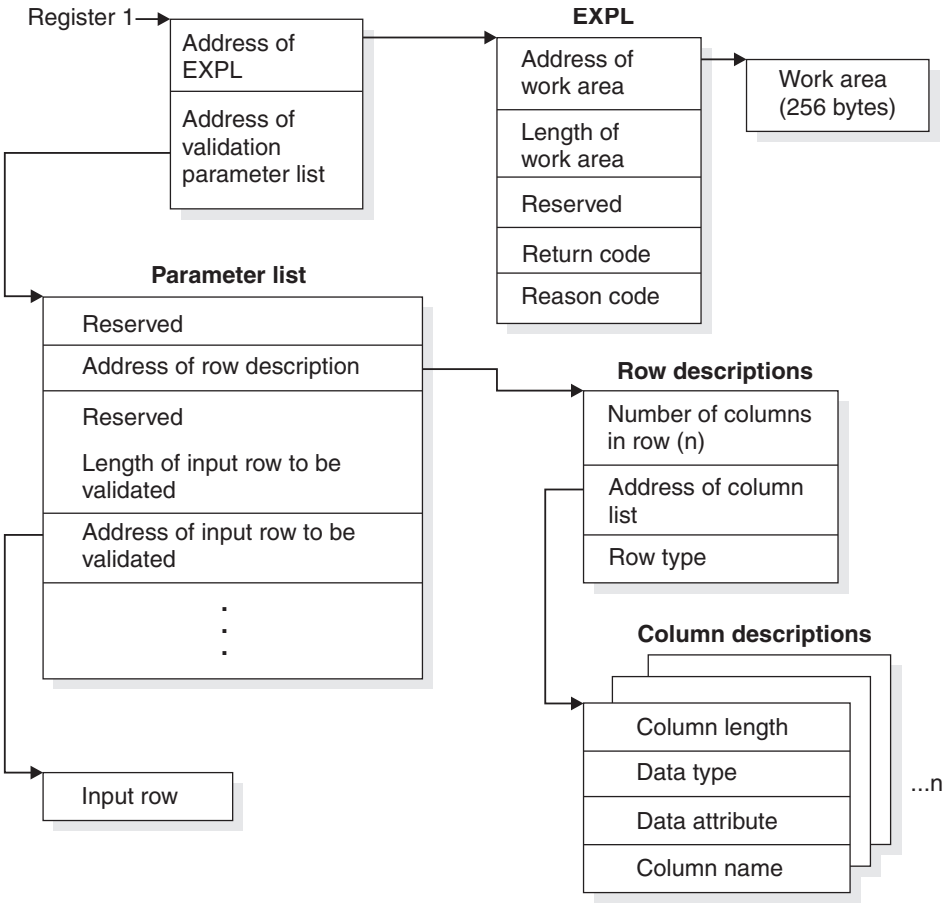


Figure 63. How a validation parameter list points to information. The address of the *n*th column description is given by: RFMTAFLD + (*n*-1)*(FFMTE-FFMT).

The following table shows the exit-specific parameter list, described by macro DSNDRVAL.

Table 52. 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.
	8	Signed 4-byte integer	Reserved
RVALROWL	C	Signed 4-byte integer	Length of the input row to be validated

Table 52. Parameter list for a validation routine (continued)

Name	Hex offset	Data type	Description
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.

PSPI

Incomplete rows and validation routines

DB2 passes input rows to a validation routine. If an input row has fewer fields than the number of columns in the table, the routine must stop processing the row after the last input field.

PSPI

Columns for which no input field is provided and that are not in reordered row format are always at the end of the row and are never defined as NOT NULL. In this case, the columns allow nulls, they are defined as NOT NULL WITH DEFAULT, or the columns are ROWID or DOCID columns.

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.

PSPI

Expected output for validation routines

The validation routines must leave a return code in EXPLRC1.

PSPI

The return code in EXPLRC1 have one of the following the meanings:

Table 53. 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).

PSPI

Date and time routines

A *date routine* is a user-written exit routine to change date values from a locally defined format to a format that is recognized by DB2 and from the ISO format to the locally defined format. Similarly, a *time routine* changes time values from a locally defined format to one that is recognized by DB2and from the ISO format to the locally-defined format.

PSPI

The following is a list of the formats recognized by DB2.

Table 54. 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, 2006". You can use a date routine that transforms the date to a format that is recognized by DB2 on insertion, such as ISO: "2006-09-21". On retrieval, the routine can transform "2006-09-21" to "September 21, 2006".

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.

PSPI

Related concepts:

General guidelines for writing exit routines

Specifying date and time routines

During DB2 installation, you can establish a date routine or time routine.

Procedure

PSPI To specify date and time routines:

1. Set LOCAL DATE LENGTH or LOCAL TIME LENGTH to the length of the longest field that is 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, 2006", you need an 18-byte field. You would set LOCAL DATE LENGTH to 18.
2. Replace all of the IBM-supplied exit routines. Use 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.
3. To make the local date or time format the default for retrieval, set DATE FORMAT or TIME FORMAT to LOCAL when installing DB2.

   This specification has the effect that DB2 *always* takes the exit routine when you retrieve from a DATE or TIME column. For example, suppose that you want to retrieve dates in your local format only occasionally; most of the time you use the USA format. You would set DATE FORMAT to USA.

What to do next

The installation parameters for LOCAL DATE LENGTH, LOCAL TIME LENGTH, DATE FORMAT, and TIME FORMAT can also be updated after DB2 is installed. If you change a length parameter, you might need to rebind the applications.

PSPI

When date and time routines are taken

A date or time routine is invoked to change a value from the locally defined format to a format that is recognized by DB2.

A date or time routine is invoked in the following circumstances:

- PSPI

  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.

PSPI

Parameter list for date and time routines

At invocation, registers are set, and the date or time routine uses the standard exit parameter list (EXPL).

PSPI

The following diagram shows how the parameter list points to other information.

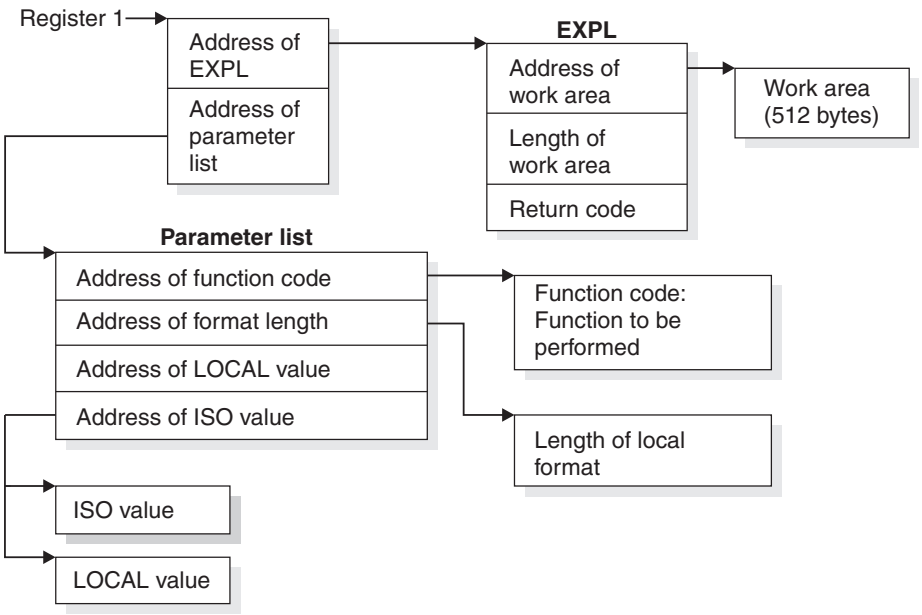


Figure 64. How a date or time parameter list points to other information

The following list shows the exit-specific parameters, described by macro DSNDDTXP.

Table 55. 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.

PSPI

Expected output for date and time routines

The date and time routines return different output depending on the format of the input value.

PSPI

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 following meanings:

Table 56. 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.

PSPI

Conversion procedures

A conversion procedure is a user-written exit routine that converts characters from one coded character set to another coded character set.

PSPI

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.

PSPI

Related concepts:

General guidelines for writing exit routines

Specifying conversion procedures

You can specify a conversion procedure that converts characters from one coded character set to another coded character set.

About this task

Procedure

To specify a conversion procedure:

Insert a row into the SYSIBM.SYSSTRINGS catalog table.

PSPI

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.

PSPI

When conversion procedures are taken

A conversion procedure is invoked when it is required to convert the coded character set that is identified by INCCSID to one that is identified by OUTCCSID.

Parameter list for conversion procedures

At invocation, registers are set, and the conversion procedure uses the standard exit parameter list (EXPL).

PSPI

A conversion procedure does *not* use an exit-specific parameter list. 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 following formats:

Table 57. 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. 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.

PSPI

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.

PSPI

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.

The following is a list of the codes for the converted string in FPVDVALE:

Table 58. Codes for the converted string in FPVDVALE

Code	Meaning
0	Successful conversion
4	Conversion with substitution

For the following remaining codes, DB2 does not use the converted string:

Table 59. 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).

The following diagram shows how the parameter list points to other information.

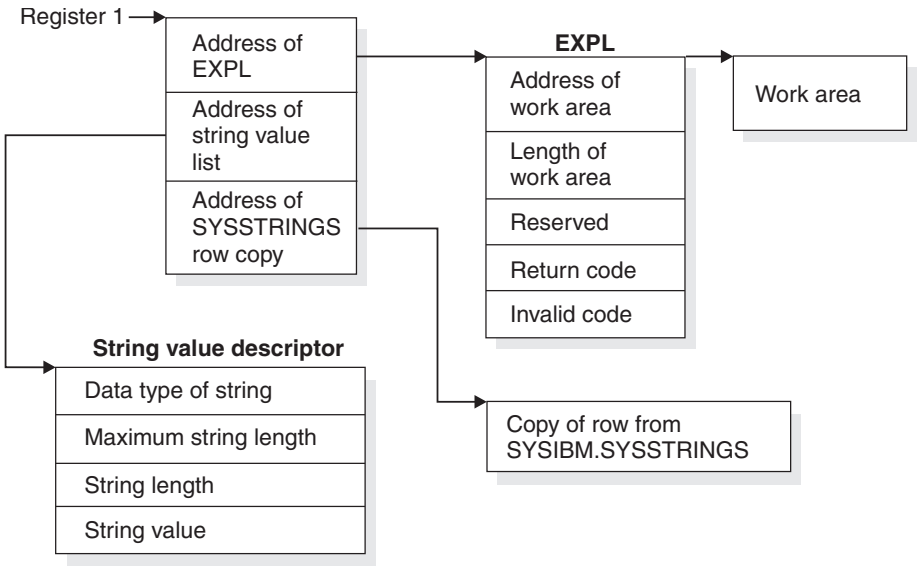


Figure 65. Pointers at entry to a conversion procedure

PSPI

Field procedures

A field procedure is a user-written exit routine that is used to transform values in a single, short string column. You can assign field procedures to a table by specifying the FIELDPROC clause of the CREATE TABLE or ALTER TABLE statement.

PSPI

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.

PSPI

Related concepts:

“General guidelines for writing exit routines” on page 664

Field-definition for field procedures

A field procedure is invoked when a table is created or altered to define the data type and attributes of an encoded value to DB2. This operation is called *field-definition*.

PSPI

The data type of the encoded value can be any valid SQL data type except DATE, TIME, TIMESTAMP, LONG VARCHAR, or LONG VARGRAPHIC. 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.

PSPI

Related reference:

“Value descriptor for field procedures” on page 654

Specifying field procedures

To transform values in single, short string columns, you can assign field procedures to the columns. If you plan to use a field procedure, specify it when you create the table. In operation, the procedure is loaded on demand.

About this task

PSPI

Restriction: Consider the following restrictions:

- You cannot use a field procedure on a column that was defined by using the NOT NULL WITH DEFAULT clause.
- You cannot add a field procedure to an existing column of a table. However, you can use the ALTER TABLE statement to add a new column that uses a field procedure to an existing table.
- You cannot use a field procedure on a LOB, ROWID, or ROW CHANGE TIMESTAMP column of a table. However, you can specify it for other columns in the same table.
- You cannot use a field procedure on a column if the column name is longer than 18 EBCDIC bytes.

Procedure

To specify a field procedure for a column:

Issue the CREATE TABLE or ALTER TABLE statement with the FIELDPROC clause.
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*. The modified FPPVL 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.

PSPI

When field procedures are taken

A field procedure that is specified for a column is invoked in certain conditions.

A field procedure is invoked generally in three conditions:

- PSPI

  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.
- 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.

- 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.

Recommendation: Avoid encoding blanks in a field procedure. 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.

PSPI

Control blocks for execution of field procedures

Certain control blocks are used to communicate to a field procedure.

Parameter list (FPPL) for field procedures

The field procedure parameter list is pointed to by register 1 on entry to a field procedure. It contains the addresses of five other areas.

PSPI

The following diagram shows those areas. The FPPL and the areas are described by the mapping macro DSNDFPPB.

