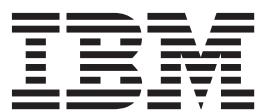


IBM DB2 10.1
for Linux, UNIX, and Windows

*Preparation Guide for DB2 10.1
Advanced DBA for Linux, UNIX, and
Windows Exam 614*

Updated September, 2014



IBM DB2 10.1
for Linux, UNIX, and Windows

*Preparation Guide for DB2 10.1
Advanced DBA for Linux, UNIX, and
Windows Exam 614*

Updated September, 2014



Note

Before using this information and the product it supports, read the general information under Appendix B, "Notices," on page 705.

Edition Notice

This document contains proprietary information of IBM. It is provided under a license agreement and is protected by copyright law. The information contained in this publication does not include any product warranties, and any statements provided in this manual should not be interpreted as such.

You can order IBM publications online or through your local IBM representative.

- To order publications online, go to the IBM Publications Center at <http://www.ibm.com/shop/publications/order>
- To find your local IBM representative, go to the IBM Directory of Worldwide Contacts at <http://www.ibm.com/planetwide/>

To order DB2 publications from DB2 Marketing and Sales in the United States or Canada, call 1-800-IBM-4YOU (426-4968).

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright IBM Corporation 2014.

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

Contents

About this book	ix
Part 1. Database design	1
Chapter 1. Automatic storage	3
Databases use automatic storage by default	4
Chapter 2. Table spaces	5
Table spaces for system, user and temporary data	7
Types of table spaces	8
Automatic storage table spaces	9
How automatic storage table spaces manage storage expansion.	9
Container names in automatic storage table spaces	11
Converting table spaces to use automatic storage	13
The table space high water mark	14
Reclaimable storage	16
File system caching configurations.	22
Extent sizes in table spaces	24
Page, table and table space size.	25
Disk I/O efficiency and table space design	26
Table spaces in a partitioned database environment	28
Creating table spaces	28
Creating temporary table spaces	32
Defining initial table spaces on database creation	33
Altering automatic storage table spaces	34
Reclaiming unused storage in automatic storage table spaces	35
Scenarios: Adding and removing storage with automatic storage table spaces	37
Monitoring a table space rebalance operation	44
Table space states	44
Switching table spaces from offline to online	53
Dropping table spaces	53
Chapter 3. Buffer pools	55
Designing buffer pools	56
Buffer pool hit ratios	57
Buffer pool memory protection (AIX running on POWER6)	57
Buffer pool monitoring in a DB2 pureScale environment	58
Buffer pool hit rates and hit ratios in a DB2 pureScale environment	59
Formulas for calculating buffer pool hit ratios	61
Calculating buffer pool hit ratios in a DB2 pureScale environment	62
Creating buffer pools	65
Modifying buffer pools	66
Dropping buffer pools.	67
Chapter 4. Storage groups	69
Default storage groups	69
Storage group and table space media attributes	70
Creating storage groups	71
Altering storage groups	72
Adding storage paths	72
Dropping storage paths	73
Monitoring storage paths.	74
Replacing the paths of a storage group	75
Renaming storage groups.	75
Dropping storage groups	76
Associating a table space to a storage group	77
Scenario: Moving a table space to a new storage group	77
Chapter 5. Multi-temperature storage	79
Chapter 6. IBM Data Studio	83
Using IBM Data Studio for key tasks	83
IBM Data Studio client	85
IBM Data Studio web console	88
Database administration with IBM Data Studio	89
Administering databases with task assistants	91
Database administration commands that you can run from task assistants	96
Managing jobs in IBM Data Studio	100
Creating and managing jobs	102
Scenario: Creating and scheduling a job	102
Importing tasks from DB2 Task Center	104
Diagramming access plans with Visual Explain	105
Diagrams of access plans	108
Query blocks	108
Setting preferences for Visual Explain	109
Part 2. Data partitioning and clustering	111
Chapter 7. Partitioned database environments	117
Database partitioning across multiple database partitions	118
Database partition groups	119
Distribution maps	122
Distribution keys	123
Table collocation	124
Partition compatibility	124
Setting up partitioned database environments	125
Adding database partition servers to an instance (Windows)	127
Setting up multiple logical partitions	128
Configuring multiple logical partitions	128
Enabling inter-partition query parallelism	129
Enabling intrapartition parallelism for queries	131
Adding database partitions in partitioned database environments	134
Adding an online database partition.	135

Restrictions when working online to add a database partition	136	Logging considerations for MDC and ITC tables	205
Adding a database partition offline (Linux and UNIX)	136	Block indexes for MDC and ITC tables	205
Adding a database partition offline (Windows)	138	Choosing MDC table dimensions	206
Error recovery when adding database partitions	139	Scenario: MDC tables	213
Enabling communication between database partitions using FCM communications	140	Scenario: Creating an ITC table	216
Managing database partitions	141	Scenario: Converting an existing table to an ITC table	216
Listing database partition servers in an instance (Windows)	142		
Eliminating duplicate entries from a list of machines in a partitioned database environment	142		
Specifying the list of machines in a partitioned database environment	143		
Changing the database configuration across multiple database partitions	144		
Adding containers to SMS table spaces on database partitions	144		
Using database partition expressions	144		
Changing database partitions (Windows)	146		
Redistributing data in a database partition group	148		
Issuing commands in partitioned database environments	148		
Dropping database partitions	159		
Tables in partitioned database environments	160		
Redistributing data across database partitions	162	Creating partitioned tables	256
Data redistribution	162	Defining ranges on partitioned tables	257
Determining if data redistribution is needed	165	Placement of the data, index and long data of a data partition	260
Prerequisites for data redistribution	166	Altering partitioned tables	261
Restrictions on data redistribution	168	Guidelines and restrictions on altering partitioned tables	262
Best practices for data redistribution	170	Special considerations for XML indexes when altering a table to ADD, ATTACH, or DETACH a partition	264
Data redistribution mechanism	171	Scenario: Rotating data in a partitioned table	265
Redistributing data across database partitions by using the REDISTRIBUTE DATABASE PARTITION GROUP command	172	Scenarios: Rolling in and rolling out partitioned table data	267
Redistributing database partition groups using the STEPWISE_REDISTRIBUTE_DBPG procedure	173	Migrating existing tables and views to partitioned tables	270
Monitoring a data redistribution operation	175	Converting existing indexes to partitioned indexes	272
Redistribution event log files	176		
Scenario: Redistributing data in new database partitions	180		
Chapter 8. Multidimensional clustering tables	185		
New insert time clustering tables	185		
Comparison of regular and MDC tables	185		
Block indexes for MDC tables	187		
Block indexes and query performance for MDC tables	189		
Maintaining clustering automatically during INSERT operations	193		
Block maps for MDC and ITC tables	195		
Updates to MDC and ITC tables	197		
Deleting from MDC and ITC tables	197		
Multidimensional and insert time clustering extent management	198		
Creating MDC or ITC tables	198		
Load for MDC and ITC tables	204		
Chapter 9. Partitioned tables	217		
Table partitioning keys	217		
Table partitioning and multidimensional clustering tables	219		
Optimization strategies for partitioned tables	224		
Partitioned materialized query table (MQT) behavior	229		
Large object behavior in partitioned tables	232		
Data partitions and ranges	233		
Adding data partitions to partitioned tables	234		
Attaching data partitions	236		
Detaching data partitions	246		
Dropping data partitions	255		
Creating partitioned tables	256		
Defining ranges on partitioned tables	257		
Placement of the data, index and long data of a data partition	260		
Altering partitioned tables	261		
Guidelines and restrictions on altering partitioned tables	262		
Special considerations for XML indexes when altering a table to ADD, ATTACH, or DETACH a partition	264		
Scenario: Rotating data in a partitioned table	265		
Scenarios: Rolling in and rolling out partitioned table data	267		
Migrating existing tables and views to partitioned tables	270		
Converting existing indexes to partitioned indexes	272		
Chapter 10. Range-clustered tables	275		
Guidelines for using range-clustered tables	276		
Scenarios: Range-clustered tables	276		
Restrictions on range-clustered tables	277		
Part 3. High Availability and Diagnostics	279		
Chapter 11. Developing a backup and recovery strategy	281		
Database logging	283		
Circular logging	284		
Archive logging	285		
Log control files	286		
Storage considerations for recovery	286		
Archived log file compression	287		
Backup and restore operations between different operating systems and hardware platforms	288		
Log stream merging and log file management in a DB2 pureScale environment	289		

Log sequence numbers in DB2 pureScale environments	293	Rolling forward changes in a table space	388
Including log files with a backup image	293	Database rollforward operations in a DB2 pureScale environment	392
Incremental backup and recovery.	295	Monitoring a rollforward operation	394
Restoring from incremental backup images	296		
Limitations to automatic incremental restore	298		
Chapter 12. Backing up databases	301	Chapter 16. DB2 Workload Manager (WLM)	397
Performing a snapshot backup	303	Workload management concepts	397
Using a split mirror as a backup image.	304	Phases of workload management	397
Using a split mirror as a backup image in a DB2 pureScale environment	305	Frequently asked questions about DB2 workload management	399
Backing up to tape	306	Integration of AIX Workload Manager with DB2 workload management	409
Backing up to named pipes.	308	Integration of Linux workload management with DB2 workload management	414
Backing up partitioned databases.	309	Workload management sample application	420
Backup and restore operations in a DB2 pureScale environment.	310	Workload management scenarios	420
Enabling automatic backup.	315	Scenario: Investigating a workload-related system slowdown	420
Configuring an automated maintenance policy using SYSPROC.AUTOMAINT_SET_POLICY or SYSPROC.AUTOMAINT_SET_POLICYFILE	316	Scenario: Identifying activities that are taking too long to complete	421
Monitoring backup operations.	317	Scenario: How to cancel activities queued for more than an hour	424
Optimizing backup performance	317	Scenario: Moving table spaces to different storage devices	427
Compatibility of online backup and other utilities	318	Additional scenarios	428
Chapter 13. Recovering databases	321	DB2 workload management tutorial	428
Optimizing recovery performance	321		
Recovering data using db2adutl	323		
Recovering a dropped table	336		
Chapter 14. Restoring databases	339	Chapter 17. High availability disaster recovery (HADR)	429
Restoring from a snapshot backup image	341	High Availability Disaster Recovery (HADR) synchronization mode	431
Using incremental restore in a test and production environment.	342	HADR multiple standby databases	435
Performing a redirected restore operation	344	Restrictions for multiple standby databases	436
Redefine table space containers by restoring a database using an automatically generated script	348	Initializing HADR in multiple standby mode	436
Performing a redirected restore using an automatically generated script.	350	Enabling multiple standby mode on a preexisting HADR setup.	438
Cloning a production database using different storage group paths	351	Modifications to a multiple standby database setup	440
Database rebuild	352	Database configuration for multiple HADR standby databases	441
Database rebuild and table space containers	356	Rolling upgrades in HADR multiple standby mode	443
Database rebuild and temporary table spaces	357	High availability disaster recovery (HADR) monitoring in multiple standby mode	444
Choosing a target image for database rebuild	357	Takeover in HADR multiple standby mode	447
Rebuilding selected table spaces	361	Scenario: Deploying an HADR multiple standby database setup	448
Rebuild and incremental backup images	362	Examples: Takeover in HADR multiple standby mode	453
Rebuilding partitioned databases	363	HADR reads on standby feature	458
Restrictions for database rebuild	364	Enabling reads on standby	458
Rebuild sessions - CLP examples	365	Data concurrency on the active standby database	459
Transporting database schemas	373	HADR delayed replay	463
Transportable objects.	375	Recovering data by using HADR delayed replay	464
Transport examples	377	Performing rolling updates in a DB2 High Availability Disaster Recovery (HADR) environment	466
Troubleshooting: transporting schemas	379		
Monitoring the progress of restore operations	380		
Optimizing restore performance	380		
Chapter 15. Rolling forward databases	383		
Rollforward sessions - CLP examples	384		

High availability disaster recovery (HADR)	
support	469
System requirements for DB2 high availability disaster recovery (HADR)	469
Installation and storage requirements for high availability disaster recovery (HADR)	471
HADR and Network Address Translation (NAT) support	472
Restrictions for High Availability Disaster Recovery (HADR)	473
DB2 High availability disaster recovery (HADR)	
management	474
DB2 High Availability Disaster Recovery (HADR) commands	475
Initializing high availability disaster recovery (HADR)	477
Initializing a standby database	479
Using a split mirror as a standby database	480
Using a split mirror as a standby database in a DB2 pureScale environment	482
Database configuration for high availability disaster recovery (HADR)	485
Setting the hadr_timeout and hadr_peer_window database configuration parameters	493
Log archiving configuration for DB2 high availability disaster recovery (HADR)	494
HADR log spooling	496
Index logging and high availability disaster recovery (HADR)	496
High availability disaster recovery (HADR) performance	498
Cluster managers and high availability disaster recovery (HADR)	500
Performing an HADR failover operation	501
Switching database roles in high availability disaster recovery (HADR)	504
Reintegrating a database after a takeover operation	504
Monitoring high availability disaster recovery (HADR) environments	505
Stopping DB2 High Availability Disaster Recovery (HADR)	507
Chapter 18. Problem-determination tools	509
DB2 diagnostic (db2diag) log files	509
Interpretation of diagnostic log file entries	510
Interpreting the informational record of the db2diag log files	513
Setting the error capture level of the diagnostic log files	514
First occurrence data capture information	514
Collecting diagnosis information based on common outage problems	515
First occurrence data capture configuration	517
Data collected as part of FODC	519
Automatic FODC data generation	525
Monitor and audit facilities using First Occurrence Data Capture (FODC)	525
db2ckbkp command	526
db2cklog command	526
Checking archive log files with the db2cklog tool	526
db2ls command	528
Listing DB2 database products installed on your system (Linux and UNIX)	529
db2mtrk command	530
Buffer pools memory allocation	530
Example 1	531
Example 2	531
db2pd command	531
Troubleshooting scripts	545
db2val command	546
Validating your DB2 copy	546
db2dart command	546
Comparison of INSPECT and db2dart	547
Part 4. Performance and scalability	551
Chapter 19. SQL and XQuery compiler	553
Query rewriting methods and examples	556
Compiler rewrite example: Operation merging	557
Compiler rewrite example: Operation movement	559
Compiler rewrite example: Operation movement - Predicate pushdown for combined SQL/XQuery statements	561
Compiler rewrite example: Predicate translation	563
Access plan optimization	564
Optimization classes	564
Choosing an optimization class	567
Setting the optimization class	568
Optimization profiles and guidelines	569
Collecting accurate catalog statistics, including advanced statistics features	571
Configuration parameters that affect query optimization	572
Chapter 20. Memory allocation	575
Database manager shared memory	577
The FCM buffer pool and memory requirements	580
Guidelines for tuning parameters that affect memory usage	580
Chapter 21. Configuring memory and memory heaps	583
Agent and process model configuration	585
Chapter 22. Self-tuning memory	587
Self-tuning memory configuration	588
Enabling self-tuning memory	588
Disabling self-tuning memory	589
Determining which memory consumers are enabled for self tuning	590
Self-tuning memory in partitioned database environments	591
Using self-tuning memory in partitioned database environments	593

Chapter 23. Automatic maintenance	595
Maintenance windows	595
Chapter 24. Automatic table and index maintenance	597
Chapter 25. Automatic statistics collection	599
Chapter 26. Configuration Advisor	605
Tuning configuration parameters using the Configuration Advisor	605
Example: Requesting configuration recommendations using the Configuration Advisor.	606
Chapter 27. Utility throttling	609
Asynchronous index cleanup	609
Asynchronous index cleanup for MDC tables.	611
Chapter 28. Data compression	613
Table compression	613
Value compression	614
Row compression	615
Classic row compression	615
Adaptive compression	616
Estimating storage savings offered by adaptive or classic row compression	619
Creating a table that uses compression	619
Enabling compression in an existing table	621
Changing or disabling compression for a compressed table	622
Compression dictionaries	623
Table-level compression dictionary creation	624
Impact of classic table reorganization on table-level compression dictionaries	626
Multiple compression dictionaries for replication source tables	627
Index compression	627
Backup compression	630
Chapter 29. Relational indexes	633
Indexes on partitioned tables	634
Relational index planning tips	639
Relational index performance tips	642
Online index defragmentation	643
Chapter 30. Parallel processing for applications	645
Intrapartition parallelism improvements	646
Optimization strategies for intra-partition parallelism	647
Part 5. Advanced concepts	651
Chapter 31. Federated systems	653
What is a data source?	654
The federated database	654
Wrappers and wrapper modules	655
How you interact with a federated system	656
The federated server	656
Federated systems and DB2 pureScale	656
Server options that affect federated databases	657
Federated database query-compiler phases	657
Federated database pushdown analysis.	657
Guidelines for determining where a federated query is evaluated.	661
Remote SQL generation and global optimization in federated databases	663
Global analysis of federated database queries	665
Chapter 32. IBM Replication solutions	667
Replication tools	668
Changes to the Replication Center in DB2 10.1	669
Chapter 33. DB2 pureScale feature	671
Extreme capacity	671
Continuous availability	673
Application transparency	674
Getting started with the DB2 pureScale Feature	676
Management of the DB2 pureScale Feature	677
Chapter 34. DB2 audit facility	679
Audit policies	681
Storage and analysis of audit logs	685
The EXECUTE category for auditing SQL statements	688
Part 6. Appendixes	693
Appendix A. DB2 technical information	695
DB2 technical library in hardcopy or PDF format	696
Displaying SQL state help from the command line processor	698
Accessing different versions of the DB2 Information Center	698
Updating the DB2 Information Center installed on your computer or intranet server	699
Manually updating the DB2 Information Center installed on your computer or intranet server	700
DB2 tutorials	702
DB2 troubleshooting information	702
Terms and conditions.	703
Appendix B. Notices	705
Index	709

About this book

This book provides information from the DB2® for Linux, UNIX, and Windows documentation to cover all the objectives that are described in the DB2 10.1 Advanced DBA for Linux, UNIX, and Windows Exam 614.