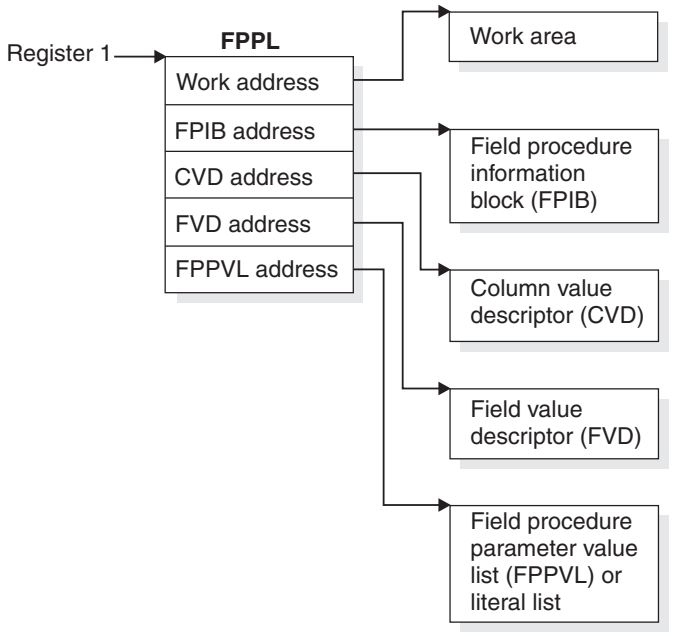


Figure 66. Field procedure parameter list

PSPI

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.

PSPI

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.

PSPI

Information block (FPIB) for field procedures

The field procedure information block communicates general information to a field procedure.

PSPI

The information block 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 following formats:

Table 60. 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

PSPI

Parameter value list (FPPVL) for field procedures

The field procedure parameter value list communicates the literal list to the field procedure during the field-definition process. The literal list is supplied in the CREATE TABLE or ALTER TABLE statement.

PSPI

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 following formats:

Table 61. 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.

Table 61. Format of FPPVL, defined in copy macro DSNDFPPB (continued)

Name	Hex offset	Data type	Description
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	Each parameter in the FIELDPROC clause has: 1. A signed 4-byte integer giving the length of the following value descriptor, which includes the lengths of FPVDTYPE, FPVDVLEN, and FPVDVALE. 2. A value descriptor

PSPI

Value descriptor for field procedures

A value descriptor describes the data type and other attributes of a value.

PSPI

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 following formats:

Table 62. 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.

PSPI

Related reference:

"Field-definition for field procedures" on page 649

Field-definition (function code 8)

You need to provide the input and output that are required for a field-definition operation.

On entry

The input that provided to the field-definition operation and the output that is required are as follows:

PSPI

The registers have the following information:

Table 63. Contents of the registers on entry

Register	Contains
1	Address of the field procedure parameter list (FPPL)
2 through 12	Unknown values that must be restored on exit.
13	Address of the register save area.
14	Return address.

Table 63. Contents of the registers on entry (continued)

Register	Contains
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 following information:

Table 64. 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 following information:

Table 65. 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 field has the information:

Table 66. 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.
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 following information:

Table 67. 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 following information:

Table 68. 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 following information:

Table 69. 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 62 on page 655 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.

PSPI

Field-encoding (function code 0)

You need to provide the input and output that are required for a field-encoding operation.

On entry

The input that is provided to the field-encoding operation, and the output that is required, are as follows:

PSPI

The registers have the following information:

Table 70. Contents of the registers on entry

Register	Contains
1	Address of the field procedure parameter list (FPPL); see "Parameter list (FPPL) for field procedures" on page 651 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 following information:

Table 71. Contents of the FPIB on entry

Field	Contains
FPBFCODE	0, the function code
FPBWKLN	The length of the work area

The **CVD** has the following following information:

Table 72. Contents of the CVD on entry

Field	Contains
FPVDTYPE	The numeric code for the data type of the column value, as shown in Table 62 on page 655.
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 following information:

Table 73. 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 following information:

Table 74. 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 following information:

Table 75. 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.

PSPI

Field-decoding (function code 4)

You need to provide the input and output that are required for a field-decoding operation.

On entry

PSPI

The registers have the following information:

Table 76. Contents of the registers on entry

Register	Contains
1	Address of the field procedure parameter list (FPPL); see Figure 66 on page 652 for a schematic diagram.

Table 76. Contents of the registers on entry (continued)

Register	Contains
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 following information:

Table 77. Contents of the FPIB on entry

Field	Contains
FPBFCODE	4, the function code
FPBWKLN	The length of the work area

The **CVD** has the following information:

Table 78. Contents of the CVD on entry

Field	Contains
FPVDTYPE	The numeric code for the data type of the column value, as shown in Table 62 on page 655.
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 following information:

Table 79. 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 following information:

Table 80. Contents of the registers on exit

Register	Contains
2 through 12	The values they contained on entry.

Table 80. Contents of the registers on exit (continued)

Register	Contains
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 following information:

Table 81. 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.

PSPI

Log capture routines

A log capture exit routine makes DB2 log data available for recovery purposes in real time.

PSPI

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.

"General guidelines for writing exit routines" on page 664 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.

PSPI

Specifying log capture routines

You can specify a log capture routine to receive data when DB2 writes data to the active log.

About this task

The module name for the log capture routine is DSNJL004, and its entry point is DSNJW117. The module is loaded during DB2 initialization and deleted during DB2 termination.

Procedure

PSPI

To specify a log capture routine:

Link module DSNJL004 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).

PSPI

When log capture routines are invoked

The log capture exit routine is invoked in three situations. Each situation is identified by a character in the parameter list of the routine.

PSPI 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 used, 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.

  On 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.

  Performance consideration: All processing time that is 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. 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 timestamp to determine the last occurrence of the control interval. This last occurrence should replace all others. The timestamp is found in the control interval. PSPI

Parameter list for log capture routines

At invocation, registers are set, and the log capture routine uses the standard exit parameter list (EXPL). You can ignore the reason and return codes in that list.

PSPI

The following is a list of the exit-specific parameters; it is mapped by macro DSNDLOGX.

Table 82. 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

Table 82. Log capture routine specific parameter list (continued)

Name	Hex offset	Data type	Description
LOGXSTIM	30	Character, 8 bytes	DB2 subsystem startup time (TIME format with DEC option: 0CYYDDDFHHMMSSTH)
LOGXREL	38	Character, 3 bytes	DB2 subsystem release level: '810' for Version 8 '910' for Version 9 '101' for Version 10
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.

PSPI

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.

PSPI

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.

PSPI

General guidelines for writing exit routines

When you use the exit routines that DB2 supplies, consider some of the general rules, requirements, and guidelines for using exit routines.

PSPI

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.

PSPI

Related concepts:

Connection routines and sign-on routines (Managing Security)

Access control authorization exit routine (Managing Security)

Coding rules for exit routines

You must follow certain rules and requirements when coding an exit routine for DB2.

•

PSPI

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.
- The following routines *must* have the names shown. The name of other routines should not start with "DSN", to avoid conflict with the DB2 modules.

Table 83. 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.
- 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.

PSPI

Modifying exit routines

Because exit routines operate as extensions of DB2, avoid changing or modifying exit routines while DB2 is running.

Execution environment for exit routines

Exit routines are invoked by standard CALL statements.

PSPI

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.

PSPI

Registers at invocation for exit routines

Registers are set when DB2 passes control to an exit routine.

PSPI

The following are registers that are set at invocation for exit routines:

Table 84. Contents of registers when DB2 passes control to an exit routine

Register	Contains
1	Address of pointer to the exit parameter list. *For a field procedure*, the address is that of the field procedure parameter list.
13	Address of the register save area.
14	Return address.
15	Address of entry point of exit routine.

PSPI

Parameter list for exit routines

The parameter list for an exit routine contains pointers to other information, which generally includes the EXPL parameter list and the exit-specific parameter list.

PSPI

Register 1 points to the address of parameter list EXPL, described by macro DSNDEXPL. The word following points to a second parameter list, which differs for each type of exit routine.

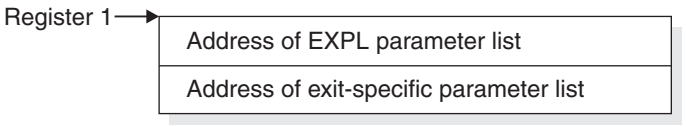


Figure 67. Use of register 1 on invoking an exit routine. (Field procedures and translate procedures do not use the standard exit-specific parameter list.)

The following is a list of the EXPL parameters. Its description is given by macro DSNDEXPL:

Table 85. 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

Table 85. Contents of EXPL parameter list (continued)

Name	Hex offset	Data type	Description
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. If the value of EXPLSITE_OFF is not 0, EXPLSITE is not used.
EXPLLUNM	3C	Character, 8 bytes	For SNA protocols, the locally known LU name of the requesting location. For TCP/IP protocols, 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, field reserved.
EXPLVIDS	55	Character, 1 byte	DB2 version identifier
EXPLSITE_OFF	56	Signed 2-byte integer	Offset from the beginning of the work area to the extended location name of the DB2 site that originated the work request. Use this value if the location name is greater than 16 bytes. The extended location name has the following format: • Signed, 2-byte integer: Length of the extended location name • Character, 128 bytes: Extended location name

Notes: 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.

PSPI

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.

Column boundaries for edit and validation procedures

DB2 stores columns contiguously, regardless of word boundaries in physical storage, except LOB and XML columns. LOB or XML values are not stored contiguously; an indicator column is stored in a base table in place of the LOB or XML values.

PSPI

You cannot specify edit procedures for any table that contains a LOB column. You cannot define edit procedures as WITH ROW ATTRIBUTES for any table that contains a ROWID column. In addition, LOB values are not available to validation procedures; indicator columns and ROWID columns represent LOB columns as input to a validation procedure.

Similarly, you cannot specify edit procedures as WITH ROW ATTRIBUTES for any table that contains an XML column. XML values are not available to validation

procedures. DOCID and XML indicator columns represent XML columns as input to a validation procedure.

PSPI

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.

PSPI

This byte is X'00' if the column value is not null; it is X'FF' if the value is null. This extra byte is included in the column length attribute (parameter FFMTFLEN).

PSPI

Fixed-length rows for edit and validation routines

If all columns in a table are of fixed length, the rows are stored in fixed-length format. The rows are byte strings.

PSPI

Example: The sample project activity table has five fixed-length columns. The first two columns do not allow nulls; the last three do.

Table 86. 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

PSPI

Varying-length rows for edit and validation routines

The rows of a table with varying-length columns are varying-length rows if they contain varying-length values. In basic row format, 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.

PSPI

The following table shows a row of the sample department table in basic row format. The first value in the DEPTNAME column indicates the column length as a hexadecimal value.

Table 87. A varying-length row in basic row format in the sample department table

DEPTNO	DEPTNAME		MGRNO		ADMRDEPT	LOCATION	
C01	0012	Information center	00	000030	A00	00	New York

Varying-length columns have no gaps after them. 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). A LOB indicator column is VARCHAR(4), and an XML indicator column is VARCHAR(6). It is stored in a base table in place of a LOB or XML column, and indicates whether the LOB or XML value for the column is null or zero length.

In reordered row format, if a table has any varying-length columns, all fixed length columns are placed at the beginning of the row, followed by the offsets to the varying length columns, followed by the values of the varying length columns.

The following table shows the same row of the sample department table, but in reordered row format. The value in the offset column indicates the offset value as a hexadecimal value.

Table 88. A varying-length row in reordered row format in the sample department table

DEPTNO	MGRNO		ADMRDEPT	LOCATION		Offset column	DEPTNAME
C01	00	000030	A00	00	New York	20	Information center

PSPI

Varying-length rows with nulls for edit and validation routines

A varying-length column can allow null values. In basic row format, the value in the length field includes the length of the null indicator byte but does not include the length field itself.

PSPI

The following table shows how the row would look in storage if nulls were allowed in DEPTNAME. The first value in the DEPTNAME column indicates the column length as a hexadecimal value.

Table 89. A varying-length row in basic row format in the sample department table

DEPTNO	DEPTNAME		MGRNO		ADMRDEPT	LOCATION	
C01	0013	Information center	00	000030	A00	00	New York

An empty string has a length of one, a X'00' null indicator, and no data following.

In reordered row format, if a table has any varying-length columns, with or without nulls, all fixed length columns are placed at the beginning of the row, followed by the offsets to the varying length columns, followed by the values of the varying length columns.

PSPI

EDITPROCs and VALIDPROCs for handling basic and reordered row formats

You can check the row format type (RFMTTYPE) to ensure that edit procedures (EDITPROC) and validation procedures (VALIDPROC) produce predictable results.

PSPI

If you write new edit and validation routines on tables with rows in basic row format (BRF) or reordered row format (RRF), make sure that EDITPROCs and VALIDPROCs are coded to check RFMTTYPE and handle both BRF and RRF formats.

If an EDITPROC or VALIDPROC handles only RRF, make sure that it checks RFMTTYPE and returns an error or warning if it detects BRF. If an EDITPROC or VALIDPROC that handles only BRF is to be used on tables in RRF, make sure that it checks RFMTTYPE and returns an error or warning if it detects RRF.

PSPI

Converting basic row format table spaces with edit and validation routines to reordered row format

You cannot convert the table spaces with edit and validation routines from basic row format to reordered row format directly. You must perform additional tasks to convert these table spaces.

Converting basic row format table spaces with edit routines to reordered row format

You can convert basic row format table spaces to reordered row format. If some tables in a table space have edit routines, you cannot convert the table space to reordered row format directly. You must take other actions for the conversion to succeed.

Procedure

PSPI To convert a table space to reordered row format, complete the following steps for each table that has an edit routine:

1. Use the UNLOAD utility to unload data from the table or tables that have edit routines.
2. Use the DROP statement to drop the table or tables that have edit routines.
3. Make any necessary modifications to the edit routines so that they can be used with rows in reordered row format.
4. Use the REORG utility to reorganize the table space. Using the REORG utility converts the table space to reordered row format.
5. Re-create tables with your modified edit routines. Also, re-create any additional related objects, such as indexes and check constraints.

6. Use the LOAD RESUME utility to load the data into the tables that have the modified edit routines. PSPI

Converting basic row format table spaces with validation routines to reordered row format

You can convert basic row format table spaces to reordered row format. If some tables in a table space have validation routines, you cannot convert the table space to reordered row format directly. You must take other actions for the conversion to succeed.

Procedure

PSPI To convert a table space to reordered row format, complete the following steps for each table that has a validation routine:

1. Use the ALTER TABLE statement to alter the validation routine to NULL.
2. Run the REORG utility or the LOAD REPLACE utility to convert the table space to reordered row format.
3. Make any necessary modifications to the validation routine so that it can be used with rows in reordered row format.
4. Use the ALTER TABLE statement to add the modified validation routine to the converted table. PSPI

Row format conversion for table spaces

The row format of a table space might be converted when you run the LOAD REPLACE or REORG TABLESPACE utilities.

If the DB2 subsystem parameter RRF is set to ENABLE, the table space is converted from basic row format to reordered row format when you run the LOAD REPLACE utility or the REORG TABLESPACE utility. (The default setting for the RRF subsystem parameter is ENABLE.) If the RRF subsystem parameter is set to DISABLE, the table space is not converted. Therefore, if the table space was in basic row format before running the LOAD REPLACE utility or the REORG TABLESPACE utility, the table space remains in basic row format. Likewise, if the table space was in reordered row format before running either of these utilities, the table space remains in reordered row format.

Exceptions:

- LOB table spaces and table spaces in the catalog and directory databases always remain in basic row format, regardless of the RRF subsystem parameter setting, or the setting of the ROWFORMAT keyword for the utility. (The ROWFORMAT keyword specifies the output row format in a table space or partition. This keyword overrides the existing RRF setting when specified.)
- XML table spaces always remain in reordered row format, regardless of the RRF subsystem parameter setting or the utility keyword setting.
- For universal table spaces that are cloned, both the base table space and the clone table space remain in the same format as when they were created, regardless of the RRF subsystem parameter setting or the utility keyword setting.
- When multiple data partitions are affected by the LOAD REPLACE utility or the REORG TABLESPACE utility, and some of the partitions are in basic row format and some are in reordered row format, the utilities convert every partition to reordered row format. This behavior is the default, regardless of the RRF subsystem parameter setting. Alternatively, you can specify ROWFORMAT BRF

in the utility statement for all affected partitions so that the table space is in basic row format after the utility completes successfully.

Example

To convert an existing table space from reordered row format to basic row format, run REORG TABLESPACE ROWFORMAT BRF against the table space. To keep the table space in basic row format on subsequent executions of the LOAD REPLACE utility or the REORG TABLESPACE utility, continue to specify ROWFORMAT BRF in the utility statement. Alternatively, you can set the RRF subsystem parameter to DISABLE.

Related reference:

REORDERED ROW FORMAT field (RRF subsystem parameter) (DB2 Installation and Migration)

REORG TABLESPACE (DB2 Utilities)

LOAD (DB2 Utilities)

Dates, times, and timestamps for edit and validation routines

The values in columns with data types of DATE, TIME, and TIMESTAMP are stored in specific formats.

PSPI

The DATE format that consists of 4 total bytes :

Table 90. DATE format

Year	Month	Day
2 bytes	1 byte	1 byte

The TIME format, which consists of 3 total bytes:

Table 91. TIME format

Hours	Minutes	Seconds
1 byte	1 byte	1 byte

The following table shows the TIMESTAMP format, which consists of 7 to 13 total bytes.

Table 92. TIMESTAMP format

Year	Month	Day	Hours	Minutes	Seconds	Partial second
2 bytes	1 byte	1 byte	1 byte	1 byte	1 byte	0 to 6 bytes

PSPI

Parameter list for row format descriptions

DB2 passes a description of the row format to an edit routine or a validation routine through a parameter list, which is generated by macro DSNDROW. The description includes both the general row characteristics and the characteristics of each column.

PSPI

DSNDROW defines the columns in the order as they are defined in the CREATE TABLE statement or possibly the ALTER TABLE statement. For rows in the reordered row format, the new column order in DSNDROW does not necessarily correspond to the order in which the columns are stored in the row. The following is the general row description:

Table 93. 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
RFMTTYPE	8	Character, 1 byte	Row type: X'00' = row with fixed-length columns X'04' = row with varying-length columns in basic row format X'08' = row with varying-length columns in reordered row format
	9	Character, 3 bytes	Reserved

The following is the description of each column:

Table 94. Description of a column format

Name	Hex offset	Data type	Description
FFMTFLEN	0	Signed fullword integer	Column length attribute
FFMTFTYP	4	Character, 1 byte	Data type code
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

The following is a description of data type codes and length attributes.

Table 95. Description of data type codes and length attributes

Data type	Code (FFMTFTYP)	Length attribute (FFMTFLEN)
BIGINT	X'32'	8
BINARY	X'34'	Length of string
VARBIN	X'38'	Length of string

Table 95. Description of data type codes and length attributes (continued)

Data type	Code (FFMTFTYP)	Length attribute (FFMTFLEN)
DECFLOAT	X'40'	8 for DECFLOAT(16) or 16 for DECFLOAT(34)
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'	7 + INTEGER((p+1) / 2), where p is the precision of the TIMESTAMP. The length can be 7 to 13.
ROWID	X'2C'	17
INDICATOR COLUMN	X'30'	4 for a LOB indicator column or 6 for an XML indicator column

PSPI

DB2 codes for numeric data in edit and validation routines

DB2 stores numeric data in a specially encoded format that is called *DB2-coded*.

PSPI

To retrieve numeric data in its original form, you must DB2-decode it according to its data type:

Table 96. 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)

Table 96. DB2 decoding procedure according to data type (continued)

Data type	DB2 decoding procedure
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)
BIGINT	Invert the sign bit (high order bit). **Value** **Meaning** **8000000000000854** 0000000000000854 (2132 decimal) **7FFFFFFFFFFFFFE0** FFFFFFFFFFFFFFE0 (-32 decimal)
DECFLOAT	Convert and return a DECFLOAT representation of a number or string representation of a number. **Value** **Meaning** **D8F77D00000000000C** 222C000000001E80 (+7.500 decfloat) **270882FFFFFFFFFFF2** A2300000000003D0 (-7.50 decfloat)

PSPI

Appendix B. Stored procedures for administration

DB2 provides stored procedures that you can call in your application programs to perform administrative functions.

For the complete list of stored procedures that are provided with DB2, see Procedures that are supplied with DB2 (DB2 SQL).

Related concepts:

Migration step 25: Configure DB2 for running stored procedures and user-defined functions (optional) (DB2 Installation and Migration)

Supplied stored procedures for utility operations (DB2 Utilities)

DB2-supplied stored procedures for application programming (DB2 Application programming and SQL)

Related tasks:

Migration step 26: Set up DB2-supplied routines (DB2 Installation and Migration)

Installing DB2-supplied routines during installation (DB2 Installation and Migration)

Common SQL API stored procedures

Common SQL API stored procedures implement a cross-database and cross-operating system SQL API that is portable across IBM data servers, including DB2 for Linux, UNIX, and Windows.

The *Common SQL API* is a solution-level API that supports common tooling across IBM data servers. This Common SQL API ensures that tooling does not break when a data server is upgraded, and it notifies the caller when an upgrade to tooling is available to capitalize on new data server functionality. Applications that support more than one IBM data server will benefit from using the Common SQL API, as it lowers the complexity of implementation. Such applications typically perform a variety of common administrative functions. For example, you can use these stored procedures to retrieve data server configuration information, return system information about the data server, and return the short message text for an SQLCODE.

The Common SQL API includes the following stored procedures that are supplied with DB2:

- GET_CONFIG stored procedure (DB2 SQL)
- GET_MESSAGE stored procedure (DB2 SQL)
- GET_SYSTEM_INFO stored procedure (DB2 SQL)
- SET_PLAN_HINT stored procedure (DB2 SQL)

These stored procedures use version-stable XML documents as parameters. These XML parameter documents adhere to a single, common document type definition (DTD). This DTD is flexible enough to represent hierarchical structures and binary data. The XML parameter documents can be parsed by using the Apache Commons Configuration component.

© Copyright IBM Corp. 1982, 2016

Each of the three XML parameter documents has a name and a version, and is typically associated with one stored procedure. The three types of XML parameter documents are:

- XML input documents
- XML output documents
- XML message documents

The XML input document is passed as input to the stored procedure. The XML output document is returned as output, and the XML message document returns messages. If the structure, attributes, or types in an XML parameter document change, the version of the XML parameter document changes. The version of all three of these documents remains in sync when you call a stored procedure. For example, if you call the GET_SYSTEM_INFO stored procedure and specify the *major_version* parameter as 1 and the *minor_version* parameter as 1, the XML input, XML output, and XML message documents will be Version 1.1 documents.

Related reference:

Procedures that are supplied with DB2 (DB2 SQL)

Related information:

Apache Commons Configuration component

Common DTD

DB2 for z/OS Stored Procedures: Through the CALL and Beyond (IBM Redbooks)

Versioning of XML documents

Common SQL API stored procedures support multiple versions of the three XML parameter documents: XML input documents, XML output documents, and XML message documents.

If the structure, attributes, or types in an XML parameter document change, the version of the XML parameter document changes. Therefore, the content of an XML parameter document varies depending on the version that you specify.

The version of all three of these documents remains in sync when you call a stored procedure. For example, if you call the GET_SYSTEM_INFO stored procedure and specify the *major_version* parameter as 1 and the *minor_version* parameter as 1, the XML input, XML output, and XML message documents will be Version 1.1 documents.

Version information in an XML parameter document is expressed as key and value pairs for **Document Type Major Version** and **Document Type Minor Version**. For example, an XML output document might define the following keys and values in a dictionary element:

```
<key>Document Type Name</key><string>Data Server Configuration Output</string>
<key>Document Type Major Version</key><integer>2</integer>
<key>Document Type Minor Version</key><integer>0</integer>
```

To determine the highest supported document version for a stored procedure, specify NULL for the *major_version* parameter, the *minor_version* parameter, and all other required parameters. The stored procedure returns the highest supported document version as values in the *major_version* and *minor_version* output parameters, and sets the *xml_output* and *xml_message* output parameters to NULL.

If you specify non-null values for the *major_version* and *minor_version* parameters, you must specify a document version that is supported . If the version is invalid, the stored procedure returns an error (-20457).

If the XML input document in the *xml_input* parameter specifies the **Document Type Major Version** and **Document Type Minor Version** keys, the value for those keys must be equal to the values that you specified in the *major_version* and *minor_version* parameters, or an error (+20458) is raised.

XML input documents

The XML input document is passed as input to common SQL API stored procedures and adheres to a single, common document type definition (DTD).

The XML input document consists of a set of entries that are common to all stored procedures, and a set of entries that are specific to each stored procedure. The XML input document has the following general structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
   <key>Document Type Name</key><string>Data Server Message Input</string>
   <key>Document Type Major Version</key><integer>1</integer>
   <key>Document Type Minor Version</key><integer>0</integer>
   <key>Document Locale</key><string>en_US</string>
   <key>Complete</key><false/>
      <!-- Document type specific data appears here. -->
</dict>
</plist>
```

The **Document Type Name** key varies depending on the stored procedure. This example shows an XML input document for the GET_MESSAGE stored procedure. In addition, the values of the **Document Type Major Version** and **Document Type Minor Version** keys depend on the values that you specified in the *major_version* and *minor_version* parameters for the stored procedure.

If the stored procedure is not running in Complete mode, you must specify the **Document Type Name** key, the required parameters, and any optional parameters that you want to specify. Specifying the **Document Type Major Version** and **Document Type Minor Version** keys are optional. If you specify the **Document Type Major Version** and **Document Type Minor Version** keys, the values must be the same as the values that you specified in the *major_version* and *minor_version* parameters. You must either specify both or omit both of the **Document Type Major Version** and **Document Type Minor Version** keys. Specifying the **Document Locale** key is optional. If you specify the **Document Locale** key, the value is ignored.

Important: XML input documents must be encoded in UTF-8 and contain only English characters.

Complete mode for returning valid XML input documents

You can use *Complete mode* to create a valid XML input document for the common SQL API stored procedures. Then, you can customize the XML input document and pass it back to the procedure.

If the **Complete** key is included and you set the value to `true`, the stored procedure will run in Complete mode, and all other entries in the XML input document will be ignored. The following example shows the minimal XML input document that is required for the stored procedure to run in Complete mode:

```
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
   <key>Complete</key><true/>
</dict>
</plist>
```

If the stored procedure runs in Complete mode, a complete input document is returned by the *xml_output* parameter of the stored procedure. The returned XML document is a full XML input document that includes a **Document Type** and sections for all possible required and optional parameters. The returned XML input document also includes entries for **Display Name**, **Hint**, and the **Document Locale**. Although these entries are not required (and will be ignored) in the XML input document, they are usually needed when rendering the document in a client application.

All entries in the returned XML input document can be rendered and changed in ways that are independent of the operating system or data server. Subsequently, the modified XML input document can be passed in the *xml_input* parameter in a new call to the same stored procedure. This enables you to programmatically create valid *xml_input* documents.

XML output documents

The XML output documents that are returned as output from common SQL API stored procedures share a common set of entries.

At a minimum, the XML output documents that are returned in the *xml_output* parameter include the following key and value pairs, followed by information that is specific to each stored procedure:

```
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
   <key>Document Type Name</key>
   <string>Data Server Configuration Output</string>
   <key>Document Type Major Version</key><integer>1</integer>
   <key>Document Type Minor Version</key><integer>0</integer>
   <key>Data Server Product Name</key><string>DSN</string>
   <key>Data Server Product Version</key><string>9.1.5</string>
   <key>Data Server Major Version</key><integer>9</integer>
   <key>Data Server Minor Version</key><integer>1</integer>
   <key>Data Server Platform</key><string>z/OS</string>
   <key>Document Locale</key><string>en_US</string>
    <!-- Document type specific data appears here. -->
</dict>
</plist>
```

The **Document Type Name** key varies depending on the stored procedure. This example shows an XML output document for the GET_CONFIG stored procedure. In addition, the values of the **Document Type Major Version** and **Document Type Minor Version** keys depend on the values that you specified in the *major_version* and *minor_version* parameters for the stored procedure.

Entries in the XML output document are grouped by using nested dictionaries. Each entry in the XML output document describes a single piece of information. In general, an XML output document is comprised of **Display Name**, **Value**, and **Hint**, as shown in the following example:

```
<key>SQL Domain</key>
<dict>
   <key>Display Name</key>
```

```
   <string>SQL Domain</string>
   <key>Value</key>
   <string>v33ec059.svl.ibm.com</string>
   <key>Hint</key>
   <string />
</dict>
```

XML output documents are generated in UTF-8 and contain only English characters.

XPath expressions for filtering output

You can use an XPath expression to filter the XML output that is returned by the GET_CONFIG, GET_MESSAGE, and GET_SYSTEM_INFO stored procedures.

To filter the output, specify a valid XPath query string in the *xml_filter* parameter of the stored procedure.

The following restrictions apply to the XPath expression that you specify:

- The XPath expression must reference a single value.
- The XPath expression must always be absolute from the root node. For example, the following path expressions are allowed: `/`, *`nodename`*, `.`, and `..`. The following expressions are not allowed: `//` and `@`
- The only predicates allowed are [*`path`*`='`*`value`*`'`] and [*`n`*].
- The only axis allowed is `following-sibling`.
- The XPath expression must end with one of the following, and, if necessary, be appended with the predicate [1]: `following-sibling::string`, `following-sibling:: data`, `following-sibling::date`, `following-sibling::real`, or `following-sibling::integer`.
- Unless the axis is found at the end of the XPath expression, it must be followed by a `::dict`, `::string`, `::data`, `::date`, `::real`, or `::integer`, and if necessary, be appended with the predicate [1].
- The only supported XPath operator is `=`.
- The XPath expression cannot contain a function, namespace, processing instruction, or comment.

Tip: If the stored procedure operates in complete mode, do not apply filtering, or an SQLCODE (+20458) is raised.

Example: The following XPath expression selects the value for the Data Server Product Version key from an XML output document:

```
/plist/dict/key[.='Data Server Product Version']/following-sibling::string[1]
```

The stored procedure returns the string `9.1.5` in the *xml_output* parameter if the value of the Data Server Product Version is 9.1.5. Therefore, the stored procedure call returns a single value rather than an XML document.

XML message documents

An XML message document provides detailed information about an SQL warning condition.

When a common SQL API stored procedure encounters an internal processing error or invalid parameter, the data server returns an SQLCODE and the

corresponding SQL message to the caller. When this occurs, the procedure returns an XML message document in the *xml_message* parameter that contains additional information about the warning.

An XML message document contains key and value pairs followed by details about an SQL warning condition. The general structure of an XML message document is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
   <key>Document Type Name</key>
   <string>Data Server Message</string>
   <key>Document Type Major Version</key><integer>1</integer>
   <key>Document Type Minor Version</key><integer>0</integer>
   <key>Data Server Product Name</key><string>DSN</string>
   <key>Data Server Product Version</key><string>9.1.5</string>
   <key>Data Server Major Version</key><integer>9</integer>
   <key>Data Server Minor Version</key><integer>1</integer>
   <key>Data Server Platform</key><string>z/OS</string>
   <key>Document Locale</key><string>en_US</string>
    ---  Details about an SQL warning condition are included here. ---
 </dict>
</plist>
```

The details about an SQL warning will be encapsulated in a dictionary entry, which is comprised of **Display Name**, **Value**, and **Hint**, as shown in the following example:

```
<key>Short Message Text</key>
<dict>
   <key>Display Name</key><string>Short Message Text</string>
   <key>Value</key>
   <string>DSNA630I DSNADMGC A PARAMETER FORMAT OR CONTENT ERROR WAS FOUND.
           The XML input document must be empty or NULL.</string>
   <key>Hint</key><string />
</dict>
```

XML message documents are generated in UTF-8 and contain only English characters.

Troubleshooting DB2 stored procedure problems

If you encounter problems setting up, calling, or running stored procedures, several troubleshooting techniques and tools are available in DB2 and z/OS.

Procedure

To troubleshoot DB2 stored procedures, perform one or more of the following actions:

- For general information about the available debugging tools and techniques, see Debugging stored procedures (DB2 Application Programming and SQL Guide).
- If you encounter problems when implementing RRS, see RRS error samples (DB2 9 for z/OS Stored Procedures: Through the CALL and Beyond).
- If you have problems calling a particular stored procedure, you might not have the required authorizations. See Privileges to execute a stored procedure called statically (DB2 9 for z/OS Stored Procedures: Through the CALL and Beyond).
- If you are troubleshooting Java stored procedures, see Java stored procedure common problems (DB2 9 for z/OS Stored Procedures: Through the CALL and Beyond).

- If your invoking program receives SQLCODE -430, see Classical debugging of stored procedures (DB2 9 for z/OS Stored Procedures: Through the CALL and Beyond).

Information resources for DB2 for z/OS and related products

Information about DB2 for z/OS and products that you might use in conjunction with DB2 for z/OS is available online in IBM Knowledge Center or on library websites.

Obtaining DB2 for z/OS publications

The current DB2 10 for z/OS DB2 for z/OS publications are available from the following website:

http://www-01.ibm.com/support/docview.wss?uid=swg27019288

Links to IBM Knowledge Center and the PDF version of each publication are provided.

DB2 for z/OS publications are also available for download from the IBM Publications Center (http://www.ibm.com/shop/publications/order).

In addition, books for DB2 for z/OS are available on a CD-ROM that is included with your product shipment:

- DB2 10 for z/OS Licensed Library Collection, LK5T-7390, in English. The CD-ROM contains the collection of books for DB2 10 for z/OS in PDF format. Periodically, IBM refreshes the books on subsequent editions of this CD-ROM.

Installable information center

You can download or order an installable version of the Information Management Software for z/OS Solutions Information Center, which includes information about DB2 for z/OS, QMF, IMS, and many DB2 and IMS Tools products. You can install this information center on a local system or on an intranet server. For more information, see http://www-01.ibm.com/support/knowledgecenter/SSEPEK_11.0.0/com.ibm.db2z11.doc/src/alltoc/installabledzic.html.

© Copyright IBM Corp. 1982, 2016

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

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, MD-NC119
Armonk, NY 10504-1785 US

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing Legal and Intellectual Property Law IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

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 jurisdictions 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 websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites 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, 2016

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 Director of Licensing IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785 US

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.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

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 actual people or business enterprises 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. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as shown below:

© (*your company name*) (*year*).
Portions of this code are derived from IBM Corp. Sample Programs.
© Copyright IBM Corp. (*enter the year or years*).

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

This information is intended to help you to plan for and administer DB2 10 for z/OS. This information also documents General-use Programming Interface and Associated Guidance Information and Product-sensitive Programming Interface and Associated Guidance Information provided by DB2 10 for z/OS.

General-use Programming Interface and Associated Guidance Information

General-use Programming Interfaces allow the customer to write programs that obtain the services of DB2 10 for z/OS.

General-use Programming Interface and Associated Guidance Information is identified where it occurs by the following markings:

GUPI

General-use Programming Interface and Associated Guidance Information...

GUPI

Product-sensitive Programming Interface and Associated Guidance Information

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 by the following markings:

PSPI

Product-sensitive Programming Interface and Associated Guidance Information...

PSPI

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered marks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at: http://www.ibm.com/legal/copytrade.shtml.

Linux is a registered trademark of Linus Torvalds 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.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions:

Applicability: These terms and conditions are in addition to any terms of use for the IBM website.

Personal use: You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use: You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights: Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Privacy policy considerations

IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies

and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at http://www.ibm.com/privacy and IBM's Online Privacy Statement at http://www.ibm.com/privacy/details the section entitled "Cookies, Web Beacons and Other Technologies" and the "IBM Software Products and Software-as-a-Service Privacy Statement" at http://www.ibm.com/software/info/product-privacy.

Glossary

The glossary is available in IBM Knowledge Center.

See the Glossary topic for definitions of DB2 for z/OS terms.

© Copyright IBM Corp. 1982, 2016

Index

A

abend
  AEY9 480
  ASP7 480
  backward log recovery 520
  CICS 294
    abnormal termination 480
    scenario 485
    waits 480
  CICS transactions 295
  current status rebuild 504
  disconnecting DB2 306
  DXR122E 471
  effects 364
  forward log recovery 514
  IMS
    procedure 476
    scenario 479
    U3047 479
    U3051 479
  IRLM
    scenario 471
    STOP command 276
    STOP DB2 command 276
  log
    damage 500
    lost information 525
  log initialization phase 503
  page problems 524
  Resource Recovery Services (RRS) 308
  restarting 214, 502
  SQLCODE -923 486
  VVDS (VSAM volume data set)
    destroyed 542
    out of space 542
access method services
  ALTER command 543
  bootstrap data set (BSDS) 360
  commands
    ALTER 35
    ALTER ADDVOLUMES 26, 35
    ALTER REMOVEVOLUMES 26
    DEFINE 35, 524
    DEFINE CLUSTER 35, 36
    EXPORT 443
    IMPORT 193, 524
    PRINT 455
    REPRO 455, 497
  damaged bootstrap data set (BSDS)
    deleting 495
    renaming 495
  damaged BSDS (bootstrap data set)
    deleting 495
    renaming 495
  data sets
    managing 35
  DEFINE CLUSTER command
    defining extents 36
    examples 36
    LINEAR option 36
    REUSE option 36
access method services *(continued)*
  DEFINE CLUSTER command *(continued)*
    SHAREOPTIONS 36
  DEFINE command 446
  table spaces
    re-creating 524
  to rename DB2 data sets 397
  work files
    redefining 446
access paths
  hash access 47, 150, 152
accessibility
  keyboard xiv
  shortcut keys xiv
active log data sets
  copying
    IDCAMS REPRO statement 362
  high-level qualifier
    changing 185, 186, 187, 188, 189
  offloading 339
  stopping
    effects 490
active log inventory
  data sets
    adding 349
active logs
  dual logging 340
  gaps
    creating 527
  offloading 340
  out-of-space conditions 488
  recovering 488
  troubleshooting 488
  truncation 341
  VSAM data sets 605
  writing 340
ADMIN_TASK_ADD stored procedure
  SQL codes 237
ADMIN_TASK_OUTPUT function 231
ADMIN_TASK_REMOVE stored procedure
  SQL codes 237
administration
  stored procedures 677
administrative task schedulers
  ADMIN_TASK_ADD stored procedure 237
  ADMIN_TASK_CANCEL stored procedure 232
  ADMIN_TASK_OUTPUT function 231
  ADMIN_TASK_REMOVE stored procedure 237
  ADMIN_TASK_STATUS function
    SQL codes 237
  ADMIN_TASK_UPDATE stored procedure 232
  architecture 238
  data sharing environment
    architecture 242
    specifying 227
    synchronization 234
    task execution 251
  interface
    security 245
  JCL jobs 251
  lifecycle 239

© Copyright IBM Corp. 1982, 2016

administrative task schedulers *(continued)*
overview 223
resources
security 246
security 243
starting 233
stopping 234
stored procedures
accounting information 242
calling 249, 250
displaying results 231
SQL codes 237
SYSIBM.ADMIN_TASKS table 241
SYSIBM.ADMIN_TASKS_HIST table 241
task execution 247
multi-thread 247
security 246
task lists 241
recovering 236
task status
listing 229, 230
tasks 223
adding 223
listing 229
removing 232
sample schedule 225
scheduling 224
stopping 232
updating 232
time zones 252
tracing
disabling 235
enabling 235
troubleshooting 235, 236
Unicode restrictions 250
user roles 244
user-defined table functions
troubleshooting 237
administrative tasks
scheduling 223
alias
retrieving catalog information about 103
ALTER BUFFERPOOL command 266
ALTER command
access method services 543
FOR option 35
TO option 35
ALTER DATABASE statement 113
options 113
ALTER FUNCTION statement 183
ALTER INDEX statement 73
ALTER PROCEDURE statement 181
ALTER STOGROUP statement 114
ADD VOLUMES clause 115
ALTER TABLE statement 125
DATA CAPTURE clause 159
default column values 128
VALIDPROC clause 158
ALTER TABLESPACE statement 117
application changes
backing out
with quiesce point 474
application defaults module 213
application environment
status 323
application errors
backing out
without a quiesce point 475
application period 53
adding 154
application plans
dependent objects 163
application programs
call attachment facility (CAF)
running 221
coding SQL statements
for IMS 218
errors 474
information
obtaining 259
issuing commands 208
recovery procedures
CICS 480
IMS 479
RRSAF (Resource Recovery Services attachment facility)
running 221
running
batch 219
CICS transactions 219
error recovery 474
IMS 218
TSO
running 217
application-period temporal tables 53
creating 57, 154
querying 60
applications
disconnecting 294, 296
archive log
retaining 397
ARCHIVE LOG command 348
archive log data sets
archiving
DFSMS (Data Facility Storage Management Subsystem) 344
BSDS (bootstrap data set) 360
deleting 345
dual logging 343
dynamic allocation 343
high-level qualified
changing 189
high-level qualifier
changing 185, 186, 187, 188, 189
locating 357
multivolumes 344
offloading 339
overview 343
recovering 492
retention period 345
types 343
writing 340
archive logs
tapes
setting limits 351
archiving
disks 344
tapes 344
attributes
data types 8
names 7
values 8

authorities
  CICS
    controlling access 219
  IMS application programs
    controlling access 218
authority
  retrieving catalog information 106
authorization access
  finding 218
AUTOESTSPACE(YES) option
  REORG TABLESPACE utility 152
availability
  recovering
    data sets 417
    page sets 417