- Part 1, “Database design,” on page 1 provides information about how to design, create, and manage database objects such as buffer pools and table spaces. It also provides information about how to use IBM® Data Studio to manage databases.
- Part 2, “Data partitioning and clustering,” on page 111 provides information about the three levels of data organization which are database partitioning, data partitioning, and table partitioning.
- Part 3, “High Availability and Diagnostics,” on page 279 provides information about managing database logs for recovery, using advanced backup and advanced recovery features, using Work Load Manager (WLM), using High Availability Disaster Recovery (HADR), and choosing the appropriate diagnostic tool for a given scenario.
- Part 4, “Performance and scalability,” on page 551 provides information about the query optimizer, how to manage and tune databases, instances, application memory, and I/O, how to improve performance with compression, how to resolve performance problems for a given scenario, how to determine the appropriate index for a given scenario, and how to exploit parallelism.
- Part 5, “Advanced concepts,” on page 651 provides information about federated database environments, replication, DB2 pureScale® environments, and the DB2 audit facility.

Passing the DB2 10.1 Advanced DBA for Linux, UNIX, and Windows Exam 614 is one of the requirements to obtain the *IBM Certified Advanced Database Administrator - DB2 10.1 for Linux, UNIX, and Windows* certification. For complete details about this certification and its requirements, visit <http://www.ibm.com/certify/certs/08005004.shtml>.

Part 1. Database design

When designing a database, you are modeling a real business system that contains a set of entities and their characteristics, or attributes, and the rules or relationships between those entities.

The first step is to describe the system that you want to represent. For example, if you were creating a database for publishing system, the system would contain several types of entities, such as books, authors, editors, and publishers.

The database needs to represent not only these types of entities and their attributes, but also a way to relate these entities to each other. For example, you need to represent the relationship between books and their authors, the relationship between books/authors and editors, and the relationship between books/authors and publishers.

Because databases consist of tables, you must construct a set of tables that will best hold this data, with each cell in the table holding a single view. There are many possible ways to perform this task. As the database designer, your job is to come up with the best set of tables possible.

For example, you could create a single table, with many rows and columns, to hold all of the information. However, using this method, some information would be repeated. Secondly, data entry and data maintenance would be time-consuming and error prone. In contrast to this single-table design, a *relational database* allows you to have multiple simple tables, reducing redundancy and avoiding the difficulties posed by a large and unmanageable table. In a relational database, tables should contain information about a single type of entity.

Also, the integrity of the data in a relational database must be maintained as multiple users access and change the data. Whenever data is shared, ensure the accuracy of the values within database tables in any of the following ways:

- Use isolation levels to determine how data is locked or isolated from other processes while the data is being accessed.
- Protect data and define relationships between data by defining constraints to enforce business rules.
- Create triggers that can do complex, cross-table data validation.
- Implement a recovery strategy to protect data so that it can be restored to a consistent state.

Database design is a more complex task than is indicated here, and there are many items that you must consider, such as space requirements, keys, indexes, constraints, security and authorization, and so forth. You can find some of this information in the DB2 Information Center, DB2 best practices, and in the many DB2 retail books that are available on this subject.

Chapter 1. Automatic storage

Automatic storage simplifies storage management for table spaces. You can create storage groups consisting of a collection of storage paths on which the database manager places your data. Then, the database manager manages the container and space allocation for the table spaces as you create and populate them. You can specify the storage paths of the default storage group when creating the database.

When free space is calculated for an automatic storage path for a given database partition, the database manager checks for the existence of the following directories or mount points within the storage path and uses the first one that it finds. You can mount file systems at a point beneath the storage path and the database manager recognizes that the actual amount of free space available for table space containers might not be the same amount that is associated with the storage path directory itself.

1. *storage_path/instance_name/NODE####/database_name*
2. *storage_path/instance_name/NODE####*
3. *storage_path/instance_name*
4. *storage_path/*

Where

- *storage_path* is a storage path associated with the database.
- *instance_name* is the instance under which the database resides.
- *NODE####* corresponds to the database partition number (for example NODE0000 or NODE0001).
- *database_name* is the name of the database.

Consider the example where two logical database partitions exist on one physical machine and the database is being created with a single storage path: /db2data. Each database partition uses this storage path but you might want to isolate the data from each partition within its own file system. In this case, you can create a separate file system for each partition and mount it at /db2data/*instance*/NODE####. When creating containers on the storage path and determining free space, the database manager knows not to retrieve free space information for /db2data, but instead retrieve it for the corresponding /db2data/*instance*/NODE#### directory.

In general, you must have the same storage paths for each partition in a multi-partition database and they must all exist before executing the **CREATE DATABASE** command. One exception to this is where database partition expressions are used within the storage path. Doing this allows the database partition number to be reflected in the storage path such that the resulting path name is different on each partition.

Related concepts:

Chapter 4, “Storage groups,” on page 69

A storage group is a named set of storage paths where data can be stored. Storage groups are configured to represent different classes of storage available to your database system. You can assign table spaces to the storage group that best suits the data. Only automatic storage table spaces use storage groups.

Related tasks:

“Adding storage paths” on page 72

You can add a storage path to a storage group by using the ALTER STOGROUP statement.

Databases use automatic storage by default

Automatic storage can make storage management easier. Rather than managing storage at the table space level using explicit container definitions, storage is managed at the storage group level and the responsibility of creating, extending and adding containers is taken over by the database manager.

Note: Although, you can create a database specifying the AUTOMATIC STORAGE NO clause, the AUTOMATIC STORAGE clause is deprecated and might be removed from a future release.

By default, all databases are created with automatic storage. However, if the database is created specifying the AUTOMATIC STORAGE NO clause it cannot use automatic storage managed table spaces.

When you create a database, by default, a default storage group is automatically created. You can establish one or more initial storage paths for it. As a database grows, the database manager creates containers across those storage paths, and extends them or automatically creates new ones as needed. The list of storage paths can be displayed using the ADMIN_GET_STORAGE_PATHS administrative view.

If a database has no storage groups, you can create a storage group using the CREATE STOGROUP statement. The newly created storage group is the default storage group and all new automatic storage managed table spaces are added to the database using this storage group. You can change the default storage group using the SET AS DEFAULT clause of the CREATE STOGROUP statement or the ALTER STOGROUP statement.

Important:

- Adding storage paths does not convert existing non-automatic storage table spaces to use automatic storage. You can convert database managed (DMS) table spaces to use automatic storage. System managed (SMS) table spaces cannot be converted to automatic storage. See “Converting table spaces to use automatic storage” on page 13 for more information.
- Once a database has storage groups created, it always has at least one storage group. You cannot remove the last storage group from the database manager.
- To help ensure predictable performance, the storage paths added to a storage group should have similar media characteristics.

Chapter 2. Table spaces

A *table space* is a storage structure containing tables, indexes, large objects, and long data. They are used to organize data in a database into logical storage groupings that relate to where data is stored on a system. Table spaces are stored in database partition groups.

Using table spaces to organize storage offers a number of benefits:

Recoverability

Putting objects that must be backed up or restored together into the same table space makes backup and restore operations more convenient, since you can backup or restore all the objects in table spaces with a single command. If you have partitioned tables and indexes that are distributed across table spaces, you can backup or restore only the data and index partitions that reside in a given table space.

More tables

There are limits to the number of tables that can be stored in any one table space; if you have a need for more tables than can be contained in a table space, you need only to create additional table spaces for them.

Automatic storage management

With automatic storage table spaces, storage is managed automatically. The database manager creates and extends containers as needed.

Ability to isolate data in buffer pools for improved performance or memory utilization

If you have a set of objects (for example, tables, indexes) that are queried frequently, you can assign the table space in which they reside a buffer pool with a single CREATE or ALTER TABLESPACE statement. You can assign temporary table spaces to their own buffer pool to increase the performance of activities such as sorts or joins. In some cases, it might make sense to define smaller buffer pools for seldom-accessed data, or for applications that require very random access into a very large table; in such cases, data need not be kept in the buffer pool for longer than a single query.

Table spaces consist of one or more *containers*. A container can be a directory name, a device name, or a file name. A single table space can have several containers. It is possible for multiple containers (from one or more table spaces) to be created on the same physical storage device (although you will get the best performance if each container you create uses a different storage device). If you are using automatic storage table spaces, the creation and management of containers is handled automatically by the database manager. If you are not using automatic storage table spaces, you must define and manage containers yourself.

Figure 1 on page 6 illustrates the relationship between tables and table spaces within a database, and the containers associated with that database.

Database

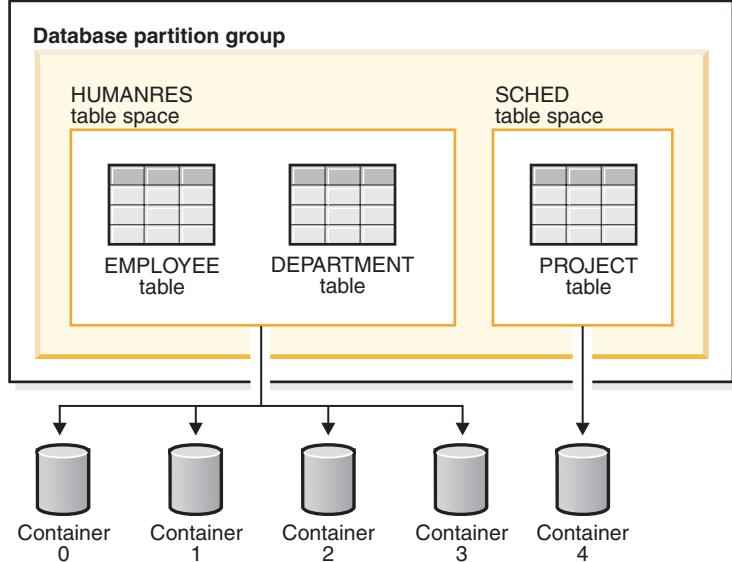


Figure 1. Table spaces and tables in a database

The EMPLOYEE and DEPARTMENT tables are in the HUMANRES table space, which spans containers 0, 1, 2 and 3. The PROJECT table is in the SCHED table space in container 4. This example shows each container existing on a separate disk.

The database manager attempts to balance the data load across containers. As a result, all containers are used to store data. The number of pages that the database manager writes to a container before using a different container is called the *extent size*. The database manager does not always start storing table data in the first container.

Figure 2 on page 7 shows the HUMANRES table space with an extent size of two 4 KB pages, and four containers, each with a small number of allocated extents. The DEPARTMENT and EMPLOYEE tables both have seven pages, and span all four containers.

HUMANRES table space

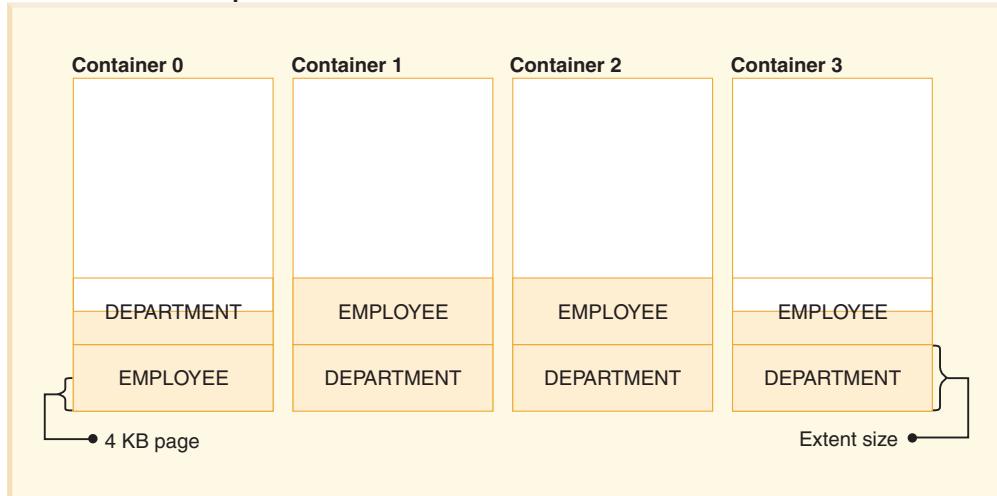


Figure 2. Containers and extents in a table space

Table spaces for system, user and temporary data

Each database must have a minimal set of table spaces that are used for storing system, user and temporary data.

A database must contain at least three table spaces:

- A *catalog table space*
- One or more *user table spaces*
- One or more *temporary table spaces*.

Catalog table spaces

A catalog table space contains all of the system catalog tables for the database. This table space is called SYSCATSPACE, and it cannot be dropped.

User table spaces

A user table space contains user-defined tables. By default, one user table space, USERSPACE1, is created.

If you do not specify a table space for a table at the time you create it, the database manager will choose one for you. Refer to the documentation for the IN *tablespace-name* clause of the CREATE TABLE statement for more information.

The page size of a table space determines the maximum row length or number of columns that you can have in a table. The documentation for the CREATE TABLE statement shows the relationship between page size, and the maximum row size and column count. Before Version 9.1, the default page size was 4 KB. In Version 9.1 and following, the default page size can be one of the other supported values. The default page size is declared when creating a new database. Once the default page size has been declared, you are still free to create a table space with one page size for the table, and a different table space with a different page size for long or LOB data. If the number of columns or the row size exceeds the limits for a table

space's page size, an error is returned (SQLSTATE 42997).

Temporary table spaces

A temporary table space contains temporary tables. Temporary table spaces can be *system temporary table spaces* or *user temporary table spaces*.

System temporary table spaces hold temporary data required by the database manager while performing operations such as sorts or joins. These types of operations require extra space to process the results set. A database must have at least one system temporary table space; by default, one system temporary table space called TEMPSPACE1 is created at database creation.

When processing queries, the database manager might need access to a system temporary table space with a page size large enough to manipulate data related to your query. For example, if your query returns data with rows that are 8KB long, and there are no system temporary table spaces with page sizes of at least 8KB, the query might fail. You might need to create a system temporary table space with a larger page size. Defining a temporary table space with a page size equal to that of the largest page size of your user table spaces will help you avoid these kinds of problems.

User temporary table spaces hold temporary data from tables created with a `DECLARE GLOBAL TEMPORARY TABLE` or `CREATE GLOBAL TEMPORARY TABLE` statement. They are not created by default at the time of database creation. They also hold instantiated versions of created temporary tables. To allow the definition of declared or created temporary tables, at least one user temporary table space should be created with the appropriate `USE` privileges. `USE` privileges are granted using the `GRANT` statement.

If a database uses more than one temporary table space and a new temporary object is needed, the optimizer will choose an appropriate page size for this object. That object will then be allocated to the temporary table space with the corresponding page size. If there is more than one temporary table space with that page size, then the table space will be chosen in a round-robin fashion, starting with one table space with that page size, and then proceeding to the next for the next object to be allocated, and so, returning to the first table space after all suitable table spaces have been used. In most circumstances, though, it is not recommended to have more than one temporary table space with the same page size.

Types of table spaces

Table spaces can be set up in different ways depending on the how you choose to manage their storage.

The three types of table spaces are known as:

- System managed space (SMS), in which the operating system's file manager controls the storage space once you have defined the location for storing database files
- Database managed space (DMS), in which the database manager controls the usage of storage space once you have allocated storage containers.
- Automatic storage table spaces, in which the database manager controls the creation of containers as needed.

Each can be used together in any combination within a database

Automatic storage table spaces

With automatic storage table spaces, storage is managed automatically. The database manager creates and extends containers as needed.

Note: Although you can create a database specifying the AUTOMATIC STORAGE NO clause, the AUTOMATIC STORAGE clause is deprecated and might be removed from a future release.

Any table spaces that you create are managed as automatic storage table spaces unless you specify otherwise or the database was created using the AUTOMATIC STORAGE NO clause. With automatic storage table spaces, you are not required to provide container definitions; the database manager looks after creating and extending containers to make use of the storage allocated to the database. If you add storage to a storage group, new containers are automatically created when the existing containers reach their maximum capacity. If you want to make use of the newly-added storage immediately, you can rebalance the table space, reallocating the data across the new, expanded set of containers and stripe sets. Or, if you are less concerned about I/O parallelism, and just want to add capacity to your table space, you can forego rebalancing; in this case, as new storage is required, new stripe sets will be created.

Automatic storage table spaces can be created in a database using the CREATE TABLESPACE statement. By default, new tables spaces in a database are automatic storage table spaces, so the MANAGED BY AUTOMATIC STORAGE clause is optional. You can also specify options when creating the automatic storage table space, such as its initial size, the amount that the table space size will be increased when the table space is full, the maximum size that the table space can grow to, and the storage group it uses. Following are some examples of statements that create automatic storage table spaces:

```
CREATE TABLESPACE TS1  
CREATE TABLESPACE TS2 MANAGED BY AUTOMATIC STORAGE  
CREATE TEMPORARY TABLESPACE TEMPTS  
CREATE USER TEMPORARY TABLESPACE USRTMP MANAGED BY AUTOMATIC STORAGE  
CREATE LARGE TABLESPACE LONGTS  
CREATE TABLESPACE TS3 INITIALSIZE 8K INCREASESIZE 20 PERCENT MANAGED BY AUTOMATIC STORAGE  
CREATE TABLESPACE TS4 MAXSIZE 2G  
CREATE TABLESPACE TS5 USING STOGROUP SG_HOT
```

Each of these examples assumes that the database for which these table spaces are being created has one or more defined storage groups. When you create a table space in a database that has no storage groups defined, you cannot use the MANAGED BY AUTOMATIC STORAGE clause; you must create a storage group, then try again to create your automatic storage table space.

How automatic storage table spaces manage storage expansion

If you are using *automatic storage table spaces*, the database manager creates and extends containers as needed. If you add storage to the storage group that the table space uses, new containers are created automatically. How the new storage space gets used, however, depends on whether you REBALANCE the table space or not.

When an automatic storage table space is created, the database manager creates a container on each of the storage paths of the storage group it is defined to use (where space permits). Once all of the space in a table space is consumed, the

database manager automatically grows the size of the table space by extending existing containers or by adding a new stripe set of containers.

Storage for automatic table spaces is managed at the storage group level; that is, you add storage to the database's *storage groups*, rather than to table spaces as you do with DMS table spaces. When you add storage to a storage group used by the table space, the automatic storage feature will create new containers as needed to accommodate data. However, table spaces that already exist will not start consuming storage on the new paths immediately. When a table space needs to grow, the database manager will first attempt to extend those containers in the last *range* of the table space. A range is all the containers across a given stripe set. If this is successful, applications will start using that new space. However, if the attempt to extend the containers fails, as might happen when one or more of the file systems are full, for example, the database manager will attempt to create a new stripe set of containers. Only at this point does the database manager consider using the newly added storage paths for the table space. Figure 3 illustrates this process.