B

back-level image copy data sets 397
BACKUP SYSTEM utility 33, 34, 583
  DB2 subsystem
    recovering 467
backups
  data
    moving 435
  data sets
    using DFSMShsm 404
  databases
    DSN1COPY 466
    image copies 418
backward log recovery phase
  failures 520
    recovering 520
  restarting 368
base table
  distinctions from temporary tables 50
base tables
  creating 46
basic direct access method (BDAM) 343
basic row format 671
basic sequential access method (BSAM) 343
batch message processing (BMP) program 303
batch processing
  TSO 218
BDAM (basic direct access method) 343
bitemporal tables 53
  creating 58
  system-period data versioning 58
blank
  column with a field procedure 651
BMP (batch message processing) program
  dependent regions
    connecting 303
bootstrap data set (BSDS)
  high-level qualifier
    changing 187, 189
BSAM (basic sequential access method)
  archive log data sets
    reading 343
BSDS (bootstrap data set)
  archive log information 360
  defining 361
  dual copies 360
  dual recovery 497
  dual-BSDS mode 361
  failure symptoms 500
BSDS (bootstrap data set) *(continued)*
  high-level qualifier
    changing 185, 186, 188, 189
  log inventory
    changing 362
  managing 360
  recovery procedures 495
  recovery scenarios 523
  restart usage 364
  restoring
    from archive logs 497
  single recovery 497
  stand-alone log services 619
buffer pools
  attributes
    altering 266
  controlling 253
  logging 339
  monitoring 267

C

CAF connections
  monitoring 287
call attachment facility (CAF)
  application programs
    running 221
CANCEL THREAD command 319
  CICS threads 294
  disconnecting from TSO 289
  effects 320
catalog alias
  defining 185
catalog definitions
  consistency 464
catalog tables
  image copies
    frequency 402, 403
  retrieving information about
    primary keys 107
    status 108
  SYSAUXRELS 110
  SYSCOLUMNS 162
    updated by COMMENT ON statement 112
    updated by CREATE VIEW statement 106
  SYSCOPY
    discarding records 410
    image copies 600
    image copy information 408
    RECOVER utility 401
  SYSFOREIGNKEYS 107
  SYSINDEXES
    dropping tables 163
  SYSINDEXPART
    space allocation information 39
  SYSPLANDEP 163
  SYSRELS
    describes referential constraints 107
  SYSROUTINES 110
  SYSSEQUENCES 112
  SYSSTOGROUP 24
    sample query 102
  SYSSYNONYMS 162
  SYSTABAUTH
    dropping tables 163
    table authorizations 106
    updated by CREATE VIEW statement 106

catalog tables *(continued)*
SYSTABAUTH *(continued)*
view authorizations 106
SYSTABLEAPART
partition order 103
SYSTABLEPART 115
SYSTABLES
rows maintained 102
updated by COMMENT ON statement 112
updated by CREATE VIEW statement 106
SYSTRIGGERS 111
SYSVIEWDEP
view dependencies 163
SYSVOLUMES 24
catalog tables, DB2
image copy 397
catalog, DB2
constraint information 109
database design 101, 113
retrieving information from 101
catalogs
DB2
DSNDB06 database 408
recovery procedures 541
point-in-time recovery 447
recovering 447
CDB (communications database)
backing up 399
high-level qualifier
changing 190
CHANGE command
IMS
purging residual recovery entries 296
change log inventory utility
bootstrap data set (BSDS) 273
BSDS (bootstrap data set)
changing 362
change number of sessions (CNOS) 552
CHANGE SUBSYS command
IMS 302
check constraints
adding 139
dropping 139
check pending status
retrieving catalog information 108
checkpoint
queue 375
checkpoint frequency
changing 350
checkpoints
log records 599, 604
CICS
applications
disconnecting 294
commands
accessing databases 290
connecting 291
connecting to DB2
authorization IDs 219
connections
controlling 290, 296
disconnecting from DB2 295
DSNC command 209
DSNC DISCONNECT command 294
dynamic plan selection
exit routine 664
CICS *(continued)*
environment
planning 219
facilities 664
diagnostic traces 334
indoubt units of recovery 291
operating
outstanding indoubt units 388
terminates AEY9 486
postponed units of recovery
displaying 294
programming
applications 219
recovery procedures 479
application failures 480
attachment facility failures 485
CICS not operational 480
DB2 connection failures 481
indoubt units of recovery 482
restarting 291
threads
connecting 292
two-phase commit 379
CICS commands
DSNC DISCONNECT 290
DSNC DISPLAY 290
DSNC DISPLAY PLAN 292
DSNC DISPLAY TRANSACTION 292
DSNC MODIFY
ERRDEST option 290
DSNC STOP 290
DSNC STRT 290, 292
example 291
responses 209
CICS terminal
issuing commands 207
CLONE keyword 468
clone tables
backup and recovery 468
creating 66
CNOS (change number of sessions)
failures 552
coding
exit routines
general rules 665
parameters 667
cold start
bypassing the damaged log 500
recovery procedures 375
special situations 525
column
retrieving
catalog information 104
comments 112
column boundaries 668
columns
adding 126
data types
altering 128
default values
altering 128
dropping 161
XML
adding 149
commands
DISPLAY BUFFERPOOL 267
issuing from application programs 208

commands *(continued)*
issuing from CICS terminals 207
issuing from IMS terminals 207
issuing from TSO consoles 205
issuing from z/OS console 205
prefix 205
START DB2 205
START FUNCTION SPECIFIC 269
COMMENT ON statement
examples 112
storing 112
commit
two-phase process 379
common SQL API
XML message documents 681
communications failure
scenarios 586
Complete mode
common SQL API 679
conditional restart
control record
backward log recovery failures 521
current status rebuild failures 512
forward log recovery failures 519
log initialization failures 512
wrap-around queue 375
excessive loss of active log data 527
multiple systems 386
overview 371
performing 374
total loss of log 526
conditional restarts
system-level backups 374
connections
CICS 290
controlling 287
controlling IMS 296
DB2 290
displaying
IMS activity 304
IDs
identifying a unit of recovery 474
message DSNR007I 366
outstanding unit of recovery 366
used by IMS 218
information
displaying 316
lost
on restart 387
monitoring 253, 304, 315
consistency groups 580
continuous operation
data sets
recovering 417
table spaces
recovering 417
control interval (CI) 339, 343, 606
sizing 23
control intervals (CI)
reading 619
control region
IMS 303
conversion procedures
expected output 647
invoking 646
overview 645
parameter list 646
conversion procedures *(continued)*
specifying 645
coordinator
multi-site updates 381
copy pools 34
SMS construct 33
COPY utility 454
backing up 466
data
copying 418
restoring 466
DFSMSdss concurrent copy 421, 441
copying
DB2 subsystems 195
relational databases 195
correlation IDs 293, 309
CICS 482
duplicates 298, 482
IMS 298
outstanding unit of recovery 366
RECOVER INDOUBT command 301
TSO connections 287
CREATE AUXILIARY TABLE statement 73
CREATE DATABASE statement 22
CREATE FUNCTION statement 86
CREATE GLOBAL TEMPORARY TABLE statement
distinctions from base tables 50
CREATE INDEX statement 73
DEFINE NO clause 73
USING clause 39
CREATE PROCEDURE statement 84
CREATE STOGROUP statement 24
VOLUMES('*') attribute 25, 31
CREATE TABLE statement 42, 82
examples 46
CREATE TABLESPACE statement
creating explicitly 42
DEFINE NO clause 25, 42
DSSIZE option 40
USING STOGROUP clause 25, 42
created temporary table
distinctions from base tables 50
created temporary tables 49
creating 49
CRESTART control statement
indoubt units of recovery
resolving 394
cron format
UNIX 227
current status rebuild
failure recovery 502
phase of restart 366

D

damaged data
renaming data sets 397
data
access control
START DB2 command 213
backing up 466
distributed
controlling connections 312
exchanging 68
inconsistencies
resolving 530
loading 78

data *(continued)*
a single row 82
multiple rows 82
modeling 4
moving 195
recovering 357
restoring 466, 554
point-in-time recovery 424
data availability
maximizing 405
data classes
assigning 40
SMS construct 40
data compression
log records 599
logging 339
data consistency
maintaining 379
point-in-time recovery 429
Data Facility Product (DFSMSdfp) 193
Data Facility Storage Management Subsystem (DFSMS)
concurrent copy 421
copying data 421
recovery 421
data management
automatic 404
data mirroring 578
recovery 578, 581
data pages
changes
control information 602
data 602
pointers 602
data set
damaged 397
Data Set Services (DFSMSdss) 193
data sets
adding 543, 546
backing up
using DFSMS 421
copying 418
DB2-managed
extending 26, 28
extension failures 26
nonpartitioned spaces 26
partitioned spaces 26
primary space allocation 27, 28, 30
recovering 458
secondary space allocation 27, 28, 30
deferring allocation 25
extending 543
high-level qualifier
changing 185
managing 22, 35
access method services 35
DFSMShsm 31
using DFSMShsm 31
with DB2 storage groups 23
migrating 116
moving 196
with utilities 197
without utilities 197
naming 38
partitioned partitioning indexes 36
partitioned secondary indexes 36
partitioned table spaces 36
recovering 460
data sets *(continued)*
using non-DB2 dump 455
renaming 413
user-managed 35
defining 36
space allocation 120
VSAM 605
data sharing environment
RECOVER TOLOGPOINT option 430
restarting 370
data sharing members
read requests 622
data types
altering 161
implications 130
specifying 8
subtypes
altering 160
database
designing
using catalog 101
implementing a design 113
database design 3
hash access 19
implementing 21
logical 3
physical 15
database exception table (DBET)
log records 605
exception states 600
image copies 600
database management systems (DBMSs)
monitoring 285
database objects 3
databases
access
controlling 312
access threads
failures 550
security failure 553
altering 101, 113
backing up
copying data 418
backups
planning 395
changes
rollbacks 337
units of recovery 337
controlling 253
copying 195
creating 22
designing 3
logical data modeling 3
physical data modeling 3
DL/I
loading data 84
dropping 22
implementing 21
monitoring 256
page set control records 605
recovering 417
failure scenarios 535
planning 395
RECOVER TOCOPY 428
RECOVER TOLOGPOINT 428
RECOVER TORBA 428
starting 254

databases *(continued)*
status information 256
stopping 264, 265
work file databases
DSNDB07 418
DataRefresher 84
date routines
expected output 644
invoking 642
overview 641
parameter list 643
specifying 642
DB2
user databases, image copy 397
DB2 catalog
DSNDB06 database 408
high-level qualifier
changing 190
recovery procedure 541
DB2 commands
commands
RECOVER INDOUBT 393
RESET INDOUBT 393
responses 209
START DATABASE 254
START DB2 212
STOP DATABASE 254
STOP DDF 332
DB2 DataPropagator
tables
altering 159
DB2 restart recovery 463
DB2 storage groups
high-level qualifier
changing 192
DB2 subsystem
abend
restarting 214
copying 196
operating 203
recovering 462, 463, 464, 468
BACKUP SYSTEM utility 467
RESTORE SYSTEM utility 467, 469
recovery 599
restarting 363, 519
log truncation 506
resolving inconsistencies 513
restoring 462
starting 211, 515
startup
application defaults module 213
stopping 211, 214, 363, 371
termination scenario 486
wait status 213
DB2-defined extents 124
DB2-managed data sets
enlarging 546
recovering 458
DB2-managed objects
changing 192
DB2I (DB2 Interactive) 217
TSO connections 205
DBD01 directory table space
quiescing 442
recovery information 408
DDF
stopping 333
DDF (distributed data facility)
alerts 331
failures
recovering 548
DDL registration tables
recovery preparation 399
DECLARE GLOBAL TEMPORARY TABLE statement
distinctions from base tables 50
declared temporary table
distinctions from base tables 50
declared temporary tables 49
creating 50
default database
(DSNDB04) 190
high-level qualifier
changing 190
DEFINE command
access method services
FOR option 35
re-creating table spaces 524
TO option 35
DELETE CLUSTER command 39
DELETE command
access method services 524
deleting
archive log data sets 345
denormalizing tables 16
dependent regions
disconnecting 304
DFSLI000 (IMS language interface module) 218
DFSMS (Data Facility Storage Management Subsystem)
archive log data sets 344
DFSMSdfp (Data Facility Product) 193
DFSMSdss (Data Set Services) 193
DFSMSdss RESTORE command
RECOVER utility 33
DFSMShsm
data classes
assigning indexes 31
assigning table spaces 31
data sets
migrating 31
DFSMShsm (Data Facility Hierarchical Storage Manager)
advantages 31
archive logs
recalling 32
BACKUP SYSTEM utility 33
backups 404
data sets
moving 193
FRBACKUP PREPARE command 583
recovery 404
DFSMShsm (Hierarchical Storage Manager)
HMIGRATE command 193
HRECALL command 193
DFSMSsms (DFSMS storage management subsystem)
BACKUP SYSTEM utility 34
diagnostic information
obtaining 326
directory
high-level qualifier
changing 190
image copies
frequency 402, 403
order of recovery
I/O errors 541
point-in-time recovery 447

directory *(continued)*
recovering 447
SYSLGRNX table
discarding records 410
records log RBA ranges 408
directory, DB2
image copy 397
disability xiv
disaster recovery
archive logs 554, 560
data mirroring 578
essential elements 444
image copies 554, 560
preparation 412
remote site recovery 467
rolling disaster 578
scenarios 553
system-level backups 554
tracker sites 569
disk dump and restore
considerations 417
disk storage
estimating 87
space requirements 87
disks
archiving 344
requirements 88
storage
estimating 88
storage group assignments
altering 114
DISPLAY BUFFERPOOL command 267
DISPLAY command
IMS
SUBSYS option 296
DISPLAY DATABASE command 253, 256
LPL option 261
SPACENAM option 259
DISPLAY DDF command 314
DISPLAY FUNCTION SPECIFIC command 268, 269
DISPLAY LOCATION command 316
DISPLAY LOG command 352
DISPLAY OASN command
IMS 476
displaying RREs 302
DISPLAY PROCEDURE command
examples 321
DISPLAY THREAD command 284
DETAIL keyword 285
DETAIL option
controlling DDF connections 282
examples 322
IMS threads
displaying 299
showing 304
LOCATION option 280, 281, 282
LUWID keyword 283
output 279
TYPE (INDOUBT) option 482
DISPLAY UTILITY command
log records 599
DISPLAY WLM command 323
distributed data
recovering 396
distributed data facility (DDF)
information
displaying 314
distributed data facility (DDF) *(continued)*
resuming 313
server activity
resuming 313
starting 312
stopping 332
FORCE option 332
QUIESCE option 332
suspending 313
distributed environment
restart conditions 530
restarting
DB2 subsystem 530
DL/I databases
data
loading 84
down-level detection
controlling 537
DSNTIPN panel 537
down-level page sets
recovering 537
DROP statement 22, 72, 85, 87
TABLESPACE option 120
DROP TABLE statement 162
DSN
command (TSO) 205
command processor 205
DSN command
TSO
command processor 220
END subcommand 289
TSO applications
running 217
DSN1COPY utility 454
data
restoring 466
log RBA
resetting 533
DSN1LOGP utility
examples 509
limitations 533
log records
extracting 599
printing 599
lost work
showing 500
output 510
DSNC DISCONNECT command
CICS 290
DSNC DISPLAY command
CICS 290, 292
PLAN option 292
TRANSACTION option 292
DSNC MODIFY command
CICS
ERRDEST option 290
DSNC STOP command
CICS 290, 295
DSNC STRT command
CICS 290, 291, 292
DSNCLI (CICS language interface module)
CICS applications
running 219
DSNDB06 database
high-level qualifier
changing 190
DSNDB07 database 418

DSNDB07 database *(continued)*
data sets
extending 547
problems
resolving 446
DSNDEXPL mapping macro 667
DSNDROW mapping macro 674
DSNDSLRB mapping macro 619
DSNDSLRF mapping macro 625
DSNJSLR macro
log CLOSE requests 627
stand-alone sample program 628
DSNJU003 utility
CRESTART control statement 394
DSNTEJ1S job 76, 78
DSNTIJIC job
improving recovery
inconsistent data 413
DSNTIPA panel
WRITE TO OPER field 341
DSNTIPL panel
BACKOUT DURATION field 372
LIMIT BACKOUT field 372
DSNTIPN panel
LEVEL UPDATE FREQ field 537
DSNTIPS panel
DEFER ALL field 372
RESTART ALL field 372
DSNZPxxx
subsystem parameters module
specifying 212
DSNZPxxx module
ARCHWTOR option 341
dual logging
active log data sets 340
archive log data sets 343
synchronization 340
dual recovery logs 397
dual-BSDS mode
restoring 361
dynamic plan selection in CICS
exit routine. 664

E

EA-enabled page sets 40
edit procedures
changing 159
column boundaries 668
overview 633
parameter list 635
specifying 634
edit routines 669, 670
data type values 673
expected output 637
invoking 634
row formats 668
Enterprise Storage Server
backups 422
entities
attribute names 7
attributes 7, 8
values 8
entity normalization 9
first normal form 10
fourth normal form 13
second normal form 10
entity normalization *(continued)*
third normal form 11
entity relationships
business rules 6
many-to-many relationships 6
many-to-one relationships 6
one-to-many relationships 6
one-to-one relationships 6
error pages
displaying 263
exception status
resetting 441
exit routine 666
general considerations 664
EXPORT command
access method services 193, 443
Extended Remote Copy (XRC) 584

F

failure symptoms
abend
log problems 520
restart failure 514
BSDS (bootstrap data set) 500
CICS
attachment abends 481
task abends 485
waits 480
logs 500
lost information 525
messages
DFH2206 480
DFS555 479
DSNB207I 535
DSNJ 523
DSNJ001I 496
DSNJ004I 490
DSNJ100 523
DSNJ103I 492
DSNJ105I 489
DSNJ106I 490
DSNJ107 523
DSNJ114I 493
DSNM002I 476
DSNM005I 478
DSNM3201I 480
DSNP007I 543
DSNP012I 542
DSNU086I 540, 541
processing failure 471
subsystem termination 486
fast copy function
Enterprise Storage Server FlashCopy 422
RVA SnapShot 422
fast log apply
RECOVER utility 416
field procedures
changing 159
control blocks 651
field-decoding 659
field-definition 649, 655
field-encoding 658
information block (FPIB) 653
invoking 650
overview 648
parameter list 651, 653

field procedures *(continued)*
specifying 650
value descriptor 654
work area 652
fixed-length rows 669
FlashCopy backups
incremental 34
FlashCopy image copies
creating 420
recovery 440
FlashCopy volume backups 464
FORCE option
STOP DB2 command 306
foreign keys
dropping 138
format
column 674
row 674
forward log recovery
failures 515
restart phase 367
scenarios 514

G

general-use programming information, described 689
GET_CONFIG stored procedure
output
filtering 681
GET_MESSAGE stored procedure
output
filtering 681
GET_SYSTEM_INFO stored procedure
output
filtering 681
global transactions 389
GUPI symbols 689

H

hash access 19
altering 152
enabling
altering tables 150
creating tables 47
size 152
space 152
heuristic damage 386
heuristic decisions 386
Hierarchical Storage Manager (DFSMShsm) 193
history tables 53
names
finding 60
requirements 57
HMIGRATE command
DFSMShsm (Hierarchical Storage Manager) 193
HRECALL command
DFSMShsm (Hierarchical Storage Manager) 193

I

I/O errors
archive log data sets
recovering 493
catalog 541
directory 541
I/O errors *(continued)*
occurrence 360
table spaces 540
ICOPY status
clearing 451
identity columns
attributes
altering 160
conditional restart 371
data
loading 79
recovering 449
values
regenerating 448
IFCID (instrumentation facility component identifier)
0330 341, 488
identifiers by number
0129 614
0306 615
IFI (instrumentation facility interface)
log data
decompressing 615
log records
reading 613, 614
READA request 613
READS request 613, 614
image copies
catalog 402, 403
directory 402, 403
frequency 402
incremental 402
recovery speed 402
image copy
data sets, back-level 397
data sets, damaged
renaming, using access methods services 397
DB2 catalog 397
directory 397
user databases 397
IMPORT command
access method services 193, 524
IMS
connecting to DB2
attachment facility 303
authorization IDs 218
connection IDs 218
controlling 296
dependent regions 303
disconnecting applications 306
environment
planning 218
indoubt units of recovery 387
language interface module (DFSLI000)
link-editing 218
loops 476
LTERM authorization IDs
message-driven regions 218
operating
tracing 334
programming
error checking 218
recovery procedures 475, 476, 478
indoubt units of recovery 387
running programs
batch work 218
threads 298, 300
waits 476

IMS commands
CHANGE SUBSYS 296, 302
DISPLAY
SUBSYS option 305
DISPLAY OASN 302, 476
DISPLAY SUBSYS 296, 305
LTERM authorization IDs 305
responses 209
START REGION 304
START SUBSYS 296
STOP REGION 304
STOP SUBSYS 296, 306
TRACE SUBSYS 296
IMS terminals
issuing commands 207
IMS threads
displaying 299
IMS.PROCLIB library
connecting
from dependent regions 303
inconsistent data
identifying 509
recovering 415
indefinite wait condition
recovering 552
index
catalog information about 105, 107
types
primary 107
index entries
sequence 73
index space data sets
deferred allocation 73
index spaces
starting 254
storage
allocating 39
with restrictions
starting 255
index-based partitions
redefining 543
index-controlled partitioning 62
scenario 198
tables
creating 66
indexes
altered tables
recovering 439
altering 167
clustering option 171
varying-length columns 170
columns
adding 168
copying 418
creating 72
creating implicitly 74
defining 73
dropping 171
implementing 72
large objects (LOBs) 73
naming 73
overview 19
partitioned table spaces
rebalancing 122
recovering 439
redefining 171
reorganizing 172
indexes *(continued)*
stopping 264
storage
allocating 93
estimating 93, 95
structure
index trees 94
leaf pages 94
root pages 94
subpages 94
unique
adding columns 169
version numbers
recycling 172
versions 75
indoubt threads
information
displaying 279
recovering 393
resolving 585
status
resetting 393
indoubt units of recovery 478
CICS 482
displaying 293
displaying 300, 308
IMS 476
recovering 301
inconsistent states 363
recovering 309
resolving 389, 394
Resource Recovery Services (RRS) 308
informational COPY-pending status
clearing 451
INSERT statement 83
data
loading 78
examples 82
installation
macros
starting IRLM automatically 276
INSTEAD OF triggers 166
integrated catalog facility
alias names
changing 185
integrated catalog facility catalog
VVDS (VSAM volume data set) failure
recovering 542
Interactive System Productivity Facility (ISPF) 217
internal resource lock manager (IRLM)
controlling 274
IRLM (internal resource lock manager)
connections
monitoring 275
diagnostic traces 335
element name
global mode 276
local mode 276
failure 471
recovery procedures 471
starting
automatically 276
manually 276
starting automatically 212
stopping 276
issuing commands
from application programs 208

J

JCL jobs
  scheduled execution 251

K

key
  foreign
    catalog information 107
  primary
    catalog information 107
keys
  adding
    foreign 136
    parent 136
    unique 136

L

language interface modules
  DSNCLI 219
large objects (LOBs)
  indexes 73
LCID (log control interval definition) 608
leaf pages 94
  indexes 94
links
  non-IBM Web sites 690
LOAD utility
  CCSID option 79
  data
    moving 193
  delimited files 79
  LOG option 451
  storage
    estimating 92
  tables
    availability 79
    loading 79
loading
  data
    DL/I 84
    sequential data sets 79
  tables 78
    INSERT statement 83
    LOAD utility 79
LOB (large object)
  retrieving catalog information 110
LOB table spaces
  pending states
    clearing 454
  recovering 437
LOBs
  invalid
    recovering 539
  storage
    estimating 91
locations
  displaying 280
  non-DB2
    displaying 281
locks
  finding 260
log
  archive
    retaining 397
log *(continued)*
  dual
    recovery 397
log activities
  stand-alone 623
log capture exit routine 599
log capture exit routines
  log records
    reading 629
log capture routines
  overview 661
  specifying 662
log CLOSE requests
  stand-alone 627
log data
  decompressing 615
  reading 615
log GET requests
  stand-alone 625
log initialization phase
  failure recovery 502, 503
log OPEN requests
  stand-alone 623
log RBA (relative byte address)
  data sharing environment 353
  non-data sharing environment 354
  range 353
  resetting 352, 353, 354
  value 353
log record header (LRH) 607
log record sequence number (LRSN) 599
log records
  active
    gathering 614
  checkpoints 604
  contents 599
  control interval definition (LCID) 608
  creating 339
  data compression 599
  extracting 599
  format 608
  header 607
  interpreting 612
  logical 606
  physical 606
  printing 599
  qualifying 617
  RBA (relative byte address) 599
  reading 599, 613, 614, 619, 629
  redo 600
  relative byte address (RBA) 599
  segments 606
  structure
    page set control records 605
  subtype codes 611
  type codes 610
  types 599
  undo 600
  units of recovery 601
  WQAxxx qualification fields 617
log services
  stand-alone 620, 628
logging
  implementing 344
logging attributes
  changing 118
logical data modeling 3

logical data modeling *(continued)*
examples 5
recommendations 5
logical database design 3
Unified Modeling Language (UML) 14
logical page list (LPL) 261, 262
deferred restart 373
displaying 261
pages
recovering 262
removing 262
logs
archiving 348
backward recovery 368
BSDS (bootstrap data set) inventory
altering 362
buffer
retrieving log records 346
displaying
DISPLAY LOG command 352
print log map utility 352
dual
archive logs 360
minimizing restart efforts 523
synchronization 340
establishing hierarchy 339
excessive loss 525
failure
symptoms 500
total loss 525
forward recovery 367
implementing 344
initialization phase 365
keeping
duration 357
managing 337, 346, 403
record retrieval 346
recovery procedures
active logs 488
archive logs 492
recovery scenario 523
restarting 364, 368
truncation 509
lost work
identifying 509
LRH (log record header) 607