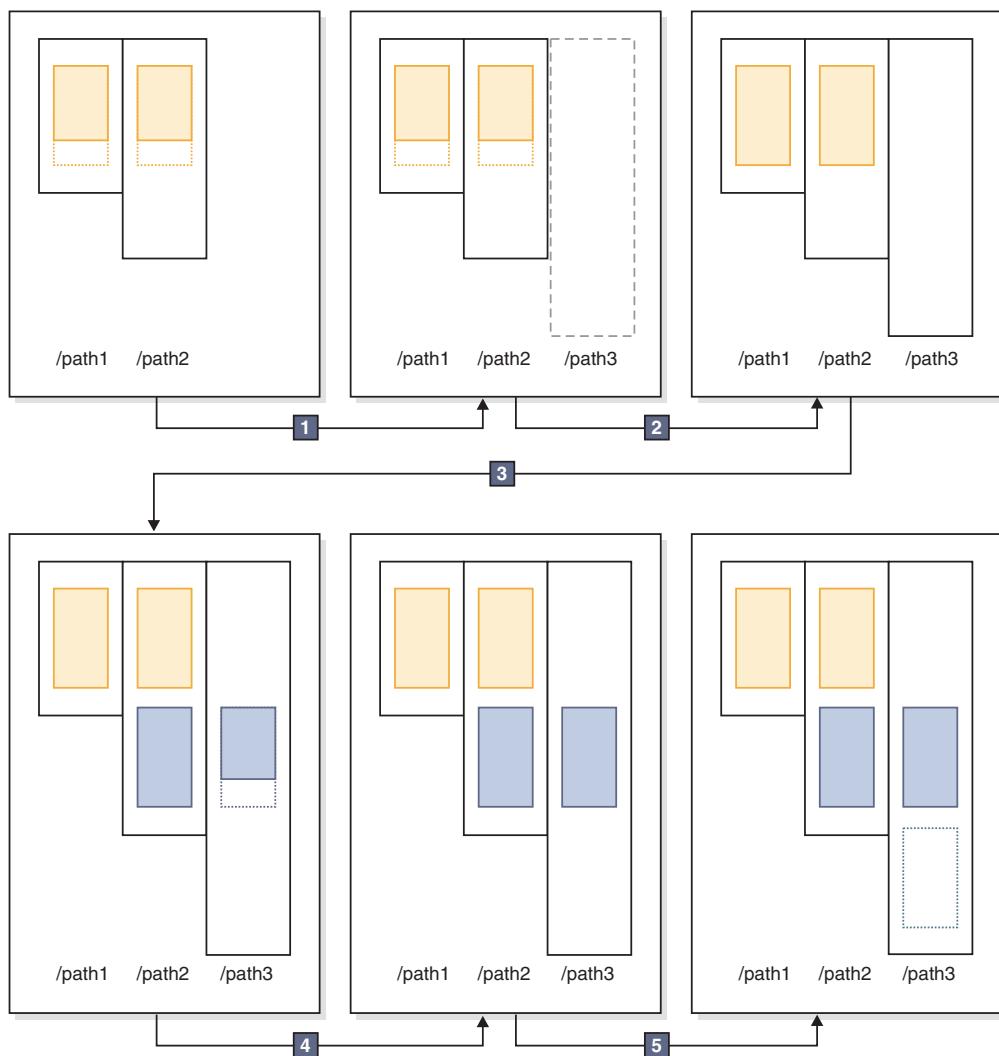


Figure 3. How automatic storage adds containers as needed

In the preceding diagram:

1. The table space starts out with two containers that have not yet reached their maximum capacity. A new storage path is added to the storage group using the ALTER STOGROUP statement with the ADD clause. However, the new storage path is not yet being used.
2. The two original containers reach their maximum capacity.
3. A new stripe set of containers is added, and they start to fill up with data.
4. The containers in the new stripe set reaching their maximum capacity.
5. A new stripe set is added because there is no room for the containers to grow.

If you want to have the automatic storage table space start using the newly added storage path immediately, you can perform a rebalance, using the REBALANCE clause of the ALTER TABLESPACE command. If you rebalance your table space, the data will be reallocated across the containers and stripe sets in the newly-added storage. This is illustrated in Figure 4.

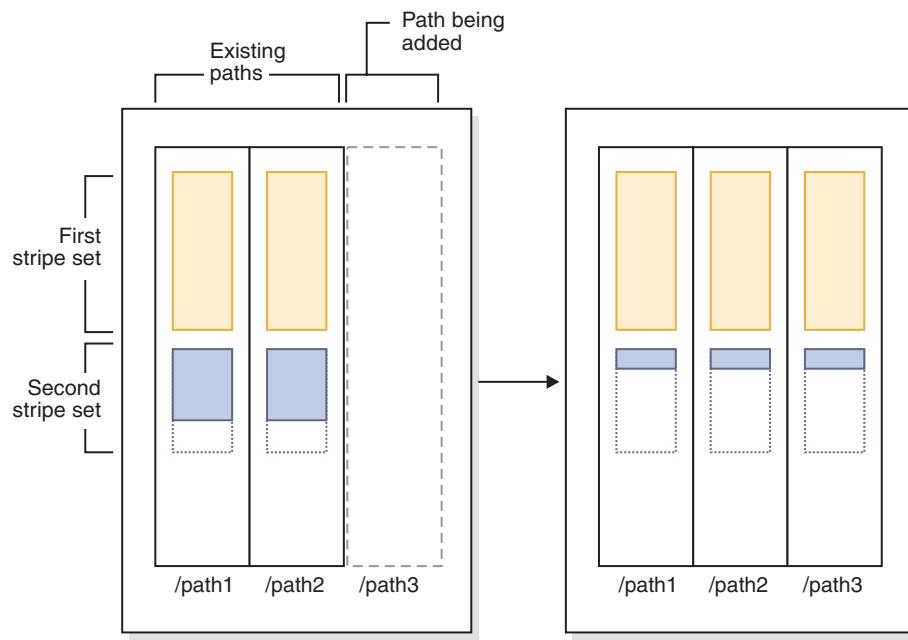


Figure 4. Results of adding new storage and rebalancing the table space

In this example, rather than a new stripe set being created, the rebalance expands the existing stripe sets into the new storage path, creating containers as needed, and then reallocates the data across all of the containers.

Container names in automatic storage table spaces

Although container names for automatic storage table spaces are assigned by the database manager, they are visible if you run commands such as **LIST TABLESPACE CONTAINERS**, or **GET SNAPSHOT FOR TABLESPACES** commands. This topic describes the conventions used for container names so that you can recognize them when they appear.

The names assigned to containers in automatic storage table spaces are structured as follows:

storage path(instance name/NODE####/database name/T#####/C#####.EXT

where:

storage path

Is a storage path associated with a storage group

instance name

Is the instance under which the database was created

database name

Is the name of the database

NODE####

Is the database partition number (for example, NODE0000)

T#####

Is the table space ID (for example, T0000003)

C#####

Is the container ID (for example, C0000012)

EXT Is an extension based on the type of data being stored:

CAT System catalog table space

TMP System temporary table space

UTM User temporary table space

USR User or regular table space

LRG Large table space

Example

For example, assume an automatic storage table space TBSAUTO has been created in the database SAMPLE. When the LIST TABLESPACES command is run, it is shown as having a table space ID of 10:

Tablespace ID	= 10
Name	= TBSAUTO
Type	= Database managed space
Contents	= All permanent data. Large table space.
State	= 0x0000
Detailed explanation:	
Normal	

If you now run the **LIST TABLESPACE CONTAINERS** command for the table space with the ID of 10, you can see the names assigned to the containers for this table space:

LIST TABLESPACE CONTAINERS FOR 10 SHOW DETAIL

Tablespace Containers for Tablespace 10

Container ID	= 0
Name	= D:\DB2\NODE0000\SAMPLE\T0000010\C0000000.LRG
Type	= File
Total pages	= 4096
Useable pages	= 4064
Accessible	= Yes

In this example, you can see the name of the container, with container ID 0, for this table space is

D:\DB2\NODE0000\SAMPLE\T0000010\C0000000.LRG

Converting table spaces to use automatic storage

You can convert some or all of your database-managed space (DMS) table spaces in a database to use automatic storage. Using automatic storage simplifies your storage management tasks.

Before you begin

Ensure that the database has at least one storage group. To do so, query SYSCAT.STOGROUPS, and issue the CREATE STOGROUP statement if the result set is empty.

Note: If you are not using the automatic storage feature, you must not use the storage paths and naming conventions that are used by automatic storage. If you use the same storage paths and naming conventions as automatic storage and you alter a database object to use automatic storage, the container data for that object might be corrupted.

Procedure

To convert a DMS table space to use automatic storage, use one of the following methods:

- **Alter a single table space.** This method keeps the table space online but involves a rebalance operation that takes time to move data from the non-automatic storage containers to the new automatic storage containers.
 1. Specify the table space that you want to convert to automatic storage. Indicate which storage group you want the table space to use. Issue the following statement:

```
ALTER TABLESPACE tbspc1 MANAGED BY AUTOMATIC STORAGE USING STOGROUP sg_medium
```

where *tbspc1* is the table space and *sg_medium* is the storage group it is defined in.
 2. Move the user-defined data from the old containers to the storage paths in the storage group *sg_medium* by issuing the following statement:

```
ALTER TABLESPACE tbspc1 REBALANCE
```

Note: If you do not specify the REBALANCE option now and issue the ALTER TABLESPACE statement later with the REDUCE option, your automatic storage containers will be removed. To recover from this problem, issue the ALTER TABLESPACE statement, specifying the REBALANCE option.

3. To monitor the progress of the rebalance operation, use the following statement:

```
SELECT * from table (MON_GET_REBALANCE_STATUS( 'tbspc1', -2))
```
- **Use a redirected restore operation.** When the redirected restore operation is in progress, you cannot access the table spaces being converted. For a full database redirected restore, all table spaces are inaccessible until the recovery is completed.
 1. Run the RESTORE DATABASE command, specifying the **REDIRECT** parameter. If you want to convert a single table space, also specify the **TABLESPACE** parameter:

```
RESTORE DATABASE database_name TABLESPACE (table_space_name) REDIRECT
```
 2. Run the SET TABLESPACE CONTAINERS command, specifying the **USING AUTOMATIC STORAGE** parameter, for each table space that you want to convert:

```
SET TABLESPACE CONTAINERS FOR tablespace_id USING AUTOMATIC STORAGE
```

3. Run the **RESTORE DATABASE** command again, this time specifying the **CONTINUE** parameter:

```
RESTORE DATABASE database_name CONTINUE
```

4. Run the **ROLLFORWARD DATABASE** command, specifying the **TO END OF LOGS** and **AND STOP** parameters:

```
ROLLFORWARD DATABASE database_name TO END OF LOGS AND STOP
```

If using a redirected restore operation, an additional ALTER TABLESPACE statement must be issued to update the database catalogs with the correct storage group association for the table space. The association between table spaces and storage groups is recorded in the system catalog tables and is not updated during the redirected restore. Issuing the ALTER TABLESPACE statement updates only the catalog tables and does not require the extra processing of a rebalance operation. If the ALTER TABLESPACE statement is not issued then query performance can be affected. If you modified the default storage group for the table space during the redirected restore operation, to keep all database partitions and system catalogs consistent, issue the **RESTORE DATABASE** command with the **USING STOGROUP** parameter.

Example

To convert a database managed table space *SALES* to automatic storage during a redirected restore, do the following:

1. To set up a redirected restore to *testdb*, issue the following command:

```
RESTORE DATABASE testdb REDIRECT
```

2. Modify the table space *SALES* to be managed by automatic storage. The *SALES* table space has an ID value of 5.

```
SET TABLESPACE CONTAINERS FOR 5 USING AUTOMATIC STORAGE
```

Note: To determine the ID value of a table space during a redirect restore use the GENERATE SCRIPT option of the **RESTORE DATABASE** command.

3. To proceed with the restore, issue the following:

```
RESTORE DATABASE testdb CONTINUE
```

4. Update the storage group information in the catalog tables.

```
CONNECT TO testdb  
ALTER TABLESPACE SALES MANAGED BY AUTOMATIC STORAGE
```

5. If you modified the storage group for the table space during the redirected restore operation, issue the following command:

```
RESTORE DATABASE testdb USING STOGROUP sg_default
```

The table space high water mark

The *high water mark* refers to the page number of the first page in the extent following the last allocated extent.

For example, if a table space has 1000 pages and an extent size of 10, there are 100 extents. If the 42nd extent is the highest allocated extent in the table space that means that the high-water mark is 420.

Tip: Extents are indexed from 0. So the high water mark is the *last page of the highest allocated extent + 1*.

Practically speaking, it's virtually impossible to determine the high water mark yourself; there are administrative views and table functions that you can use to

determine where the current high water mark is, though it can change from moment to moment as row operations occur.

Note that the high water mark is not an indicator of the number of used pages because some of the extents below the high-water mark might have been freed as a result of deleting data. In this case, even though there might be free pages below it, the high water mark remains as highest allocated page in the table space.

You can lower the high water mark of a table space by consolidating extents through a table space size reduction operation.

Example

Figure 5 shows a series of allocated extents in a table space.

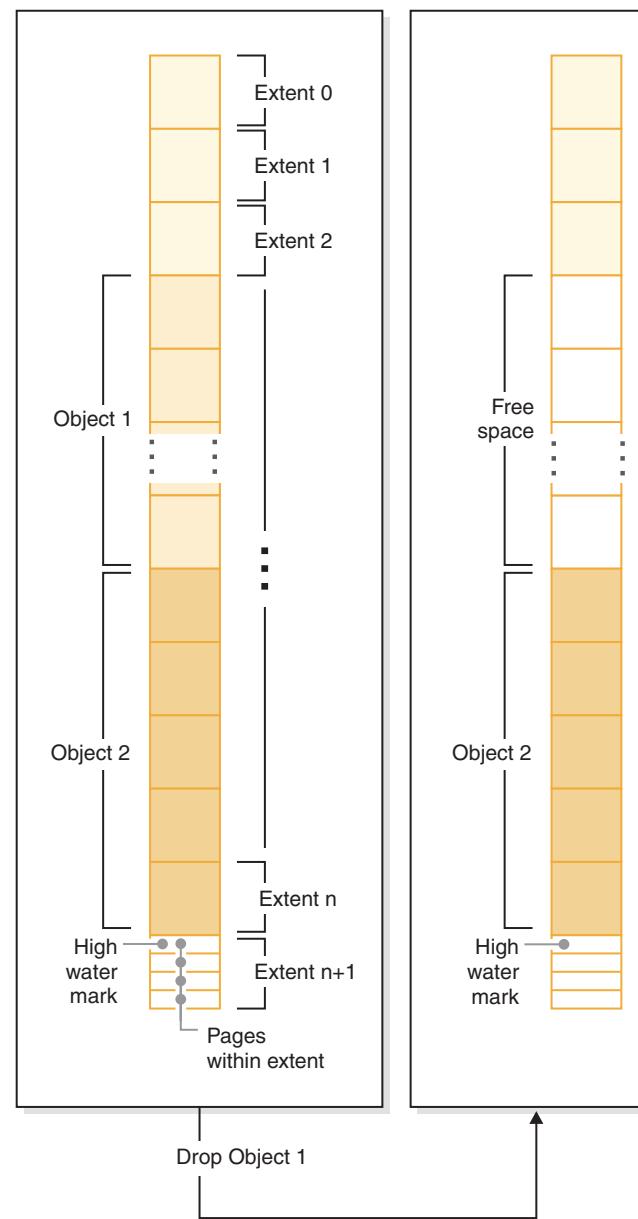


Figure 5. High water mark

When an object is dropped, space is freed in the table space. However, until any kind of storage consolidation operation is performed, the high water mark remains at the previous level. It might even move higher, depending how new extents to the container are added.

Reclaimable storage

Reclaimable storage is a feature of nontemporary automatic storage and DMS table spaces in DB2 V9.7 and later. You can use it to consolidate in-use extents below the *high water mark* and return unused extents in your table space to the system for reuse.

With table spaces created before DB2 V9.7, the only way to release storage to the system was to drop containers, or reduce the size of containers by eliminating unused extents *above* the high water mark. There was no direct mechanism for lowering the high water mark. It could be lowered by unloading and reloading data into an empty table space, or through indirect operations, like performing table and index reorganizations. With this last approach, it might have been that the high water mark could still not be lowered, even though there were free extents below it.

During the extent consolidation process, extents that contain data are moved to unused extents below the high water mark. After extents are moved, if free extents still exist below the high water mark, they are released as free storage. Next, the high water mark is moved to the page in the table space just after the last in-use extent. In table spaces where reclaimable storage is available, you use the ALTER TABLESPACE statement to reclaim unused extents. Figure 6 on page 17 shows a high-level view of how reclaimable storage works.

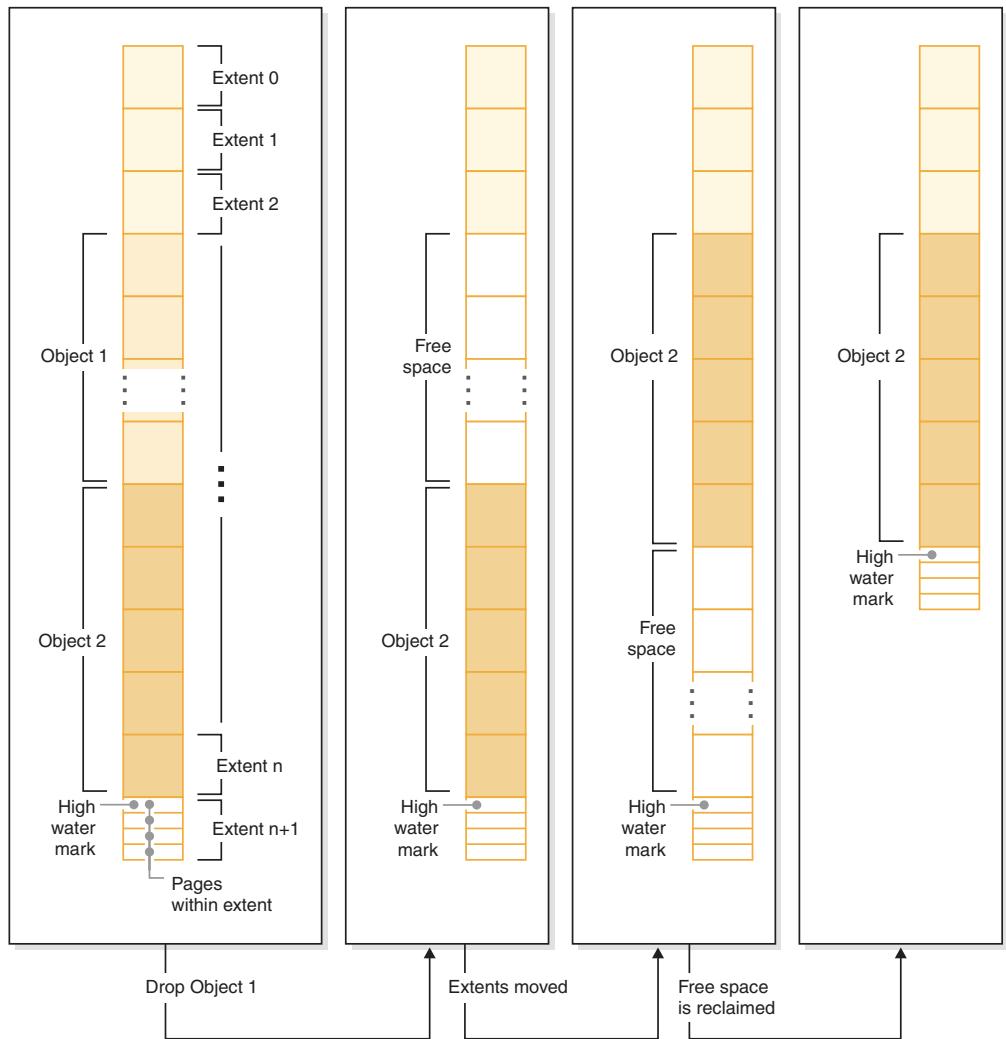


Figure 6. How reclaimable storage works. When reclaimable storage is enabled for a table space, the in-use extents can be moved to occupy unused extents lower in the table space.

All nontemporary automatic storage and DMS table spaces created in DB2 Version 9.7 and later provide the capability for consolidating extents below the high water mark. For table spaces created in an earlier version, you must first replace the table space with a new one created using DB2 V9.7. You can either unload and reload the data or move the data with an online table move operation using the SYSPROC.ADMIN_MOVE_TABLE procedure. Such a migration is not required, however. Table spaces for which reclaimable storage is enabled can coexist in the same database as table spaces without reclaimable storage.

Reducing the size of table spaces through extent movement is an online operation. In other words, data manipulation language (DML) and data definition language (DDL) can continue to be run while the reduce operation is taking place. Some operations, such as a backup or restore cannot run concurrently with extent movement operations. In these cases, the process requiring access to the extents being moved (for example, backup) waits until a number of extents have been moved (this number is non-user-configurable), at which point the backup process obtains a lock on the extents in question, and continues from there.

You can monitor the progress of extent movement using the MON_GET_EXTENT_MOVEMENT_STATUS table function.

Tip: To maximize the amount of space that the ALTER TABLESPACE statement reclaims, first perform a REORG operation on the tables and indexes in the table space.

Automatic storage table spaces

You can reduce automatic storage table spaces in a number of ways:

Container reduction only

With this option, no extents are moved. The database manager attempts to reduce the size of the containers by first freeing extents for which deletes are pending. (It is possible that some “pending delete” extents cannot be freed for recoverability reasons, so some of these extents may remain.) If the high water mark was among those extents freed, then the high water mark is lowered, otherwise no change to the high water mark takes place. Next, the containers are re-sized such that total amount of space in the table space is equal to or slightly greater than the high water mark. This operation is performed using the ALTER TABLESPACE with the REDUCE clause by itself.

Lower high water mark only

With this option, the maximum number of extents are moved to lower the high water mark, however, no container resizing operations are performed. This operation is performed using the ALTER TABLESPACE with the LOWER HIGH WATER MARK clause by itself.

Lower high water mark and reduce containers by a specific amount

With this option, you can specify an absolute amount in kilo-, mega-, or gigabytes by which to reduce the table space. Or you can specify a relative amount to reduce by entering a percentage. Either way, the database manager first attempts to reduce space by the requested amount without moving extents. That is, it attempts to reduce the table space by reducing the container size only, as described in Container reduction only, by freeing delete pending extents, and attempting to lower the high water mark. If this approach does not yield a sufficient reduction, the database manager then begins moving used extents lower in the table space to lower the high water mark. After extent movement has completed, the containers are resized such that total amount of space in the table space is equal to or slightly greater than the high water mark. If the table space cannot be reduced by the requested amount because there are not enough extents that can be moved, the high water mark is lowered as much as possible. This operation is performed using the ALTER TABLESPACE with a REDUCE clause that includes a specified amount by which to reduce the size the table space.

Lower high water mark and reduce containers the maximum amount possible

In this case, the database manager moves as many extents as possible to reduce the size of the table space and its containers. This operation is performed using the ALTER TABLESPACE with the REDUCE MAX clause.

Once the extent movement process has started, you can stop it using the ALTER TABLESPACE statement with the REDUCE STOP clause. Any extents that have been moved are committed, the high water mark lowered as much as possible, and containers are re-sized to the new, lowered high water mark.

DMS table spaces

DMS table spaces can be reduced in two ways:

Container reduction only

With this option, no extents are moved. The database manager attempts to reduce the size of the containers by first freeing extents for which deletes are pending. (It is possible that some ““pending delete”” extents cannot be deleted for recoverability reasons, so some of these extents might remain.) If the high water mark was among those extents freed, then the high water mark is lowered. Otherwise no change to the high water mark takes place. Next, the containers are resized such that total amount of space in the table space is equal to or slightly greater than the high water mark. This operation is performed using the ALTER TABLESPACE with the REDUCE *database-container* clause by itself.

Lower high water mark only

With this option, the maximum number of extents are moved to lower the high water mark, however, no container resizing operations are performed. This operation is performed using the ALTER TABLESPACE with the LOWER HIGH WATER MARK clause by itself.

Lowering the high water mark and reducing container size is a combined, automatic operation with automatic storage table spaces. By contrast, with DMS table spaces, to achieve both a lowered high water mark and smaller container sizes, you must perform two operations:

1. First, you must lower the high water mark for the table space using the ALTER TABLESPACE statement with the LOWER HIGH WATER MARK clause.
2. Next you must use the ALTER TABLESPACE statement with the REDUCE *database-container* clause by itself to perform the container resizing operations.

Once the extent movement process has started, you can stop it using the ALTER TABLESPACE statement with the LOWER HIGH WATER MARK STOP clause. Any extents that have been moved are committed, the high water mark are reduced to its new value.

Examples

Example 1: Reducing the size of an automatic storage table space by the maximum amount.

Assuming a database with one automatic storage table space TS and three tables T1, T2, and T3 exists, we drop tables T1 and T3:

```
DROP TABLE T1  
DROP TABLE T3
```

Now, assuming that the extents are now free, the following statement causes the extents formerly occupied by T1 and T3 to be reclaimed, and the high water mark of the table space reduced:

```
ALTER TABLESPACE TS REDUCE MAX
```

Example 2: Reducing the size of an automatic storage table space by a specific amount.

Assume that we have a database with one automatic storage table space TS and two tables T1, and T2. Next, we drop table T1:

```
DROP TABLE T1
```

Now, to reduce the size of the table space by 1 MB, use the following statement:

```
ALTER TABLESPACE TS REDUCE SIZE 1M
```

Alternatively, you could reduce the table space by a percentage of its existing size with a statement such as this:

```
ALTER TABLESPACE TS REDUCE SIZE 5 PERCENT
```

Example 3: Reducing the size of an automatic storage table space when there is free space below the high water mark.

Like Example 1, assume that we have a database with one automatic storage table space TS and three tables T1, T2, and T3. This time, when we drop T2 and T3, there is a set of five free extents just below the high water mark. Now, assuming that each extent in this case was made up of two 4K pages, there is actually 40 KB of free space just below the high water mark. If you issue a statement such as this one:

```
ALTER TABLESPACE TS REDUCE SIZE 32K
```

the database manager can lower the high water mark and reduce the container size without the need to perform any extent movement. This scenario is illustrated in Figure 7 on page 21

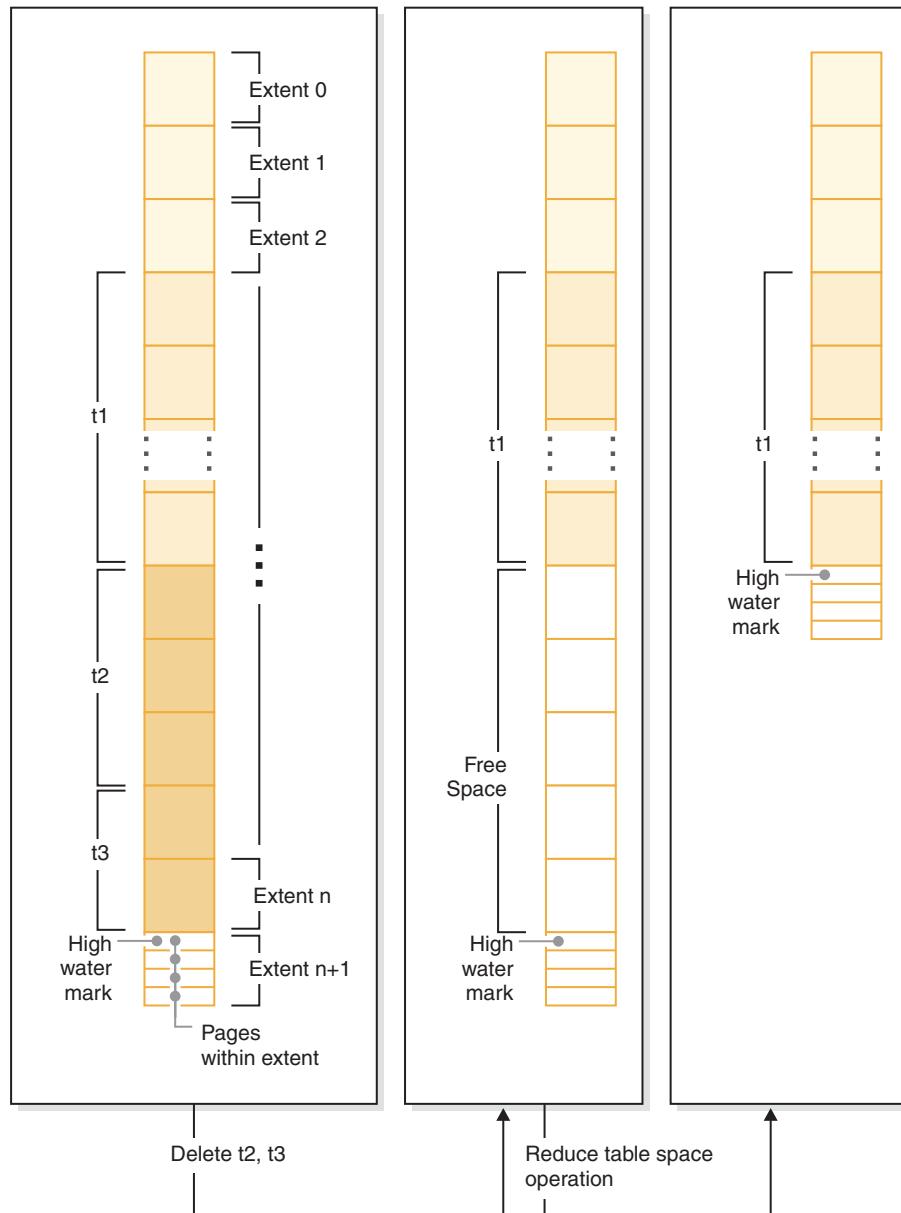


Figure 7. Lowering the high water mark without needing to move extents.

Example 4: Reducing the size of a DMS table space.

Assume that we have a database with one DMS table space TS and three tables T1, T2, and T3. Next, we drop tables T1 and T3:

```
DROP TABLE T1
DROP TABLE T3
```

To lower the high water mark and reduce the container size with DMS table space is a two-step operation. First, lower the high water mark through extent movement with the following statement:

```
ALTER TABLESPACE TS LOWER HIGH WATER MARK
```

Next, you would reduce the size of the containers with a statement such as this one:

```
ALTER TABLESPACE TS REDUCE (ALL CONTAINERS 5 M)
```

File system caching configurations

The operating system, by default, caches file data that is read from and written to disk.

A typical read operation involves physical disk access to read the data from disk into the file system cache, and then to copy the data from the cache to the application buffer. Similarly, a write operation involves physical disk access to copy the data from the application buffer into the file system cache, and then to copy it from the cache to the physical disk. This behavior of caching data at the file system level is reflected in the FILE SYSTEM CACHING clause of the CREATE TABLESPACE statement. Since the database manager manages its own data caching using buffer pools, the caching at the file system level is not needed if the size of the buffer pool is tuned appropriately.

Note: The database manager already prevents caching of most DB2 data, except temporary data and LOBs on AIX®, by invalidating the pages from the cache.

In some cases, caching at the file system level and in the buffer pools causes performance degradation because of the extra CPU cycles required for the double caching. To avoid this double caching, most file systems have a feature that disables caching at the file system level. This is generically referred to as *non-buffered I/O*. On UNIX, this feature is commonly known as *Direct I/O (or DIO)*. On Windows, this is equivalent to opening the file with the FILE_FLAG_NO_BUFFERING flag. In addition, some file systems such as IBM JFS2 or Symantec VERITAS VxFS also support enhanced Direct I/O, that is, the higher-performing *Concurrent I/O (CIO)* feature. The database manager supports this feature with the NO FILE SYSTEM CACHING table space clause. When this is set, the database manager automatically takes advantage of CIO on file systems where this feature exists. This feature might help to reduce the memory requirements of the file system cache, thus making more memory available for other uses.

Before Version 9.5, the keyword FILE SYSTEM CACHING was implied if neither NO FILE SYSTEM CACHING nor FILE SYSTEM CACHING was specified. With Version 9.5, if neither keyword is specified, the default, NO FILE SYSTEM CACHING, is used. This change affects only newly created table spaces. Existing table spaces created prior to Version 9.5 are not affected. This change applies to AIX, Linux, Solaris, and Windows with the following exceptions, where the default behavior remains to be FILE SYSTEM CACHING:

- AIX JFS
- Solaris non-VxFS
- Linux for System z®
- All SMS temporary table space files
- Long Field (LF) and Large object (LOB) data files in SMS permanent table space files.

To override the default setting, specify FILE SYSTEM CACHING or NO FILE SYSTEM CACHING.

Important: The SMS table space type has been deprecated in Version 10.1 for user-defined permanent table spaces and might be removed in a future release. The SMS table space type is not deprecated for catalog and temporary table spaces. For more information, see “SMS permanent table spaces have been deprecated” in *What’s New for DB2 Version 10.1*.

Supported configurations

Table 1 shows the supported configuration for using table spaces without file system caching. It also indicates: (a) whether DIO or enhanced DIO will be used in each case, and (b) the default behavior when neither NO FILE SYSTEM CACHING nor FILE SYSTEM CACHING is specified for a table space based on the platform and file system type.

Table 1. Supported configurations for table spaces without file system caching

Platforms	File system type and minimum level required	DIO or CIO requests submitted by the database manager when NO FILE SYSTEM CACHING is specified	Default behavior when neither NO FILE SYSTEM CACHING nor FILE SYSTEM CACHING is specified
AIX 6.1 and higher	Journal File System (JFS)	DIO	FILE SYSTEM CACHING (See Note 1.)
AIX 6.1 and higher	General Parallel File System (GPFS™)	DIO	NO FILE SYSTEM CACHING
AIX 6.1 and higher	Concurrent Journal File System (JFS2)	CIO	NO FILE SYSTEM CACHING
AIX 6.1 and higher	VERITAS Storage Foundation for DB2 4.1 (VxFS)	CIO	NO FILE SYSTEM CACHING
HP-UX Version 11i v3 (Itanium)	VERITAS Storage Foundation 4.1 (VxFS)	CIO	FILE SYSTEM CACHING
Solaris 10, 11	UNIX File System (UFS)	CIO	FILE SYSTEM CACHING (See Note 2.)
Solaris 10, 11	VERITAS Storage Foundation for DB2 4.1 (VxFS)	CIO	NO FILE SYSTEM CACHING
Linux distributions SLES 10 SP3 or higher, and RHEL 5.2 or higher (on these architectures: x86, x64, POWER®)	ext2, ext3, reiserfs	DIO	NO FILE SYSTEM CACHING
Linux distributions SLES 10 SP3 or higher, and RHEL 5.2 or higher (on these architectures: x86, x64, POWER)	VERITAS Storage Foundation 4.1 (VxFS)	CIO	NO FILE SYSTEM CACHING
Linux distributions SLES 10 SP3 or higher, and RHEL 5.2 or higher (on this architecture: zSeries)	ext2, ext3 or reiserfs on a Small Computer System Interface (SCSI) disks using Fibre Channel Protocol (FCP)	DIO	FILE SYSTEM CACHING
Windows	No specific requirement, works on all DB2 supported file systems	DIO	NO FILE SYSTEM CACHING

Note:

1. On AIX JFS, FILE SYSTEM CACHING is the default.

2. On Solaris UFS, NO FILE SYSTEM CACHING is the default.
3. The VERITAS Storage Foundation for the database manager might have different operating system prerequisites. The platforms listed previously are the supported platforms for the current release. Consult the VERITAS Storage Foundation for DB2 support for prerequisite information.
4. If SFDB2 5.0 is used instead of the previously specified minimum levels, the SFDB2 5.0 MP1 RP1 release must be used. This release includes fixes that are specific to the 5.0 version.
5. If you do not want the database manager to choose NO FILE SYSTEM CACHING for the default setting, specify FILE SYSTEM CACHING in the relevant SQL, commands, or APIs.

Examples

Example 1: By default, this new table space will be created using non-buffered I/O; the NO FILE SYSTEM CACHING clause is implied:

```
CREATE TABLESPACE table space name ...
```

Example 2: On the following statement, the NO FILE SYSTEM CACHING clause indicates that file system level caching will be OFF for this particular table space:

```
CREATE TABLESPACE table space name ... NO FILE SYSTEM CACHING
```

Example 3: The following statement disables file system level caching for an existing table space:

```
ALTER TABLESPACE table space name ... NO FILE SYSTEM CACHING
```

Example 4: The following statement enables file system level caching for an existing table space:

```
ALTER TABLESPACE table space name ... FILE SYSTEM CACHING
```

Extent sizes in table spaces

An *extent* is a block of storage within a table space container. It represents the number of pages of data that will be written to a container before writing to the next container. When you create a table space, you can choose the extent size based on your requirements for performance and storage management.