M

mapping macro
DSNDEXPL 667
DSNDROW 674
mapping macros
DSNDSLRB 619
DSNDSLRF 625
materialized query tables
altering 155
attributes
altering 157
changing
base tables 157
creating 61
definitions
altering 158
registering 155
media failures
recovering 575
message by identifier
$HASP373 211
DFS058 297
DFS058I 306
DFS3602I 478
DFS3613I 297
DFS554I 479
DFS555A 479
DFS555I 479
DSN1150I 521
DSN1151I 475
DSN1157I 509, 521
DSN1160I 509, 521
DSN1162I 475, 509, 521
DSN1213I 529
DSN2001I 482
DSN2025I 486
DSN2034I 482
DSN2035I 482
DSN2036I 482
DSN3100I 211, 214, 486
DSN3104I 214, 486
DSN3201I 480
DSN9032I 312
DSNB204I 535
DSNB207I 535
DSNB232I 537
DSNC012I 295
DSNC016I 388
DSNC025I 295
DSNI006I 262
DSNI021I 262
DSNI022I 262
DSNJ001I 211, 342, 365, 500, 502
DSNJ002I 342
DSNJ003I 342, 497
DSNJ004I 342, 490
DSNJ005I 342
DSNJ007I 506, 516
DSNJ008E 341
DSNJ012I 506, 516
DSNJ072E 344
DSNJ099I 211
DSNJ100I 495, 496, 500, 523
DSNJ103I 492, 506, 516
DSNJ104I 492, 506, 516
DSNJ105I 489
DSNJ106I 490, 506, 516
DSNJ107I 495, 500, 523
DSNJ108I 495
DSNJ110E 341, 488
DSNJ111E 341, 488
DSNJ113E 506, 516, 522
DSNJ114I 493
DSNJ115I 492
DSNJ1191 500
DSNJ119I 523
DSNJ120I 365, 495, 496
DSNJ124I 490
DSNJ125I 361
DSNJ126I 495
DSNJ127I 211
DSNJ128I 494
DSNJ130I 365
DSNJ139I 342
DSNJ311E 346
DSNJ312I 346

message by identifier *(continued)*
DSNJ317I 346
DSNJ318I 346
DSNJ319I 346
DSNL001I 312
DSNL002I 332
DSNL003I 312
DSNL004I 312
DSNL005I 332
DSNL006I 332
DSNL009I 319
DSNL010I 319
DSNL030I 553
DSNL080I 314
DSNL200I 316
DSNL432I 332
DSNL433I 332
DSNL500I 552
DSNL501I 548, 552
DSNL502I 548, 552
DSNL700I 549
DSNL701I 550
DSNL702I 550
DSNL703I 550
DSNL704I 550
DSNL705I 550
DSNM001I 297, 305
DSNM002I 305, 306, 476, 486
DSNM003I 297, 305
DSNM004I 387, 476
DSNM005I 302, 387, 478
DSNP001I 543
DSNP007I 543
DSNP012I 542
DSNR001I 211
DSNR002I 211, 500
DSNR003I 211, 357, 475, 518, 521
DSNR004I 211, 366, 367, 500, 502, 514
DSNR005I 211, 367, 500, 502, 520
DSNR006I 211, 368, 500
DSNR007I 211, 366, 367
DSNR031I 367
DSNT360I 256, 259, 260, 263
DSNT361I 256, 259, 260, 263, 600
DSNT362I 256, 259, 260, 263
DSNT397I 259, 260, 263
DSNU086I 540, 541
DSNU561I 547
DSNU563I 547
DSNV086E 486
DSNV400I 346
DSNV401I 287, 293, 300, 301, 346, 482
DSNV402I 280, 304, 346
DSNV406I 293, 300, 301, 482
DSNV408I 293, 301, 309, 377, 482
DSNV414I 293, 301, 309, 482
DSNV415I 293, 301, 309, 482
DSNV431I 294
DSNV435I 376
DSNX940I 321
DSNY001I 211
DSNY002I 214
DSNZ002I 211
DXR105E 276
DXR117I 276
DXR1211 276
DXR122E 471
message by identifier *(continued)*
DXR1651 276
EDC3009I 542
IEC161I 535
message processing program (MPP)
connections 303
messages
CICS 210
route codes 209
unsolicited 210
MIGRATE command
DFSMShsm (Hierarchical Storage Manager) 193
modeling
data 4
MODIFY utility
image copies
retaining 413
moving
data 195
tools 193
data sets 196
with utilities 197
without utilities 197
MPP (message processing program)
connections 303
multi-site updates 381
examples 382
multiple systems
conditional restart 386

N

NetView
monitoring errors 330
network ID (NID)
CICS 482
IMS 298, 476
NID (network ID)
CICS 482
IMS 476
thread identification 298
non-data sharing environment
RECOVER TOLOGPOINT option 433
NOT LOGGED attribute 118
table spaces 118
null value
effect on storage space 669
numeric data 675

O

objects
dropped
recovering 455
dropping
avoiding 455
information
displaying 259
not logged
recovering 450
XML
altering implicitly 184
offloading
active logs 340
canceling 356
interruptions 342

offloading *(continued)*
messages 342
quiescing 346
restarting 356
status
DISPLAY LOG command 357
trigger events 341
offloads
canceling 348
operating environments
CICS 219
IMS 218
originating sequence number (OASN) 298
indoubt units of recovery 476

P

packages
invalidated
dropping a table 162
dropping a view 165
dropping an index 171
page errors
logical 261
physical 261
page sets
altering 124
control records 605
copying 418
page sizes
calculating 90
pages
errors 261
index size 94
information
obtaining 261
number of records 90
root 94
parent keys
dropping 138
partial recovery 428
participants
multi-site updates 381
partition order
retrieving
catalog information 103
partition size
increasing 123
partition-by-growth table spaces
recovering 438
partitioned table spaces
partition size
increasing 123
recovering 436
recovering indexes
COPY utility 440
RECOVER utility 440
partitioning columns
nullable 65
partitioning methods
differences 62
partitions
adding 140
altering 142
boundaries
changing 143
extending 146
partitions *(continued)*
boundaries *(continued)*
truncating 147
index-controlled
redefining 543
inserting rows 149
rebalancing with REORG 122
redefining
index-based partitioning 546, 547
redistributing 122
rotating 144
table-controlled
redefining 543
pending definition changes
materializing 176
periods
application 53
system 53
phases of execution
restart 364
physical data modeling 3
physical database design 15
plan selection exit routine
description 664
writing 664
point of consistency 337
IMS 379
multiple system 379
point-in-time recovery 424, 428, 462
catalog 447
data consistency 429
DB2 subsystem 536
directory 447
planning 424
RECOVER utility 427
postponed units of recovery
resolving 376
postponed-abort unit of recovery 384
prefix
command 205
primary space allocation
examples 30
PRINT command
access method services 455
print log map utility 352, 357
before fall back 524
bootstrap data set (BSDS) contents 273
prior point of consistency
recovery procedures 441
product-sensitive programming information, described 689
programming interface information, described 689
PSB name
IMS 218
PSPI symbols 689
PSTOP transaction type 303

Q

QMF-related failures 485
QSAM (queued sequential access method) 343
queued sequential access method (QSAM) 343
QUIESCE option
STOP DB2 command 306

R

RBA (relative byte address)
  range in messages 342
RDO (resource definition online)
  MSGQUEUE attribute 210
  STATSQUEUE attribute 210
REBUILD INDEX utility 73
REBUILD-pending status
  for indexes 397
RECORDING MAX field
  panel DSNTIPA
    preventing frequent BSDS wrapping 522
records
  performance 90
  size
    calculating 90
RECOVER BSDS command
  copying BSDS 361
RECOVER INDOUBT command 293, 309, 393
  free locked resources 482
RECOVER TABLESPACE utility
  modified data
    recovering 524
RECOVER TOLOGPOINT option
  data sharing environment 430
  non-data sharing environment 433
RECOVER utility 427, 583
  catalog tables 447
  data inconsistency 413
  deferred objects 371
  DFSMS concurrent copies 421
  DFSMSdss RESTORE command 33
  directory tables 447
  DSNDB07 database 446
  fast log apply 416
  functions 417
  messages 417
  object-level recoveries 422
  objects 417
  options
    TOCOPY 428
    TOLOGPOINT 428, 474
    TORBA 428
  recovery cycle 572
  restrictions 418
  running in parallel 416
RECOVER-pending status
  clearing 452
recovery
  application changes
    backing out 474
  backward log recovery failures 520
  BSDS (bootstrap data set) 497
  catalog 447
  catalog definitions
    consistency 464
  communications failure 586
  compressed data 435
  data
    moving 435
  data availability
    maximizing 405
  data sets 460
    DB2-managed 458
    DFSMS 421
    DFSMShsm 404
    non-DB2 dump and restore 455
recovery *(continued)*
  databases
    active logs 599
    backup copies 399
    RECOVER TOCOPY 428
    RECOVER TOLOGPOINT 428
    RECOVER TORBA 428
  DB2 outages
    cold start 591
  DB2 subsystem 527, 536, 599
  DDF (distributed data facility) failures 548
  directory 447
  disk failures 472
  distributed data
    planning 396
  down-level page sets 537
  FlashCopy image copies 440
  FlashCopy volume backups 464
  heuristic decisions
    correcting 597
  implications 450
  IMS outage with cold start
    scenario 590
  IMS-related failures
    during indoubt resolution 478
    indoubt units of recovery 476
  inconsistent data 415
    resolving 519
  indexes 397
  indexes on tables
    partitioned table spaces 440
  indoubt threads 585
  indoubt units of recovery
    CICS 293, 482
    IMS 301
  information
    reporting 409
  integrated catalog facility catalog
    VVDS failure 542
  invalid LOBs 539
  LOB table spaces 437
  logs
    truncating 506
  lost status information 511
  media 417
  multiple systems environment 385
  objects
    dropped 455
    identifying 441
  operations 401
  outages
    minimizing 405
  planning 424
  point in time 428, 536
  prior point in time 424
  prior point of consistency 441
  procedures 471
  reducing time 402
  restart 443
  scenarios 591
  segmented table spaces 437
  table spaces 540
    COPY utility 454
    dropped 458
    DSN1COPY utility 454
    point in time 442
    QUIESCE 442

recovery *(continued)*
table spaces *(continued)*
RECOVER TOCOPY 428
RECOVER TOLOGPOINT 428
RECOVER TORBA 428
work file 447
tables 456
temporary resource failures 487
work file table spaces 447
XML table spaces 437
recovery cycle
RECOVER utility 572
RECOVERY option
REPORT utility 474
recovery procedures 543
application program errors 474
CICS-related failures
application failures 480
attachment facility failures 485
indoubt units of recovery 482
not operational 480
DB2-related failures
active log failures 488
archive log failures 492
BSDS (bootstrap data set) 495
catalog or directory I/O errors 541
database failures 535
subsystem termination 486
table space I/O errors 540
IMS-related failures 474
application failures 479
control region failures 476
integrated catalog facility catalog
VVDS failure 543
IRLM failures 471
out-of-disk-space 543
QMF-related failures 485
restart 500
z/OS failures 471
z/OS power failures 471
recovery scenarios
DB2 cold start 595
failures
current status rebuild phase 502
log initialization phase 502
heuristic decisions
making 588
referential constraint
existing tables
adding 135
violation
recovering 547
referential structure
consistency
maintaining 413
referential structures 397
registers 623
relational databases
copying 195
relative byte address (RBA) 342, 353
REMARKS column
SYSTABLES catalog table 102
remote DBMS (database management system)
indoubt units of recovery
resolving 392
remote logical units
failures 552
renaming damaged DB2 data sets 397
reordered row format 671
REORG TABLESPACE utility
examples
rebalancing partitions 122
REBALANCE option 122
rebalancing partitions 122
redistributing partitions 122
REORG utility
data
moving 193
examples 159
LOG option 451
UNLOAD EXTERNAL 193
REPAIR utility
inconsistent data
resolving 533
REPORT utility
options
RECOVERY 474
TABLESPACESET 474
table spaces
recovering 409
REPRO command
access method services 455, 497
RESET INDOUBT command 393
residual recovery entry (RRE) 302
resource definition online (RDO) 210
STATUSQUEUE attribute 290
resource limit facility
recovery preparation 399
resource managers
indoubt units of recovery
resolving 389
Resource Recovery Services (RRS)
abend 308
connections
controlling 307
indoubt units of recovery 308
postponed units of recovery 310
Resource Recovery Services attachment facility (RRSAF)
connections
displaying 310
monitoring 310
disconnecting 311
resource translation table (RTT) 303
restart 371
automatic 369
backward log recovery
failure during 520
phase 368
BSDS (bootstrap data set) problems 523
cold start situations 525
conditional
control record governs 371
excessive loss of active log data 527
total loss of log 526
current status rebuild phase 366
failure recovery 502
data object availability 373
forward log recovery phase 367
failure during 514
implications
table spaces 370
inconsistencies
resolving 529
log data set problems 523

restart *(continued)*
log initialization phase 365
failure recovery 502
lost connections 387
multiple-system environment 385
normal 364
objects
deferring processing 373
recovery 375
recovery preparation 443
restart processing
deferring 372
limiting 515
restarting
DB2 subsystem 363, 372
RESTORE phase
RECOVER utility 417
RESTORE SYSTEM
recovery cycle
establishing 570
RESTORE SYSTEM utility 33, 462, 469, 583
DB2 subsystem
recovering 467
restoring
data 424
databases 462
DB2 subsystem 462
return areas
specifying 616
return codes 623
RFMTTYPE
BRF 671
RRF 671
rollbacks
consistency
maintaining 384
rolling disaster 578
root pages 94
indexes 94
routines
conversion procedures 645, 646, 647
date routines 641, 642, 643, 644
edit routines 634, 637
field procedures 648, 649, 650, 651, 652, 653, 654, 655, 658, 659
log capture routines 661
time routines 641, 642, 643, 644
validation routines 637, 638, 639, 640
writing 633
row format conversion
table spaces 672
row formats 669, 670
ROWID column
data
loading 79
inserting 83
rows
formats for exit routines 668
incomplete 640
RRDF (Remote Recovery Data Facility)
tables
altering 159
RRE (residual recovery entry)
detecting 302
logged at IMS checkpoint 387
not resolved 387
purging 302
RRSAF (Resource Recovery Services attachment facility)
application programs
running 221
RTT (resource translation table)
transaction types 303
RVA (RAMAC Virtual Array)
backups 422