When selecting an extent size, consider:

- The size and type of tables in the table space.

Space in DMS table spaces is allocated to a table one extent at a time. As the table is populated and an extent becomes full, a new extent is allocated. DMS table space container storage is pre-reserved which means that new extents are allocated until the container is completely used.

Space in SMS table spaces is allocated to a table either one extent at a time or one page at a time. As the table is populated and an extent or page becomes full, a new extent or page is allocated until all of the extents or pages in the file system are used. When using SMS table spaces, multipage file allocation is allowed. Multipage file allocation allows extents to be allocated instead of a page at a time.

Multipage file allocation is enabled by default. The value of the **multipage_alloc** database configuration parameter indicate whether multipage file allocation is enabled.

Note: Multipage file allocation is not applicable to temporary table spaces.

A table is made up of the following separate table objects:

- A data object. This is where the regular column data is stored.
- An index object. This is where all indexes defined on the table are stored.
- A long field (LF) data object. This is where long field data, if your table has one or more LONG columns, is stored.
- Two large object (LOB) data objects. If your table has one or more LOB columns, they are stored in these two table objects:
 - One table object for the LOB data
 - A second table object for metadata describing the LOB data.
- A block map object for multidimensional clustering (MDC) tables.
- An extra XDA object, which stores XML documents.

Each table object is stored separately, and each object allocates new extents as needed. Each DMS table object is also paired with a metadata object called an extent map, which describes all of the extents in the table space that belong to the table object. Space for extent maps is also allocated one extent at a time.

Therefore, the initial allocation of space for an object in a DMS table space is two extents. (The initial allocation of space for an object in an SMS table space is one page.)

If you have many small tables in a DMS table space, you might have a relatively large amount of space allocated to store a relatively small amount of data. In such a case, specify a small extent size. However, if you have a very large table that has a high growth rate, and you are using a DMS table space with a small extent size, you might needlessly allocate additional extents more frequently.

- The type of access to the tables.

If access to the tables includes many queries or transactions that process large quantities of data, prefetching data from the tables might provide significant performance benefits.

- The minimum number of extents required.

If there is not enough space in the containers for five extents of the table space, the table space is not created.

Page, table and table space size

For DMS, temporary DMS and nontemporary automatic storage table spaces, the page size you choose for your database determines the upper limit for the table space size. For tables in SMS and temporary automatic storage table spaces, page size constrains the size of the tables themselves.

You can use a 4K, 8K, 16K or 32K page size limit. Each of these page sizes also has maximums for each of the table space types that you must adhere to.

Table 2 shows the table space size limits for DMS and nontemporary automatic storage table spaces, by page size:

Table 2. Size limits for DMS and nontemporary automatic storage table spaces. DMS and nontemporary automatic storage table spaces are constrained by page size.

Table space type	4K page size limit	8K page size limit	16K page size limit	32K page size limit
DMS and nontemporary automatic storage table spaces (regular)	64G	128G	256G	512G

Table 2. Size limits for DMS and nontemporary automatic storage table spaces (continued). DMS and nontemporary automatic storage table spaces are constrained by page size.

Table space type	4K page size limit	8K page size limit	16K page size limit	32K page size limit
DMS, temporary DMS and nontemporary automatic storage table spaces (large)	8192G	16 384G	32 768G	65 536G

Table 3 shows the table size limits tables in SMS and temporary automatic storage table spaces, by page size:

Table 3. Size limits for tables in SMS and temporary automatic storage table spaces. With tables in SMS and temporary automatic storage table spaces, it is the table objects themselves, not the table spaces that are constrained by page size.

Table space type	4K page size limit	8K page size limit	16K page size limit	32K page size limit
SMS	64G	128G	256G	512G
Temporary SMS, temporary automatic storage	8192G	16 384G	32 768G	65 536G

For database and index page size limits for the different types of table spaces, see the database manager page size-specific limits in “SQL and XML limits” in the *SQL Reference*.

Disk I/O efficiency and table space design

The type and design of your table space determines the efficiency of the I/O performed against that table space.

You should understand the following concepts before considering other issues concerning table space design and use:

Big-block reads

A read where several pages (usually an extent) are retrieved in a single request. Reading several pages at once is more efficient than reading each page separately.

Prefetching

The reading of pages in advance of those pages being referenced by a query. The overall objective is to reduce response time. This can be achieved if the prefetching of pages can occur asynchronously to the execution of the query. The best response time is achieved when either the CPU or the I/O subsystem is operating at maximum capacity.

Page cleaning

As pages are read and modified, they accumulate in the database buffer pool. When a page is read in, it is read into a buffer pool page. If the buffer pool is full of modified pages, one of these modified pages must be written out to the disk before the new page can be read in. To prevent the buffer pool from becoming full, page cleaner agents write out modified pages to guarantee the availability of buffer pool pages for future read requests.

Whenever it is advantageous to do so, the database manager performs big-block reads. This typically occurs when retrieving data that is sequential or partially

sequential in nature. The amount of data read in one read operation depends on the extent size - the bigger the extent size, the more pages can be read at one time.

Sequential prefetching performance can be further enhanced if pages can be read from disk into contiguous pages within a buffer pool. Since buffer pools are page-based by default, there is no guarantee of finding a set of contiguous pages when reading in contiguous pages from disk. Block-based buffer pools can be used for this purpose because they not only contain a page area, they also contain a block area for sets of contiguous pages. Each set of contiguous pages is named a block and each block contains a number of pages referred to as blocksize. The size of the page and block area, as well as the number of pages in each block is configurable.

How the extent is stored on disk affects I/O efficiency. In a DMS table space using device containers, the data tends to be contiguous on disk, and can be read with a minimum of seek time and disk latency. If files are being used, a large file that has been pre-allocated for use by a DMS table space also tends to be contiguous on disk, especially if the file was allocated in a clean file space. However, the data might have been broken up by the file system and stored in more than one location on disk. This occurs most often when using SMS table spaces, where files are extended one page at a time, making fragmentation more likely.

You can control the degree of prefetching by changing the PREFETCHSIZE option on the CREATE TABLESPACE or ALTER TABLESPACE statements, or you can set the prefetch size to AUTOMATIC to have the database manager automatically choose the best size to use. (The default value for all table spaces in the database is set by the **dft_prefetch_sz** database configuration parameter.) The PREFETCHSIZE parameter tells the database manager how many pages to read whenever a prefetch is triggered. By setting PREFETCHSIZE to be a multiple of the EXTENTSIZEx parameter on the CREATE TABLESPACE statement, you can cause multiple extents to be read in parallel. (The default value for all table spaces in the database is set by the **dft_extent_sz** database configuration parameter.) The EXTENTSIZEx parameter specifies the number of 4 KB pages that will be written to a container before skipping to the next container.

For example, suppose you had a table space that used three devices. If you set the PREFETCHSIZE to be three times the EXTENTSIZEx, the database manager can do a big-block read from each device in parallel, thereby significantly increasing I/O throughput. This assumes that each device is a separate physical device, and that the controller has sufficient bandwidth to handle the data stream from each device. Note that the database manager might have to dynamically adjust the prefetch parameters at run time based on query speed, buffer pool utilization, and other factors.

Some file systems use their own prefetching method (such as the Journaled File System on AIX). In some cases, file system prefetching is set to be more aggressive than the database manager prefetching. This might cause prefetching for SMS and DMS table spaces with file containers to seem to outperform prefetching for DMS table spaces with devices. This is misleading, because it is likely the result of the additional level of prefetching that is occurring in the file system. DMS table spaces should be able to outperform any equivalent configuration.

For prefetching (or even reading) to be efficient, a sufficient number of clean buffer pool pages must exist. For example, there could be a parallel prefetch request that reads three extents from a table space, and for each page being read, one modified page is written out from the buffer pool. The prefetch request might be slowed

down to the point where it cannot keep up with the query. Page cleaners should be configured in sufficient numbers to satisfy the prefetch request.

Table spaces in a partitioned database environment

In a partitioned database environment, each table space is associated with a specific database partition group. This allows the characteristics of the table space to be applied to each database partition in the database partition group.

When allocating a table space to a database partition group, the database partition group must already exist. The association between the table space and the database partition group is defined when you create the table space using the CREATE TABLESPACE statement.

You cannot change the association between a table space and a database partition group. You can only change the table space specification for individual database partitions within the database partition group using the ALTER TABLESPACE statement.

In a single-partition environment, each table space is associated with a default database partition group as follows:

- The catalog table spaces SYSCATSPACE is associated with IBMCATGROUP
- User table spaces are associated with IBMDEFAULTGROUP
- Temporary table spaces are associated with IBMTEMPGROUP.

In a partitioned database environment, the IBMCATGROUP partition will contain all three default table spaces, and the other database partitions will each contain only TEMPSPACE1 and USERSPACE1.

Creating table spaces

Creating a table space within a database assigns containers to the table space and records its definitions and attributes in the database system catalog.

About this task

For automatic storage table spaces, the database manager assigns containers to the table space based on the storage paths associated with the database.

For non-automatic storage table spaces, you must know the path, device or file names for the containers that you will use when creating your table spaces. In addition, for each device or file container you create for DMS table spaces, you must know the how much storage space you can allocate to each container.

If you are specifying the PREFETCHSIZE, use a value that is a multiple of the EXTENTSIZE value. For example if the EXTENTSIZE is 10, the PREFETCHSIZE should be 20 or 30. You should let the database manager automatically determine the prefetch size by specifying AUTOMATIC as a value.

Use the keywords NO FILE SYSTEM CACHING and FILE SYSTEM CACHING as part of the CREATE TABLESPACE statement to specify whether the database manager uses Direct I/O (DIO) or Concurrent I/O (CIO) to access the table space. If you specify NO FILE SYSTEM CACHING, the database manager attempts to use CIO wherever possible. In cases where CIO is not supported (for example, if JFS is used), the database manager uses DIO instead.

When you issue the CREATE TABLESPACE statement, the dropped table recovery feature is turned on by default. This feature lets you recover dropped table data using table space-level restore and rollforward operations. This is useful because it is faster than database-level recovery, and your database can remain available to users. However, the dropped table recovery feature can have some performance impact on forward recovery when there are many drop table operations to recover or when the history file is very large.

If you plan to drop numerous tables and you use circular logging or you do not want to recover any of the dropped tables, disable the dropped table recovery feature by explicitly setting the DROPPED TABLE RECOVERY option to OFF when you issue the CREATE TABLESPACE statement. Alternatively, you can turn off the dropped table recovery feature after creating the table space by using the ALTER TABLESPACE statement.

Procedure

- To create an automatic storage table space using the command line, enter either of the following statements:

```
CREATE TABLESPACE name
```

or

```
CREATE TABLESPACE name
    MANAGED BY AUTOMATIC STORAGE
```

Assuming the table space is created in an automatic storage database, each of the two previously shown statements is equivalent; table spaces created in such a database will, by default, be automatic storage table spaces unless you specify otherwise.

- To create an SMS table space using the command line, enter:

```
CREATE TABLESPACE name
    MANAGED BY SYSTEM
    USING ('path')
```

Important: The SMS table space type has been deprecated in Version 10.1 for user-defined permanent table spaces and might be removed in a future release. The SMS table space type is not deprecated for catalog and temporary table spaces. For more information, see “SMS permanent table spaces have been deprecated” in *What’s New for DB2 Version 10.1*.

- To create a DMS table space using the command line, enter:

```
CREATE TABLESPACE name
    MANAGED BY DATABASE
    USING (FILE 'path' size)
```

Note that by default, DMS table spaces are created as large table spaces.

After the DMS table space is created, you can use the ALTER TABLESPACE statement to add, drop, or resize containers to a DMS table space and modify the PREFETCHSIZE, OVERHEAD, and TRANSFERRATE settings for a table space. You should commit the transaction issuing the table space statement as soon as possible following the ALTER TABLESPACE SQL statement to prevent system catalog contention.

Important: Starting with Version 10.1 Fix Pack 1, the DMS table space type is deprecated for user-defined permanent table spaces and might be removed in a future release. The DMS table space type is not deprecated for catalog and temporary table spaces. For more information, see “DMS permanent table spaces have been deprecated” in *What’s New for DB2 Version 10.1*.

Example

Example 1: Creating an automatic storage table space on Windows.

The following SQL statement creates an automatic storage table space called RESOURCE in the storage group called STOGROUP1:

```
CREATE TABLESPACE RESOURCE  
    MANAGED BY AUTOMATIC STORAGE  
    USING STOGROUP STOGROUP1
```

Example 2: Creating an SMS table space on Windows.

The following SQL statement creates an SMS table space called RESOURCE with containers in three directories on three separate drives:

```
CREATE TABLESPACE RESOURCE  
    MANAGED BY SYSTEM  
    USING ('d:\acc_tbsp', 'e:\acc_tbsp', 'f:\acc_tbsp')
```

Example 3: Creating a DMS table space on Windows.

The following SQL statement creates a DMS table space with two file containers, each with 5 000 pages:

```
CREATE TABLESPACE RESOURCE  
    MANAGED BY DATABASE  
    USING (FILE'd:\db2data\acc_tbsp' 5000,  
          FILE'e:\db2data\acc_tbsp' 5000)
```

In the previous two examples, explicit names are provided for the containers. However, if you specify relative container names, the container is created in the subdirectory created for the database.

When creating table space containers, the database manager creates any directory levels that do not exist. For example, if a container is specified as /project/user_data/container1, and the directory /project does not exist, then the database manager creates the directories /project and /project/user_data.

Any directories created by the database manager are created with PERMISSION 711. Permission 711 is required for fenced process access. This means that the instance owner has read, write, and execute access, and others have execute access. Any user with execute access also has the authority to traverse through table space container directories. Because only the instance owner has read and write access, the following scenario might occur when multiple instances are being created:

- Using the same directory structure as described previously, suppose that directory levels /project/user_data do not exist.
- user1 creates an instance, named user1 by default, then creates a database, and then creates a table space with /project/user_data/container1 as one of its containers.
- user2 creates an instance, named user2 by default, then creates a database, and then attempts to create a table space with /project/user_data/container2 as one of its containers.

Because the database manager created directory levels /project/user_data with PERMISSION 700 from the first request, user2 does not have access to these directory levels and cannot create container2 in those directories. In this case, the CREATE TABLESPACE operation fails.

There are two methods to resolve this conflict:

1. Create the directory /project/user_data before creating the table spaces and set the permission to whatever access is needed for both

user1 and user2 to create the table spaces. If all levels of table space directory exist, the database manager does not modify the access.

2. After user1 creates /project/user_data/container1, set the permission of /project/user_data to whatever access is needed for user2 to create the table space.

If a subdirectory is created by the database manager, it might also be deleted by the database manager when the table space is dropped.

The assumption in this scenario is that the table spaces are not associated with a specific database partition group. The default database partition group IBMDEFAULTGROUP is used when the following parameter is not specified in the statement:

IN *database_partition_group_name*

Example 4: Creating DMS table spaces on AIX.

The following SQL statement creates a DMS table space on an AIX system using three logical volumes of 10 000 pages each, and specifies their I/O characteristics:

```
CREATE TABLESPACE RESOURCE
    MANAGED BY DATABASE
    USING (DEVICE '/dev/rdb1v6' 10000,
           DEVICE '/dev/rdb1v7' 10000,
           DEVICE '/dev/rdb1v8' 10000)
    OVERHEAD 7.5
    TRANSFERRATE 0.06
```

The UNIX devices mentioned in this SQL statement must already exist, and the instance owner and the SYSADM group must be able to write to them.

Example 5: Creating a DMS table space on a UNIX system.

The following example creates a DMS table space on a database partition group called ODDGROUP in a UNIX multi-partition database.

ODDGROUP must be previously created with a CREATE DATABASE PARTITION GROUP statement. In this case, the ODDGROUP database partition group is assumed to be made up of database partitions numbered 1, 3, and 5. On all database partitions, use the device /dev/hdisk0 for 10 000 4 KB pages. In addition, declare a device for each database partition of 40 000 4 KB pages.

```
CREATE TABLESPACE PLANS IN ODDGROUP
    MANAGED BY DATABASE
    USING (DEVICE '/dev/HDISK0' 10000, DEVICE '/dev/n1hd01' 40000)
        ON DBPARTITIONNUM 1
        (DEVICE '/dev/HDISK0' 10000, DEVICE '/dev/n3hd03' 40000)
        ON DBPARTITIONNUM 3
        (DEVICE '/dev/HDISK0' 10000, DEVICE '/dev/n5hd05' 40000)
        ON DBPARTITIONNUM 5
```

The database manager can greatly improve the performance of sequential I/O using the sequential prefetch facility, which uses parallel I/O.

Example 6: Creating an SMS table space with a page size larger than the default.

You can also create a table space that uses a page size larger than the default 4 KB size. The following SQL statement creates an SMS table space on a Linux and UNIX system with an 8 KB page size.

```
CREATE TABLESPACE SMS8K
  PAGESIZE 8192
  MANAGED BY SYSTEM
  USING ('FSMS_8K_1')
  BUFFERPOOL BUFFPOOL8K
```

Notice that the associated buffer pool must also have the same 8 KB page size.

The created table space cannot be used until the buffer pool it references is activated.

Creating temporary table spaces

Temporary table spaces hold temporary data required by the database manager when performing operations such as sorts or joins, since these activities require extra space to process the results set. You create temporary table spaces using a variation of the CREATE TABLESPACE statement.

About this task

A *system temporary table space* is used to store system temporary tables. A database must always have at least one system temporary table space since system temporary tables can only be stored in such a table space. When a database is created, one of the three default table spaces defined is a system temporary table space called "TEMPSPACE1". You should have at least one system temporary table space of each page size for the user table spaces that exist in your database, otherwise some queries might fail. See "Table spaces for system, user and temporary data" on page 7 for more information.

User temporary table spaces are not created by default when a database is created. If your application programs need to use temporary tables, you must create a user temporary table space where the temporary tables will reside. Like regular table spaces, user temporary table spaces can be created in any database partition group other than IBMTEMPGROUP. IBMDEFAULTGROUP is the default database partition group that is used when creating a user temporary table.

Restrictions

For system temporary table spaces in a partitioned environment, the only database partition group that can be specified when creating a system temporary table space is IBMTEMPGROUP.

Procedure

- To create a system temporary table space in addition to the default TEMPSPACE1, use a CREATE TABLESPACE statement that includes the keywords SYSTEM TEMPORARY. For example:

```
CREATE SYSTEM TEMPORARY TABLESPACE tmp_tbsp
  MANAGED BY SYSTEM
  USING ('d:\tmp_tbsp','e:\tmp_tbsp')
```
- To create a user temporary table space, use the CREATE TABLESPACE statement with the keywords USER TEMPORARY. For example:

```
CREATE USER TEMPORARY TABLESPACE usr_tbsp
  MANAGED BY AUTOMATIC STORAGE
```

Defining initial table spaces on database creation

When a database is created, three table spaces are defined by default. The SYSCATSPACE for the system catalog tables. The TEMPSPACE1 for system temporary tables created during database processing. The USERSPACE1 for user-defined tables and indexes. You can also specify additional user table spaces or characteristics for the default table spaces to be created at the database creation.

About this task

Note: When you first create a database no user temporary table space is created.

Unless otherwise specified, the three default table spaces are managed by automatic storage.

Using the **CREATE DATABASE** command, you can specify the page size for the default buffer pool and the initial table spaces. This default also represents the default page size for all future CREATE BUFFERPOOL and CREATE TABLESPACE statements. If you do not specify the page size when creating the database, the default page size is 4 KB.

To define initial table spaces using the command line, enter:

```
CREATE DATABASE name
  PAGESIZE page size
  CATALOG TABLESPACE
    MANAGED BY AUTOMATIC STORAGE
    EXTENTSIZE value PREFETCHSIZE value
  USER TABLESPACE
    MANAGED BY AUTOMATIC STORAGE
    EXTENTSIZE value PREFETCHSIZE value
  TEMPORARY TABLESPACE
    MANAGED BY AUTOMATIC STORAGE
  WITH "comment"
```

If you do not want to use the default definition for these table spaces, you might specify their characteristics on the **CREATE DATABASE** command. For example, the following command could be used to create your database on Windows:

```
CREATE DATABASE PERSONL
  PAGESIZE 16384
  CATALOG TABLESPACE
    MANAGED BY AUTOMATIC STORAGE
    EXTENTSIZE 16 PREFETCHSIZE 32
  USER TABLESPACE
    MANAGED BY AUTOMATIC STORAGE
    EXTENTSIZE 32 PREFETCHSIZE 64
  TEMPORARY TABLESPACE
    MANAGED BY AUTOMATIC STORAGE
  WITH "Personnel DB for BSchiefer Co"
```

In this example, the default page size is set to 16 384 bytes, and the definition for each of the initial table spaces is explicitly provided. You only need to specify the table space definitions for those table spaces for which you do not want to use the default definition.

Note: When working in a partitioned database environment, you cannot create or assign containers to specific database partitions. First, you must create the database with default user and temporary table spaces. Then you should use the CREATE TABLESPACE statement to create the required table spaces. Finally, you can drop the default table spaces.

The coding of the MANAGED BY phrase on the **CREATE DATABASE** command follows the same format as the MANAGED BY phrase on the **CREATE TABLESPACE** statement.

You can add additional user and temporary table spaces if you want. You cannot drop the catalog table space SYSCATSPACE, or create another one; and there must always be at least one system temporary table space with a page size of 4 KB. You can create other system temporary table spaces. You also cannot change the page size or the extent size of a table space after it has been created.

Altering automatic storage table spaces

Much of the maintenance of automatic storage table spaces is handled automatically. The changes that you can make to automatic storage table spaces are limited to rebalancing, and reducing the size of the overall table space.

Automatic storage table spaces manage the allocation of storage for you, creating and extending containers as needed up to the limits imposed by storage paths. The only maintenance operations that you can perform on automatic storage spaces are:

- Rebalancing
- Reclaiming unused storage by lowering the high water mark
- Reducing the size of the overall table space.
- Changing an automatic storage table space's storage group

You can rebalance an automatic storage table space when you add a storage path to a storage group. This causes the table space to start using the new storage path immediately. Similarly, when you drop a storage path from a storage group, rebalancing moves data out of the containers on the storage paths you are dropping and allocates it across the remaining containers.

Adding new storage paths, or dropping paths is handled at the storage group level. To add storage paths to a database, you use the ADD clause of the **ALTER STOGROUP** statement. You can rebalance or not, as you prefer, though if you do not rebalance, the new storage paths are not used until the containers that existed previously are filled to capacity. If you rebalance, any newly added storage paths become available for immediate use.

To drop storage paths, use the **DROP** clause of the **ALTER STORGROUP** statement. This action puts the storage paths into a “drop pending” state. Growth of containers on the storage path you specify cease. However, before the path can be fully removed from the database, you must rebalance all of the table spaces using the storage path using the **REBALANCE** clause on the **ALTER TABLESPACE** command. If a temporary table space has containers on a storage path in a drop pending state, you can either drop and re-create the table space, or restart the database to remove it from the storage path.

Restriction: You cannot rebalance temporary automatic storage table spaces; rebalancing is supported only for regular and large automatic storage table spaces.

You can reclaim the storage below the high water mark of a table space using the **LOWER HIGH WATER MARK** clause of the **ALTER TABLESPACE** statement. This has the effect of moving as many extents as possible to unused extents lower in the table space. The high water mark for the table space is lowered in the process, however containers remain the size they were before the operation was performed.

Automatic storage table spaces can be reduced in size using the REDUCE option of the ALTER TABLESPACE statement. When you reduce the size of an automatic storage table space, the database manager attempts to lower the high water mark for the table space and reduce the size of the table space containers. In attempting to lower the high water mark, the database manager might drop empty containers and might move used extents to free space nearer the beginning of the table space. Next, containers are resized such that total amount of space in the table space is equal to or slightly greater than the high water mark.

Reclaiming unused storage in automatic storage table spaces

When you reduce the size of an automatic storage table space, the database manager attempts to lower the *high water mark* for the table space and reduce the size of the table space containers. In attempting to lower the high water mark, the database manager might drop empty containers and might move used extents to free space nearer the beginning of the table space. Next, containers are re-sized such that total amount of space in the table space is equal to or slightly greater than the high water mark.

Before you begin

You must have an automatic storage table space that was created with DB2 Version 9.7 or later. Reclaimable storage is not available in table spaces created with earlier versions of the DB2 product. You can see which table spaces in a database support reclaimable storage using the MON_GET_TABLESPACE table function.

About this task

You can reduce the size of an automatic storage space for which reclaimable storage is enabled in a number of ways. You can specify that the database manager reduce the table space by:

- The maximum amount possible
- An amount that you specify in kilobytes, megabytes or gigabytes, or pages
- A percentage of the current size of the table space.

In each case, the database manager attempts to reduce the size by moving extents to the beginning of the table space, which, if sufficient free space is available, will reduce the high water mark of the table space. Once the movement of extents has completed, the table space size is reduced to the new high water mark.

You use the REDUCE clause of the ALTER TABLESPACE statement to reduce the table space size for an automatic storage table space. You can specify an amount to reduce the table space by, as noted previously.

Note:

- If you do not specify an amount by which to reduce the table space, the table space size is reduced as much as possible without moving extents. The database manager attempts to reduce the size of the containers by first freeing extents for which deletes are pending. (It is possible that some “pending delete” extents cannot be freed for recoverability reasons, so some of these extents may remain.) If the high water mark was among those extents freed, then the high water mark is lowered, otherwise no change to the high water mark takes place. Next, the containers are re-sized such that total amount of space in the table space is equal to or slightly greater than the high water mark. This operation is performed using the ALTER TABLESPACE with the REDUCE clause by itself.

- If you only want to lower the high water mark, consolidating in-use extents lower in the table space without performing any container operations, you can use the ALTER TABLESPACE statement with the LOWER HIGH WATER MARK clause.
- Once a REDUCE or LOWER HIGH WATER MARK operation is under way, you can stop it by using the REDUCE STOP or LOWER HIGH WATER MARK STOP clause of the ALTER TABLESPACE statement. Any extents that have been moved will be committed, the high water mark will be reduced to its new value and containers will be re-sized to the new high water mark.

Restrictions

- You can reclaim storage only in table spaces created with DB2 Version 9.7 and later.
- When you specify either the REDUCE or the LOWER HIGH WATER MARK clause on the ALTER TABLESPACE statement, you cannot specify other parameters.
- If the extent holding the page currently designated as the high water mark is in “pending delete” state, the attempt to lower the high water mark through extent movement might fail, and message ADM6008I will be logged. Extents in “pending delete” state cannot always be moved, for recoverability reasons. These extents are eventually freed through normal database maintenance processes, at which point they can be moved.
- The following clauses are not supported with the ALTER TABLESPACE statement when executed in DB2 data sharing environments:
 - ADD *database-container-clause*
 - BEGIN NEW STRIPE SET *database-container-clause*
 - DROP *database-container-clause*
 - LOWER HIGH WATER MARK
 - LOWER HIGH WATER MARK STOP
 - REBALANCE
 - REDUCE *database-container-clause*
 - REDUCE + LOWER HIGH WATER MARK action
 - RESIZE *database-container-clause*
 - USING STOGROUP

Procedure

To reduce the size of an automatic storage table space:

1. Formulate an ALTER TABLESPACE statement that includes a REDUCE clause.
`ALTER TABLESPACE table-space-name REDUCE reduction-clause`
2. Run the ALTER TABLESPACE statement.

Example

Example 1: Reducing an automatic storage table space by the maximum amount possible.

```
ALTER TABLESPACE TS1 REDUCE MAX
```

In this case, the keyword MAX is specified as part of the REDUCE clause, indicating that the database manager should attempt to move the maximum number of extents to the beginning of the table space.

Example 2: Reducing an automatic storage table space by a percentage of the current table space size.

```
ALTER TABLESPACE TS1 REDUCE 25 PERCENT
```

This attempts to reduce the size of the table space TS1 to 75% of its original size, if possible.

Scenarios: Adding and removing storage with automatic storage table spaces

The three scenarios in this section illustrate the impact of adding and removing storage paths on automatic storage table spaces.

Once storage paths have been added to or removed from storage groups, you can use a rebalance operation to create one or more containers on the new storage paths or remove containers from the dropped paths. The following should be considered when rebalancing table spaces:

- If for whatever reason the database manager decides that no containers need to be added or dropped, or if containers could not be added due to "out of space" conditions, then you will receive a warning.
- The REBALANCE clause must be specified on its own.
- You cannot rebalance temporary automatic storage table spaces; only regular and large automatic storage table spaces can be rebalanced.
- The invocation of a rebalance is a logged operation that is replayed during a rollforward (although the storage layout might be different)
- In partitioned database environments, a rebalance is initiated on every database partition in which the table space resides.
- When storage paths are added or dropped, you are not forced to rebalance. In fact, subsequent storage path operations can be performed over time before ever doing a rebalance operation. If a storage path is dropped and is in the "Not In Use" state, then it is dropped immediately as part of the ALTER STOGROUP operation. If the storage path is in the "In Use" state and dropped but table spaces not rebalanced, the storage path (now in the "Drop Pending" state), is not used to store additional containers or data.

Scenario: Adding a storage path and rebalancing automatic storage table spaces

This scenario shows how storage paths are added to a storage group and how a REBALANCE operation creates one or more containers on the new storage paths.

The assumption in this scenario is to add a new storage path to a storage group and have an existing table space be striped across that new path. I/O parallelism is improved by adding a new container into each of the table space's stripe sets.

Use the ALTER STOGROUP statement to add a new storage path to a storage group. Then, use the REBALANCE clause on the ALTER TABLESPACE statement to allocate containers on the new storage path and to rebalance the data from the existing containers into the new containers. The number and size of the containers to be created depend on both the definition of the current stripe sets for the table space and on the amount of free space on the new storage paths.

Figure 8 on page 38 illustrates a storage path being added, with the "before" and "after" layout of a rebalanced table space:

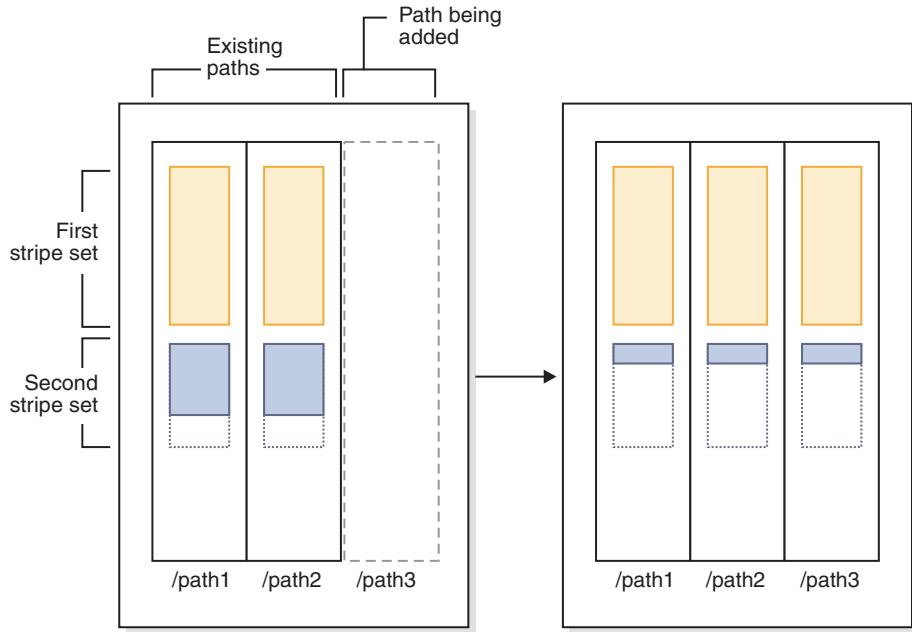


Figure 8. Adding a storage path and rebalancing an automatic storage table space

Note: The diagrams that are displayed in this topic are for illustrative purposes only. They are not intended to suggest a specific approach or best practice for storage layout. Also, the diagrams illustrate a single table space only; in actual practice you would likely have several automatic storage table spaces that share the same storage path.

A similar situation could occur when an existing table space has multiple stripe sets with differing numbers of containers in them, which could have happened due to *disk full* conditions on one or more of the storage paths during the life of the table space. In this case, it would be advantageous for the database manager to add containers to those existing storage paths to fill in the “holes” in the stripe sets (assuming of course that there is now free space to do so). The REBALANCE operation can be used to do this as well.

Figure 9 on page 39 is an example where a “hole” exists in the stripe sets of a table space (possibly caused by deleting table rows, for example) being rebalanced, with the “before” and “after” layout of the storage paths.

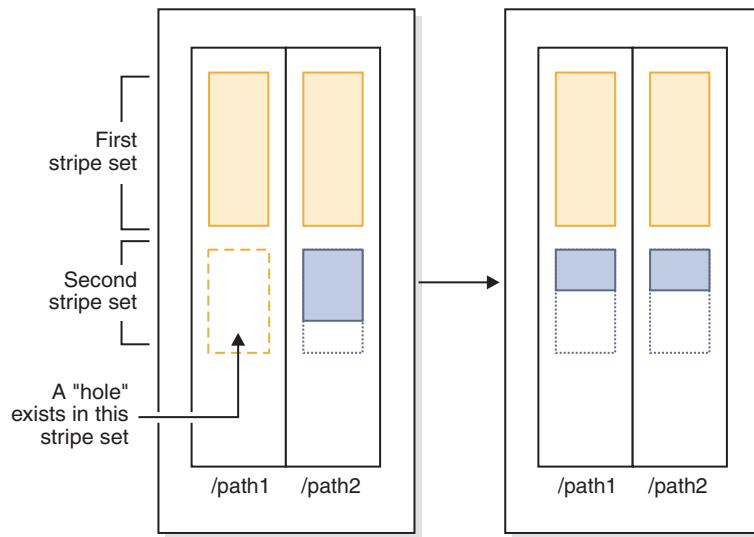


Figure 9. Rebalancing an automatic storage table space to fill gaps

Example

You created a storage group with two storage paths:

```
CREATE STOGROUP sg ON '/path1', '/path2'
```

After creating the database, automatic storage table spaces were subsequently created in this storage group.

You decide to add another storage path to the storage group (/path3) and you want all of the automatic storage table spaces to use the new storage path.

1. The first step is to add the storage path to the storage group:

```
ALTER STOGROUP sg ADD '/path3'
```

2. The next step is to determine all of the affected permanent table spaces. This can be done by manually scanning table space snapshot output or via SQL. The following SQL statement will generate a list of all the regular and large automatic storage table spaces in the storage group:

```
SELECT TBSP_NAME
  FROM table (MON_GET_TABLESPACE(' ', -2))
 WHERE TBSP_USING_AUTO_STORAGE = 1
   AND TBSP_CONTENT_TYPE IN ('ANY', 'LARGE')
   AND STORAGE_GROUP_NAME = 'sg'
 ORDER BY TBSP_ID
```

3. Once the table spaces have been identified, the next step is to perform the following statement for each of the table spaces listed. Provided that there is sufficient space on the remaining storage paths, it generally shouldn't matter what order the rebalances are performed in (and they can be run in parallel).

```
ALTER TABLESPACE tablespace_name REBALANCE
```

After this, you must determine how you want to handle temporary table spaces. One option is to stop (deactivate) and start (activate) the database. This results in the containers being redefined. Alternatively, you can drop and re-create the temporary table spaces, or create a new temporary table space first, then drop the old one-this way you do not attempt to drop the last temporary table space in the database, which is not allowed. To determine the list of affected table spaces, you can manually scan table space snapshot output or you can execute an SQL

statement. The following SQL statement generates a list of all the system temporary and user temporary automatic storage table spaces in the database:

```
SELECT TBSP_NAME
  FROM tabTe (MON_GET_TABLESPACE(' ', -2))
 WHERE TBSP_USING_AUTO_STORAGE = 1
   AND TBSP_CONTENT_TYPE IN ('USRTEMP','SYSTEMP')
   AND STORAGE_GROUP_NAME = 'sg'
 ORDER BY TBSP_ID
```

Scenario: Dropping a storage path and rebalancing automatic storage table spaces

This scenario shows how storage paths are dropped and how the REBALANCE operation drops containers from table spaces that are using the paths.

Before the operation of dropping a storage path can be completed, any table space containers on that path must be removed. If an entire table space is no longer needed, you can drop it before dropping the storage path from the storage group. In this situation, no rebalance is required. If, however, you want to keep the table space, a REBALANCE operation is required. In this case, when there are storage paths in the “drop pending” state, the database manager performs a *reverse rebalance*, where movement of extents starts from the high water mark extent (the last possible extent containing data in the table space), and ends with extent 0.