S

sample library 78
scheduled tasks
adding 223
defining 225
listing 229
removing 232
status
listing 229
multiple executions 230
stopping 232
updating 232
schema definition
authorization 78
processing 78
schemas
creating 76
implementing 76
processing 76
authorization 76
processor 76
SDSNLOAD library
loading 303
SDSNSAMP library
schema definitions
processing 78
secondary space allocation
examples 30
segmented table spaces
recovering 437
SELECT statement
example
SYSIBM.SYSCOLUMNS 104
SYSIBM.SYSINDEXES 105
SYSIBM.SYSTABAUTH 106
SYSIBM.SYSTABLEPART 103
SYSIBM.SYSTABLES 102, 112
examples 115
SYSIBM.SYSPLANDEP 163
SYSIBM.SYSVIEWDEP 163
sequence
catalog information 112
shortcut keys
keyboard xiv
SIGNON-ID option
IMS 218
SMS (Storage Management Subsystem) 344
solid-state drive (SSD)
migrating 116
space
allocating
for tables 35
space attributes 117
specifying 140
SPUFI
disconnecting 289
SQL statements
ADD ORGANIZE BY HASH clause 150

SQL statements *(continued)*
ALTER TABLE 150
canceling 318
CREATE TABLE
ORGANIZE BY HASH clause 47
SQL transaction
unit of recovery 337
SSM (subsystem member)
error options 303
specifying
on EXEC parameter 303
START DATABASE command 253
DSNDB07 database 446
examples 254
FORCE option 255
SPACENAM option 254
START DB2 command
data access
restricting 213
issuing
from z/OS console 211
modes 306
PARM option 212
restarting 371
START DDF command 312
START FUNCTION SPECIFIC command 268, 269
START REGION command
IMS 304
START SUBSYS command
IMS 296
START TRACE command 334, 614
starting
DB2 515
STOP DATABASE command 253, 264
DSNDB07 database 446
examples 265
SPACENAM option 254
STOP DB2 command 214
FORCE option 306, 371
QUIESCE option 371
STOP DDF command 332
FORCE option 332
MODE(SUSPEND) option 313
QUIESCE option 332
STOP FUNCTION SPECIFIC command 268, 270
STOP REGION command
IMS 304
STOP SUBSYS command
IMS 296, 306
STOP TRACE command 334
STOP transaction type 303
STOP-pending (STOPP) status 264
storage
allocating
for indexes 93
for tables 90
LOBs 91
managing
DFSMShsm 31
reclaiming 162
storage group, DB2
retrieving catalog information 102
storage groups
advantages 23
altering 114
changing 114
control interval (CI) sizing 23
storage groups *(continued)*
creating 24
implementing 22
managing 23
SMS 25
with SMS 114
volumes
adding 114, 115
removing 115
Storage Management Subsystem (SMS)
archive log data sets 344
stored procedures
administration 677
altering 181
common SQL API 677
Complete mode 679
XML input documents 679
XML output documents 680
XML parameter documents 678
creating 84
debugging 326
diagnostic information 326
displaying
statistics 321
thread information 322
dropping 85
external
migrating 329
external SQL
migrating 328
GET_CONFIG
filtering output 681
GET_MESSAGE
filtering output 681
GET_SYSTEM_INFO
filtering output 681
implementing 84
information
displaying 320
migrating 328
monitoring 320
native SQL
migrating 328
prioritizing 336
scheduling 249
SQLCODE -430 682
troubleshooting 682
subsystem member (SSM) 303
syntax diagram
how to read xv
SYS1.LOGREC data set 486
SYS1.PARMLIB library
IRLM
specifying 274
SYSCOPY
catalog table records
retaining 410
SYSLGRNX directory table
REPORT utility information 409
table space records
retaining 410
SYSSYNONYMS catalog table 162
SYSTABLES catalog table 162
system checkpoints
monitoring 351
system period 53
adding 153

system-level backups 425
  conditional restarts 374
  data
    moving 435
  disaster recovery 554
  object-level recoveries 422
system-period data versioning 53
  bitemporal tables 58
  defining 153
  restrictions 54
  temporal tables 55
system-period temporal tables 53
  altering 155
  creating 55, 153
  querying 60
  recovering 469
  requirements 56
system-wide points of consistency 413

T

table
  creating
    description 50
  retrieving
    catalog information 102
    comments 112
  types 50
table check constraint
  catalog information 109
table space set
  recovering 437
table spaces
  altering 117, 176
  copying 418
  creating
    explicitly 42
  data
    loading 78
    rebalancing 122
  defining
    implicitly 42
  dropping 120
  EA-enabled 40
  implementing 41
  logging attribute 451
  not logged 370
  NOT LOGGED attribute 118
  partitioned
    increasing partition size 123
  quiescing 442
  re-creating 120
  recovering 436, 450, 458, 540
  reordered row format
    converting 671, 672
  reorganizing 133
  restoring 532
  row format
    converting 672
  schema changes
    applying 133
  starting 254
  stopping 264
  version numbers
    recycling 134
  versions 132, 133
table-based partitions
  redefining 543
table-controlled partitioning 62
  automatic conversion 63
  nullable partitioning columns 65
  scenario 198
  tables
    creating 62
tables
  altered
    recovering indexes 439
  altering 125
    application period 154
    columns 126
    system period 153
  application-period temporal 53
  bitemporal 53
  changes
    capturing 159
  check constraints
    adding 139
    dropping 139
  clone
    creating 66
  creating 46
  data
    exchanging 68
  dropping
    implications 162
  history 53
  identity column values
    regenerating 448
  identity columns
    recovering 449
  implementing 46
  inserting
    a single row 82
    multiple rows 82
  loading 78
    considerations 83
    INSERT statement 81
    LOAD utility 79
  naming guidelines 47
  page sizes
    altering 165
    moving 165
  populating 78
  re-creating 164
  recovering 456
  rows
    validating 158
  space
    allocating 35
  storage
    allocating 90
    estimating 90
  system-period temporal 53
  temporal 53
  XML columns
    adding 149
tables spaces
  with restrictions
    starting 255
TABLESPACESET option
  REPORT utility 474
takeover site
  setting 576, 577

tapes
archiving 344
TCP/IP
failure recovery 551
temporal tables 53
altering 155
bitemporal 53
creating 53, 55
application period 57
periods 53
querying 60
recovering 469
system-period data versioning 55
temporary tables
created
creating 49
creating 49
types 49
TERM UTILITY command 398
restrictions 415
terminal monitor program (TMP) 220
terminating
DB2
scenarios 486
DB2 subsystem 363
abend 364
normal restart 364
multiple systems 383
termination
types 363
threads
allied 312
attachment in IMS 298
canceling 319
CICS 292
displaying 292
conversation-level information
displaying 282
database access 312
displaying 283, 284
IMS 304
indoubt
displaying 279
monitoring 277, 292
termination
CICS 290
IMS 300, 306
types
active allied 278
active database access 278
pooled database access 278
time routines
expected output 644
invoking 642
overview 641
parameter list 643
specifying 642
TMP (terminal monitor program)
DSN command processor 205
sample job 220
TSO batch work 220
TOCOPY option
RECOVER utility 428
TOLOGPOINT option
RECOVER utility 428
TORBA option
RECOVER utility 428
TRACE SUBSYS command
IMS 296
traces
controlling 333, 334
IMS 334
diagnostic
CICS 334
IRLM 335
tracker site 569
characteristics 569
converting
takeover site 576, 577
disaster recovery 569
maintaining 575
recovering
RECOVER utility 577
RESTORE SYSTEM utility 576
recovery cycle
RESTORE SYSTEM utility 570
setting up 570
transaction managers
distributed transactions
recovering 389
transactions
CICS
accessing 292
entering 219
IMS 218
connecting to DB2 296
thread attachment 298
thread termination 300
types 303
trigger
catalog information 111
troubleshooting
QMF-related failures 485
stored procedures 682
truncation
active logs 341, 509
TSO
application programs
conditions 217
running 217
background execution 220
connections
controlling 287
disconnecting 289
monitoring 287
DSNELI language interface module
link editing 217
TSO commands
DSN 205
END subcommand 289
TSO connections
monitoring 287
TSO consoles
issuing commands 205
two-phase commit
CICS 379
coordinator 379
IMS 379
participants 379
process 379

U

UDF
  catalog information 110
Unified Modeling Language (UML) 14
unit of recovery ID (URID) 608
units of recovery
  in-abort 384
    backward log recovery 368
    excluded in forward log recovery 367
  in-commit 384
    included in forward log recovery 367
  indoubt 214, 384
    CICS 482
    displaying 300
    displaying CICS 293
    IMS 476
    included in forward log recovery 367
    inconsistent states 363
    recovering 309
    recovering CICS 293
    recovering IMS 301
    resolving 387, 388, 389
  inflight 384
    backward log recovery 368
    excluded in forward log recovery 367
  log records 600, 601
  overview 337
  postponed
    CICS 294
    displaying 301, 310
  postponed-abort 384
  Resource Recovery Services (RRS) 310
  rollbacks 384
  rolling back 338
  SQL transactions 337
units of work
  status
    determining 392
UNIX
  cron format 227
UNLOAD utility
  delimited files 79
URID (unit of recovery ID) 608
user-defined data sets
  extending 544
  volumes
    adding 544
user-defined functions
  altering 183
  controlling 268
  creating 86
  dropping 87
  implementing 86
  monitoring 269
  starting 269
  stopping 270
user-managed data sets
  data classes
    specifying 40
  deleting 39
  enlarging 545
  extending 39
  high-level qualifier
    changing 191
  name format 36
  requirements 36
utilities
  online
    changing 272
    controlling 271
    monitoring 272
    starting 271
  running 262
  stand-alone 273
    controlling 271, 272
utility jobs
  running
    recovery procedures 567

V

validation procedures
  column boundaries 668
validation routines 669, 670
  assignments
    altering 158
  data type values 673
  expected output 640
  incomplete 640
  invoking 638
  overview 637
  parameter list 639
  row formats 668
  rows
    checking 158
  specifying 638
VARY WLM command 324
version numbers
  recycling 172
view
  using
    adding comments 112
    retrieving catalog information 105
    retrieving comments 112
views
  altering 165
    INSTEAD OF triggers 166
  creating 69
  dependent objects 163
  dropping 72, 165
    INSTEAD OF triggers 166
  implementing 69
  names 71
  overview 18
  updating 71
virtual storage access method (VSAM) 339
volume serial numbers 360
VSAM (virtual storage access method)
  control interval (CI)
    block size 343
    log records 339
    processing 455
  VSAM volume data set (VVDS) 543
    recovering 542
VTAM
  failures
    recovering 551
VVDS (VSAM volume data set)
  recovering 542

W

wait status
  ending 213
WebSphere Application Server
  indoubt units of recovery 389
WLM application environment
  quiescing 324
  refreshing 324
  restarting 324
  startup procedures 324
  stopping 324
WLM_REFRESH stored procedure 324
work
  submitting 217
work file database
  starting 254
work file databases
  changing high-level qualifier 190
    migrated installation 191
    new installation 190
  data sets
    enlarging 547
  enlarging 543
  extending 547
  troubleshooting 418
work file table spaces
  error ranges
    recovering 447
write error page range (WEPR) 261

X

XML columns
  adding 149
  data
    loading 79
XML input documents
  common SQL API 679
  versioning 678
XML message documents
  versioning 678
XML objects
  altering
    implicitly 184
XML output documents
  common SQL API 680
  versioning 678
XML parameter documents
  versioning 678
XML table spaces
  pending states
    removing 454
  recovering 437
XRC (Extended Remote Copy) 584
XRF (extended recovery facility)
  CICS toleration 396
  IMS toleration 396

Z

z/OS
  commands
    DISPLAY WLM 323
  power failure
    recovering 471
  restart function 369
z/OS abend
  IEC030I 494
  IEC031I 494
  IEC032I 494
z/OS commands
  MODIFY irlmproc 274, 275, 276
  MODIFY irlmproc,ABEND 276
  START irlmproc 274, 276
  STOP irlmproc 274, 276
  TRACE 274
z/OS console
  issuing commands to DB2 205

Product Number: 5605-DB2
5697-P31

Printed in USA

SC19-2968-14

Spine information:

DB2 10 for z/OS

Administration Guide