When the REBALANCE operation is run:

- A reverse rebalance is performed. Data in any containers in the “drop pending” state is moved into the remaining containers.
- The containers in the “drop pending” state are dropped.
- If the current table space is the last table space using the storage path, then the storage path is dropped as well.

If the containers on the remaining storage paths are not large enough to hold all the data being moved, the database manager might have to first create or extend containers on the remaining storage paths before performing the rebalance.

Figure 10 on page 41 is an example of a storage path being dropped, with the “before” and “after” layout of the storage paths after the table space is rebalanced:

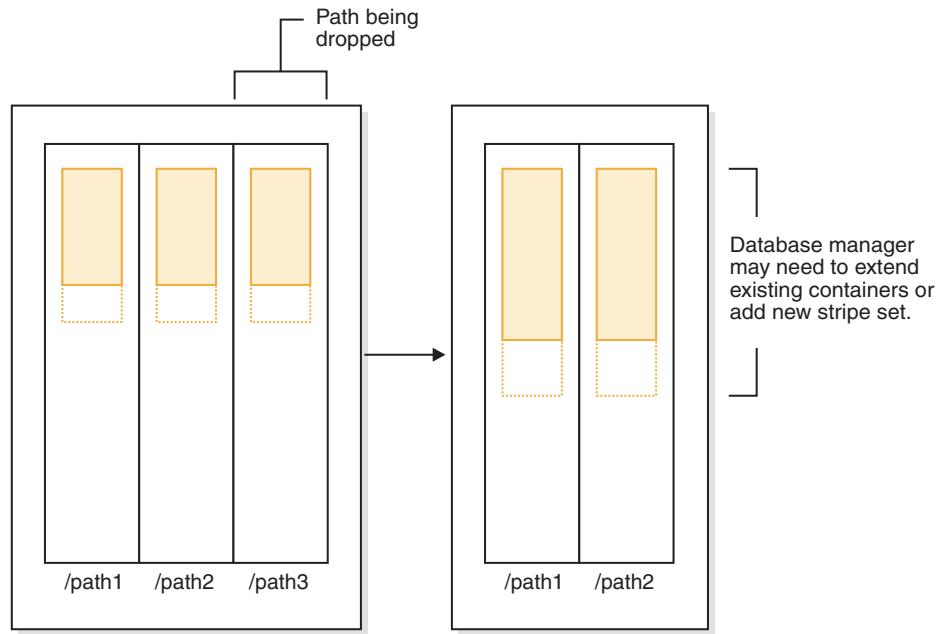


Figure 10. Dropping a storage path and rebalancing an automatic storage table space

Example

Create a storage group with three storage paths:

```
CREATE STOGROUP sg ON '/path1', '/path2', '/path3'
```

After creating the storage group, automatic storage table spaces were subsequently created using it.

You want to put the /path3 storage path into the "Drop Pending" state by dropping it from the storage group, then rebalance all table spaces that use this storage path so that it is dropped.

1. The first step is to drop the storage path from the storage group:

```
ALTER STOGROUP sg DROP '/path3'
```

2. The next step is to determine all the affected non-temporary table spaces. The following SQL statement generates a list of all the regular and large automatic storage table spaces in the database that have containers residing on a "Drop Pending" path:

```
SELECT TBSP_NAME
  FROM table (MON_GET_TABLESPACE(' ', -2))
 WHERE TBSP_USING_AUTO_STORAGE = 1
   AND TBSP_CONTENT_TYPE IN ('ANY', 'LARGE')
   AND STORAGE_GROUP_NAME = 'sg'
 ORDER BY TBSP_ID
```

3. Once the table spaces have been identified, the next step is to perform the following statement for each of the table spaces listed:

```
ALTER TABLESPACE <tablespace_name> REBALANCE
```

- a. If you have dropped multiple storage paths from the storage group and want to free up storage on a specific path, you can query the list of containers in the storage group to find the ones that exist on the storage path. For example, consider a path called /path3. The following query provides a list of table spaces that have containers that reside on path /path3:

```
SELECT TBSP_NAME FROM SYSIBMADM.SNAPCONTAINER  
WHERE CONTAINER_NAME LIKE '/path3'  
GROUP BY TBSP_NAME;
```

- b. You can then issue a REBALANCE statement for each table space in the result set.
4. To determine the list of affected table spaces, generate a list of all the system temporary and user temporary automatic storage table spaces that are defined on the dropped storage paths:

```
SELECT TBSP_NAME  
FROM table (MON_GET_TABLESPACE(' ', -2))  
WHERE TBSP_USING_AUTO_STORAGE = 1  
AND TBSP_CONTENT_TYPE IN ('USRTEMP','SYSTEMP')  
AND STORAGE_GROUP_NAME = 'sg'  
ORDER BY TBSP_ID
```

Scenario: Adding and removing storage paths and rebalancing automatic storage table spaces

This scenario shows how storage paths can be both added and removed, and how the REBALANCE operation rebalances all of the automatic storage table spaces.

It is possible for storage to be added and dropped from a storage group at the same time. This operation can be done by using a single ALTER STOGROUP statement or through multiple ALTER STOGROUP statements separated by some period (during which the table spaces are not rebalanced).

As described in “Scenario: Adding a storage path and rebalancing automatic storage table spaces” on page 37, a situation can occur in which the database manager fills in “holes” in stripe sets when dropping storage paths. In this case the database manager will create containers and drop containers as part of the process. In all of these scenarios, the database manager recognizes that some containers need to be added (where free space allows) and that some need to be removed. In these scenarios, the database manager might need to perform a two-pass rebalance operation (the phase and status of which is described in the snapshot monitor output):

1. First, new containers are allocated on the new paths (or on existing paths if filling in “holes”).
2. A forward rebalance is performed.
3. A reverse rebalance is performed, moving data off the containers on the paths being dropped.
4. The containers are physically dropped.

Figure 11 on page 43 is an example of storage paths being added and dropped, with the “before” and “after” layout of a rebalanced table space:

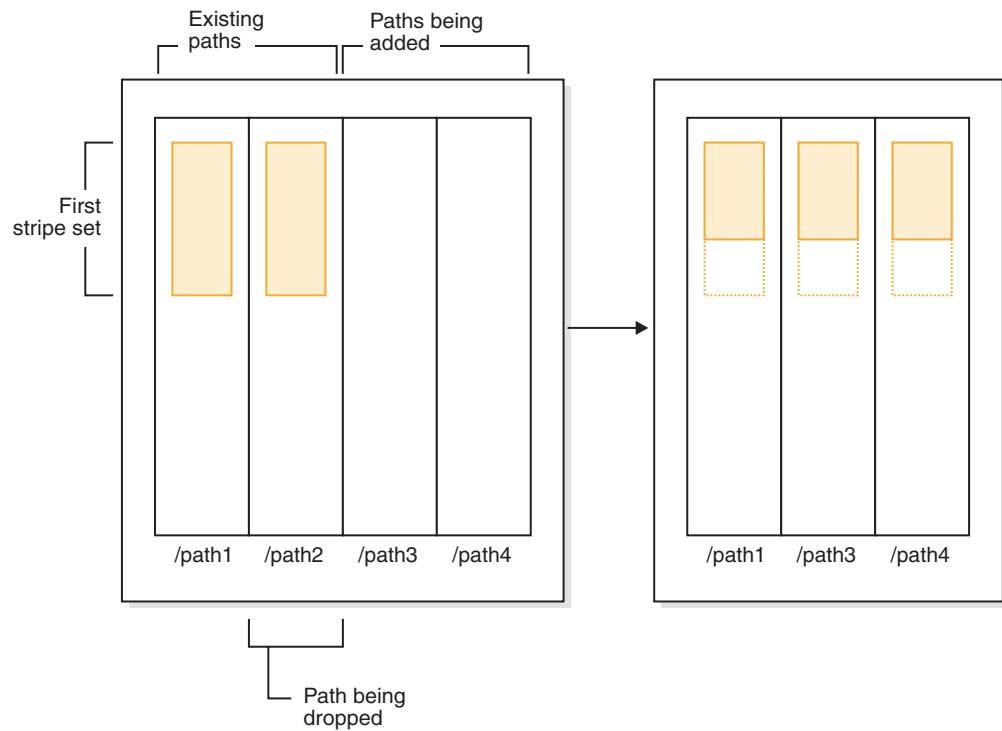


Figure 11. Adding and dropping storage paths, and then rebalancing an automatic storage table space

Example

A storage group is created with two storage paths:

```
CREATE STOGROUP sg ON '/path1', '/path2', '/path4'
```

Assume that you want to add another storage path to the storage group (/path3) and remove one of the existing paths (/path2), and you also want all of your automatic storage table spaces to be rebalanced. The first step is to add the new storage path /path3 to the storage group and to initiate the removal of /path2:

```
ALTER STOGROUP sg ADD '/path3'
ALTER STOGROUP sg DROP '/path2'
```

The next step is to determine all of the affected table spaces. This analysis can be done by manually scanning table space snapshot output or using SQL statements. The following SQL statement generates a list of all the regular and large automatic storage table spaces in the database:

```
SELECT TBSP_NAME
  FROM tabIe (MON_GET_TABLESPACE(' ', -2))
 WHERE TBSP_USING_AUTO_STORAGE = 1
   AND TBSP_CONTENT_TYPE IN ('ANY','LARGE')
   AND STORAGE_GROUP_NAME = 'sg'
 ORDER BY TBSP_ID
```

Once the table spaces are identified, the next step is to perform the following statement for each of the table spaces listed:

```
ALTER TABLESPACE tablespace_name REBALANCE
```

where *tablespace_name* is the name of the table spaces identified in the previous step.

Note: You cannot rebalance temporary table spaces managed by automatic storage. If you want to stop using the storage that was allocated to temporary table spaces, one option is to drop the temporary table spaces and then recreate them.

Monitoring a table space rebalance operation

You can use the **MON_GET_REBALANCE_STATUS** table function to monitor the progress of rebalance operations on a database.

About this task

This procedure returns data for a table space only if a rebalance operation is in progress. Otherwise, no data is returned.

Procedure

To monitor a table space rebalance operation:

Issue the **MON_GET_REBALANCE_STATUS** table function with the **tbsp_name** and **dbpartitionnum** parameters:

```
select
    varchar(tbsp_name, 30) as tbsp_name,
    dbpartitionnum,
    member,
    rebalancer_mode,
    rebalancer_status,
    rebalancer_extents_remaining,
    rebalancer_extents_processed,
    rebalancer_start_time
from table(mon_get_rebalance_status(NULL,-2)) as t
```

Results

This output is typical of the output for monitoring the progress of a table space rebalance operation:

```
TBSP_NAME          DBPARTITIONNUM MEMBER REBALANCER_MODE
-----  -----  -----
SYSCATSPACE           0      0 REV_REBAL

REBALANCER_STATUS REBALANCER_EXTENTS_REMAINING REBALANCER_EXTENTS_PROCESSED REBALANCER_START_TIME
-----  -----  -----  -----
ACTIVE                6517                      4 2011-12-01-12.08.16.000000

1 record(s) selected.
```

Table space states

This topic provides information about the supported table space states.

There are currently at least 25 table or table space states supported by the IBM DB2 database product. These states are used to control access to data under certain circumstances, or to elicit specific user actions, when required, to protect the integrity of the database. Most of them result from events related to the operation of one of the DB2 database utilities, such as the load utility, or the backup and restore utilities. The following table describes each of the supported table space states. The table also provides you with working examples that show you exactly how to interpret and respond to states that you might encounter while administering your database. The examples are taken from command scripts that were run on AIX; you can copy, paste and run them yourself. If you are running the DB2 database product on a system that is not UNIX, ensure that any path names are in the correct format for your system. Most of the examples are based

on tables in the SAMPLE database that comes with the DB2 database product. A few examples require scenarios that are not part of the SAMPLE database, but you can use a connection to the SAMPLE database as a starting point.

Table 4. Supported table space states

State	Hexadecimal state value	Description
Backup Pending	0x20	<p>A table space is in this state after a point-in-time table space rollforward operation, or after a load operation (against a recoverable database) that specifies the COPY NO option. The table space (or, alternatively, the entire database) must be backed up before the table space can be used. If the table space is not backed up, tables within that table space can be queried, but not updated.</p> <p>Note: A database must also be backed up immediately after it is enabled for rollforward recovery. A database is recoverable if the logarchmeth1 database configuration parameter is set to any value other than OFF. You cannot activate or connect to such a database until it has been backed up, at which time the value of the backup_pending informational database configuration parameter is set to NO.</p> <p>Example</p> <p>Given the staff_data.del input file with the following content:</p> <pre>11,"Melnyk",20,"Sales",10,70000,15000;</pre> <p>Load this data into the staff table specifying the copy no as follows:</p> <pre>update db cfg for sample using logarchmeth1 logretain; backup db sample; connect to sample; load from staff_data.del of del messages load.msg insert into staff copy no; update staff set salary = 69000 where id = 11; list tablespaces; connect reset;</pre> <p>Information returned for USERSPACE1 shows that this table space is in Backup Pending state.</p>
Backup in Progress	0x800	<p>This is a transient state that is only in effect during a backup operation.</p> <p>Example</p> <p>Perform an online backup as follows:</p> <pre>backup db sample online;</pre> <p>From another session, execute one of the following scripts while the backup operation is running:</p> <ul style="list-style-type: none"> • connect to sample; <pre>list tablespaces show detail; connect reset;</pre> • connect to sample; <pre>get snapshot for tablespaces on sample; connect reset;</pre> <p>Information returned for USERSPACE1 shows that this table space is in Backup in Progress state.</p>

Table 4. Supported table space states (continued)

State	Hexadecimal state value	Description
DMS Rebalance in Progress	0x10000000	<p>This is a transient state that is only in effect during a data rebalancing operation. When new containers are added to a table space that is defined as database managed space (DMS), or existing containers are extended, a rebalancing of the table space data might occur. <i>Rebalancing</i> is the process of moving table space extents from one location to another in an attempt to keep the data striped. An <i>extent</i> is a unit of container space (measured in pages), and a stripe is a layer of extents <i>across the set of containers</i> for a table space.</p> <p>Example</p> <pre>Given the staffdata.del input file with 20000 or more records, create the table newstaff, load it using this input file, and then add a new container to table space ts1: connect to sample; create tablespace ts1 managed by database using (file '/home/melnyk/melnyk/NODE0000/SQL00001/ts1c1' 1024); create table newstaff like staff in ts1; load from staffdata.del of del insert into newstaff nonrecoverable; alter tablespace ts1 add (file '/home/melnyk/melnyk/NODE0000/SQL00001/ts1c2' 1024); list tablespaces; connect reset;</pre> <p>Information returned for TS1 shows that this table space is in DMS Rebalance in Progress state.</p>
Disable Pending	0x200	<p>A table space may be in this state during a database rollforward operation and should no longer be in this state by the end of the rollforward operation. The state is triggered by conditions that result from a table space going offline and compensation log records for a transaction not being written. The appearance and subsequent disappearance of this table space state is transparent to users.</p> <p>An example illustrating this table space state is beyond the scope of this document.</p>
Drop Pending	0x8000	<p>A table space is in this state if one or more of its containers is found to have a problem during a database restart operation. (A database must be restarted if the previous session with this database terminated abnormally, such as during a power failure, for example.) If a table space is in Drop Pending state, it will not be available, and can only be dropped.</p> <p>An example illustrating this table space state is beyond the scope of this document.</p>

Table 4. Supported table space states (continued)

State	Hexadecimal state value	Description
Load in Progress	0x20000	<p>This is a transient state that is only in effect during a load operation (against a recoverable database) that specifies the COPY NO option. See also Load in Progress table state.</p> <p>Example</p> <p>Given the staffdata.del input file with 20000 or more records, create the table newstaff and load it specifying COPY NO and this input file:</p> <pre>update db cfg for sample using logarchmeth1 logretain; backup db sample; connect to sample; create table newstaff like staff; load from staffdata.del of del insert into newstaff copy no; connect reset;</pre> <p>From another session, get information about table spaces while the load operation is running by executing one of the sample scripts shown in the Backup in Progress example.</p> <p>Information returned for USERSPACE1 shows that this table space is in Load in Progress (and Backup Pending) state.</p>
Normal	0x0	<p>A table space is in Normal state if it is not in any of the other (abnormal) table space states. Normal state is the initial state of a table space after it is created.</p> <p>Example</p> <p>Create a table space and then get information about that table space as follows:</p> <pre>connect to sample; create tablespace ts1 managed by automatic storage; list tablespaces show detail;</pre> <p>Information returned for USERSPACE1 shows that this table space is in Normal state.</p>

Table 4. Supported table space states (continued)

State	Hexadecimal state value	Description
Offline and Not Accessible	0x4000	<p>A table space is in this state if there is a problem with one or more of its containers. A container might be inadvertently renamed, moved, or damaged. After the problem has been rectified, and the containers that are associated with the table space are accessible again, this abnormal state can be removed by disconnecting all applications from the database and then reconnecting to the database. Alternatively, you can issue an ALTER TABLESPACE statement, specifying the SWITCH ONLINE clause, to remove the Offline and Not Accessible state from the table space without disconnecting other applications from the database.</p> <p>Example</p> <p>Create table space ts1 with containers tsc1 and tsc2, create table stafftemp, and import data from the st_data.del file as follows:</p> <pre>connect to sample; create tablespace ts1 managed by database using (file '/home/melnyk/melnyk/NODE0000/SQL00001/tsc1' 1024); alter tablespace ts1 add (file '/home/melnyk/melnyk/NODE0000/SQL00001/tsc2' 1024); export to st_data.del of del select * from staff; create table stafftemp like staff in ts1; import from st_data.del of del insert into stafftemp; connect reset;</pre> <p>Rename table space container tsc1 to tsc3 and then try to query the STAFFTEMP table:</p> <pre>connect to sample; select * from stafftemp;</pre> <p>The query returns SQL0290N (table space access is not allowed), and the LIST TABLESPACES command returns a state value of 0x4000 (Offline and Not Accessible) for TS1. Rename table space container tsc3 back to tsc1. This time the query runs successfully.</p>
Quiesced Exclusive	0x4	<p>A table space is in this state when the application that invokes the table space quiesce function has exclusive (read or write) access to the table space. Use the QUIESCE TABLESPACES FOR TABLE command to explicitly set a table space to Quiesced Exclusive.</p> <p>Example</p> <p>Set table spaces to Normal before setting them to Quiesced Exclusive as follows:</p> <pre>connect to sample; quiesce tablespaces for table staff reset; quiesce tablespaces for table staff exclusive; connect reset;</pre> <p>From another session, execute the following script:</p> <pre>connect to sample; select * from staff where id=60; update staff set salary=50000 where id=60; list tablespaces; connect reset;</pre> <p>Information returned for USERSPACE1 shows that this table space is in Quiesced Exclusive state.</p>

Table 4. Supported table space states (continued)

State	Hexadecimal state value	Description
Quiesced Share	0x1	<p>A table space is in this state when both the application that invokes the table space quiesce function and concurrent applications have read (but not write) access to the table space. Use the QUIESCE TABLESPACES FOR TABLE command to explicitly set a table space to Quiesced Share.</p> <p>Example</p> <p>Set table spaces to Normal before setting them to Quiesced Share as follows:</p> <pre>connect to sample; quiesce tablespaces for table staff reset; quiesce tablespaces for table staff share; connect reset;</pre> <p>From another session, execute the following script:</p> <pre>connect to sample; select * from staff where id=40; update staff set salary=50000 where id=40; list tablespaces; connect reset;</pre> <p>Information returned for USERSPACE1 shows that this table space is in Quiesced Share state.</p>
Quiesced Update	0x2	<p>A table space is in this state when the application that invokes the table space quiesce function has exclusive write access to the table space. Use the QUIESCE TABLESPACES FOR TABLE command to explicitly set a table space to Quiesced Update state.</p> <p>Example</p> <p>Set table spaces to Normal before setting them to Quiesced Update as follows:</p> <pre>connect to sample; quiesce tablespaces for table staff reset; quiesce tablespaces for table staff intent to update; connect reset;</pre> <p>From another session, execute the following script:</p> <pre>connect to sample; select * from staff where id=50; update staff set salary=50000 where id=50; list tablespaces; connect reset;</pre> <p>Information returned for USERSPACE1 shows that this table space is in Quiesced Update state.</p>

Table 4. Supported table space states (continued)

State	Hexadecimal state value	Description
Reorg in Progress	0x400	<p>This is a transient state that is only in effect during a reorg operation.</p> <p>Example</p> <p>Reorganize the staff table as follows:</p> <pre>connect to sample; reorg table staff; connect reset;</pre> <p>From another session, get information about table spaces while the reorg operation is running by executing one of the sample scripts shown in the Backup in Progress example.</p> <p>Information returned for USERSPACE1 shows that this table space is in Reorg in Progress state.</p> <p>Note: Table reorganization operations involving the SAMPLE database are likely to complete in a short period of time and, as a result, it may be difficult to observe the Reorg in Progress state using this approach.</p>
Restore Pending	0x100	<p>Table spaces for a database are in this state after the first part of a redirected restore operation (that is, before the SET TABLESPACE CONTAINERS command is issued). The table space (or the entire database) must be restored before the table space can be used. You cannot connect to the database until the restore operation has been successfully completed, at which time the value of the restore_pending informational database configuration parameter is set to NO.</p> <p>Example</p> <p>When the first part of the redirected restore operation in Storage May be Defined completes, all of the table spaces are in Restore Pending state.</p>
Restore in Progress	0x2000	<p>This is a transient state that is only in effect during a restore operation.</p> <p>Example</p> <p>Enable the sample database for rollforward recovery then back up the sample database and the USERSPACE1 table space as follows:</p> <pre>update db cfg for sample using logarchmeth1 logretain; backup db sample; backup db sample tablespace (userspace1);</pre> <p>Restore the USERSPACE1 table space backup assuming the timestamp for this backup image is 20040611174124:</p> <pre>restore db sample tablespace (userspace1) online taken at 20040611174124;</pre> <p>From another session, get information about table spaces while the restore operation is running by executing one of the sample scripts shown in the Backup in Progress example.</p> <p>Information returned for USERSPACE1 shows that this table space is in Restore in Progress state.</p>

Table 4. Supported table space states (continued)

State	Hexadecimal state value	Description
Roll Forward Pending	0x80	<p>A table space is in this state after a restore operation against a recoverable database. The table space (or the entire database) must be rolled forward before the table space can be used. A database is recoverable if the logarchmeth1 database configuration parameter is set to any value other than OFF. You cannot activate or connect to the database until a rollforward operation has been successfully completed, at which time the value of the rollfwd_pending informational database configuration parameter is set to NO.</p> <p>Example</p> <p>When the online table space restore operation in Restore in Progress completes, the table space USERSPACE1 is in Roll Forward Pending state.</p>
Roll Forward in Progress	0x40	<p>This is a transient state that is only in effect during a rollforward operation.</p> <p>Example</p> <p>Given the staffdata.del input file with 20000 or more record, create a table and tablespace followed by a database backup:</p> <pre>update db cfg for sample using logarchmeth1 logretain; backup db sample; connect to sample; create tablespace ts1 managed by automatic storage; create table newstaff like staff in ts1; connect reset; backup db sample tablespace (ts1) online;</pre> <p>Assuming that the timestamp for the backup image is 20040630000715, restore the database backup and rollforward to the end of logs as follows:</p> <pre>connect to sample; load from staffdata.del of del insert into newstaff copy yes to /home/melnyk/backups; connect reset; restore db sample tablespace (ts1) online taken at 20040630000715; rollforward db sample to end of logs and stop tablespace (ts1) online;</pre> <p>From another session, get information about table spaces while the rollforward operation is running by executing one of the sample scripts shown in the Backup in Progress example.</p> <p>Information returned for TS1 shows that this table space is in Roll Forward in Progress state.</p>
Storage May be Defined	0x2000000	<p>Table spaces for a database are in this state after the first part of a redirected restore operation (that is, before the SET TABLESPACE CONTAINERS command is issued). This allows you to redefine the containers.</p> <p>Example</p> <p>Assuming that the timestamp for the backup image is 20040613204955, restore a database backup as follows:</p> <pre>restore db sample taken at 20040613204955 redirect; list tablespaces;</pre> <p>Information returned by the LIST TABLESPACES command shows that all of the table spaces are in Storage May be Defined and Restore Pending state.</p>

Table 4. Supported table space states (continued)

State	Hexadecimal state value	Description
Storage Must be Defined	0x1000	<p>Table spaces for a database are in this state during a redirected restore operation to a new database if the set table space containers phase is omitted or if, during the set table space containers phase, the specified containers cannot be acquired. The latter can occur if, for example, an invalid path name has been specified, or there is insufficient disk space.</p> <p>Example</p> <p>Assuming that the timestamp for the backup image is 20040613204955, restore a database backup as follows:</p> <pre>restore db sample taken at 20040613204955 into mydb redirect; set tablespace containers for 2 using (path 'ts2c1'); list tablespaces;</pre> <p>Information returned by the LIST TABLESPACES command shows that table space SYSCATSPACE and table space TEMPSPACE1 are in Storage Must be Defined, Storage May be Defined, and Restore Pending state. Storage Must be Defined state takes precedence over Storage May be Defined state.</p>
Suspend Write	0x10000	<p>A table space is in this state after a write operation has been suspended.</p> <p>An example illustrating this table space state is beyond the scope of this document.</p>
Table Space Creation in Progress	0x40000000	<p>This is a transient state that is only in effect during a create table space operation.</p> <p>Example</p> <p>Create table spaces ts1, ts2, and ts3 as follows:</p> <pre>connect to sample; create tablespace ts1 managed by automatic storage; create tablespace ts2 managed by automatic storage; create tablespace ts3 managed by automatic storage;</pre> <p>From another session, get information about table spaces while the create table space operations are running by executing one of the sample scripts shown in the Backup in Progress example.</p> <p>Information returned for TS1, TS2, and TS3 shows that these table spaces are in Table Space Creation in Progress state.</p>
Table Space Deletion in Progress	0x20000000	<p>This is a transient state that is only in effect during a delete table space operation.</p> <p>Example</p> <p>Create table spaces ts1, ts2, and ts3 then drop them as follows:</p> <pre>connect to sample; create tablespace ts1 managed by automatic storage; create tablespace ts2 managed by automatic storage; create tablespace ts3 managed by automatic storage; drop tablespaces ts1, ts2, ts3;</pre> <p>From another session, get information about table spaces while the drop table space operations are running by executing one of the sample scripts shown in the Backup in Progress example.</p> <p>Information returned for TS1, TS2, and TS3 shows that these table spaces are in Table Space Deletion in Progress state.</p>

Switching table spaces from offline to online

The SWITCH ONLINE clause of the ALTER TABLESPACE statement can be used to remove the OFFLINE state from a table space if the containers associated with that table space are accessible.

Procedure

To remove the OFFLINE state from a table space using the command line, enter:

```
db2 ALTER TABLESPACE name
    SWITCH ONLINE
```

Alternatively, disconnect all applications from the database and then to have the applications connect to the database again. This removes the OFFLINE state from the table space.

Results

The table space has the OFFLINE state removed while the rest of the database is still up and being used.

Dropping table spaces

When you drop a table space, you delete all the data in that table space, free the containers, remove the catalog entries, and cause all objects defined in the table space to be either dropped or marked as invalid.

About this task

You can reuse the containers in an empty table space by dropping the table space, but you must commit the DROP TABLESPACE statement before attempting to reuse the containers.

Note: You cannot drop a table space without dropping all table spaces that are associated with it. For example, if you have a table in one table space and its index created in another table space, you must drop both index and data table spaces in one DROP TABLESPACE statement.

Procedure

- Dropping user table spaces:

You can drop a user table space that contains all of the table data including index and LOB data within that single user table space. You can also drop a user table space that might have tables spanned across several table spaces. That is, you might have table data in one table space, indexes in another, and any LOBs in a third table space. You must drop all three table spaces at the same time in a single statement. All of the table spaces that contain tables that are spanned must be part of this single statement or the drop request fails.

The following SQL statement drops the table space ACCOUNTING:

```
DROP TABLESPACE ACCOUNTING
```

- Dropping user temporary table spaces:

You can drop a user temporary table space only if there are no declared or created temporary tables currently defined in that table space. When you drop the table space, no attempt is made to drop all of the declared or created temporary tables in the table space.

Note: A declared or created temporary table is implicitly dropped when the application that declared it disconnects from the database.

- Dropping system temporary table spaces:

You cannot drop a system temporary table space that has a page size of 4 KB without first creating another system temporary table space. The new system temporary table space must have a page size of 4 KB because the database must always have at least one system temporary table space that has a page size of 4 KB. For example, if you have a single system temporary table space with a page size of 4 KB, and you want to add a container to it, and it is an SMS table space, you must first add a new 4 KB page size system temporary table space with the proper number of containers, and then drop the old system temporary table space. (If you are using DMS, you can add a container without needing to drop and re-create the table space.)

The default table space page size is the page size that the database was created with (which is 4 KB by default, but can also be 8 KB, 16 KB, or 32 KB).

1. To create a system temporary table space, issue the statement:

```
CREATE SYSTEM TEMPORARY TABLESPACE name
    MANAGED BY SYSTEM USING ('directories')
```

2. Then, to drop a system table space using the command line, enter:

```
DROP TABLESPACE name
```

3. The following SQL statement creates a system temporary table space called TEMPSPACE2:

```
CREATE SYSTEM TEMPORARY TABLESPACE TEMPSPACE2
    MANAGED BY SYSTEM USING ('d:\systemp2')
```

4. After TEMPSPACE2 is created, you can drop the original system temporary table space TEMPSPACE1 with the statement:

```
DROP TABLESPACE TEMPSPACE1
```

Chapter 3. Buffer pools

A *buffer pool* is an area of main memory that has been allocated by the database manager for the purpose of caching table and index data as it is read from disk. Every DB2 database must have a buffer pool.

Each new database has a default buffer pool defined, called IBMDEFAULTBP. Additional buffer pools can be created, dropped, and modified, using the CREATE BUFFERPOOL, DROP BUFFERPOOL, and ALTER BUFFERPOOL statements. The SYSCAT.BUFFERPOOLS catalog view accesses the information for the buffer pools defined in the database.

In a DB2 pureScale environment, each member has its own local buffer pool (LBP). However there is an additional group buffer pool (GBP) that is maintained by the cluster caching facility. The GBP is shared by all members. It is used as a cache for pages used by individual members across a DB2 pureScale instance to improve performance and ensure consistency.

How buffer pools are used

Note: The information that follows discusses buffer pools in environments other than DB2 pureScale environments. Buffer pools work differently in DB2 pureScale environments. For more information, see "Buffer pool monitoring in a DB2 pureScale environment", in the *Database Monitoring Guide and Reference*.

When a row of data in a table is first accessed, the database manager places the page that contains that data into a buffer pool. Pages stay in the buffer pool until the database is shut down or until the space occupied by the page is required by another page.

Pages in the buffer pool can be either in-use or not, and they can be dirty or clean:

- In-use pages are currently being read or updated. To maintain data consistency, the database manager only allows one agent to be updating a given page in a buffer pool at one time. If a page is being updated, it is being accessed exclusively by one agent. If it is being read, it might be read by multiple agents simultaneously.
- "Dirty" pages contain data that has been changed but has not yet been written to disk.
- After a changed page is written to disk, it is clean and might remain in the buffer pool.

A large part of tuning a database involves setting the configuration parameters that control the movement of data into the buffer pool and the writing of data from the buffer out to disk. If not needed by a recent agent, the page space can be used for new page requests from new applications. Database manager performance is degraded by extra disk I/O.

Designing buffer pools

The sizes of all buffer pools can have a major impact on the performance of your database.

Before you create a new buffer pool, resolve the following items:

- What buffer pool name do you want to use?
- Whether the buffer pool is to be created immediately or following the next time that the database is deactivated and reactivated?
- Whether the buffer pool should exist for all database partitions, or for a subset of the database partitions?
- What page size you want for the buffer pool? See “Buffer pool page sizes”.
- Whether the buffer pool will be a fixed size, or whether the database manager will automatically adjust the size of the buffer pool in response to your workload? It is suggested that you allow the database manager to tune your buffer pool automatically by leaving the SIZE parameter unspecified during buffer pool creation. For details, see the SIZE parameter of the “CREATE BUFFERPOOL statement” and “Buffer pool memory considerations” on page 57.
- Whether you want to reserve a portion of the buffer pool for block based I/O? For details, see: “Block-based buffer pools for improved sequential prefetching”.

Relationship between table spaces and buffer pools

When designing buffer pools, you must understand the relationship between table spaces and buffer pools. Each table space is associated with a specific buffer pool. IBMDEFAULTBP is the default buffer pool. The database manager also allocates these system buffer pools: IBMSYSTEMBP4K, IBMSYSTEMBP8K, IBMSYSTEMBP16K, and IBMSYSTEMBP32K (formerly known as the “hidden buffer pools”). To associate another buffer pool with a table space, the buffer pool must exist and the two must have the same page size. The association is defined when the table space is created (using the CREATE TABLESPACE statement), but it can be changed at a later time (using the ALTER TABLESPACE statement).

Having more than one buffer pool allows you to configure the memory used by the database to improve overall performance. For example, if you have a table space with one or more large (larger than available memory) tables that are accessed randomly by users, the size of the buffer pool can be limited, because caching the data pages might not be beneficial. The table space for an online transaction application might be associated with a larger buffer pool, so that the data pages used by the application can be cached longer, resulting in faster response times. Care must be taken in configuring new buffer pools.

Buffer pool page sizes

The page size for the default buffer pool is set when you use the **CREATE DATABASE** command. This default represents the default page size for all future CREATE BUFFERPOOL and CREATE TABLESPACE statements. If you do not specify the page size when creating the database, the default page size is 4 KB.

Note: If you have determined that a page size of 8 KB, 16 KB, or 32 KB is required by your database, you must have at least one buffer pool of the matching page size defined and associated with table space in your database.

However, you might need a buffer pool that has different characteristics than the system buffer pool. You can create new buffer pools for the database manager to use. You might have to restart the database for table space and buffer pool changes to take effect. The page sizes that you specify for your table spaces should determine the page sizes that you choose for your buffer pools. The choice of page size used for a buffer pool is important because you cannot alter the page size after you create a buffer pool.

Buffer pool memory considerations

Memory requirements

When designing buffer pools, you should also consider the memory requirements based on the amount of installed memory on your computer and the memory required by other applications running concurrently with the database manager on the same computer. Operating system data swapping occurs when there is insufficient memory to hold all the data being accessed. This occurs when some data is written or swapped to temporary disk storage to make room for other data. When the data on temporary disk storage is needed, it is swapped back into main memory.

Buffer pool memory protection

With Version 9.5, data pages in buffer pool memory are protected using storage keys, which are available only if explicitly enabled by the DB2_MEMORY_PROTECT registry variable, and only on AIX (5.3 TL06 5.4), running on POWER6®.

Buffer pool memory protection works on a per-agent level; any particular agent will only have access to buffer pool pages when that agent needs access. Memory protection works by identifying at which times the DB2 engine threads should have access to the buffer pool memory and at which times they should not have access. For details, see: "Buffer pool memory protection (AIX running on POWER6)."

Address Windowing Extensions (AWE) and Extended Storage (ESTORE)

Note: AWE and ESTORE features have been discontinued, including the ESTORE-related keywords, monitor elements, and data structures. To allocate more memory, you must upgrade to a 64-bit hardware operating system, and associated DB2 products. You should also modify applications and scripts to remove references to this discontinued functionality.

Buffer pool hit ratios

Buffer pool hit ratios reflect the extent to which data needed for queries is found in memory, as opposed to having to be read in from external storage. You can calculate hit rates and ratios with formulas that are based on buffer pool monitor elements. For more information, see "Formulas for calculating buffer pool hit ratios" at the following URL: <http://publib.boulder.ibm.com/infocenter/db2luw/v10r1/topic/com.ibm.db2.luw.admin.mon.doc/doc/r0056871.html>

Buffer pool memory protection (AIX running on POWER6)

The database manager uses the buffer pool to apply additions, modifications, and deletions to much of the database data.

Storage keys is a new feature in IBM Power6 processors and the AIX operating system that allows the protection of ranges of memory using hardware keys at a

kernel thread level. Storage key protection reduces buffer pool memory corruption problems and limits errors that might halt the database. Attempts to illegally access the buffer pool by programming means cause an error condition that the database manager can detect and deal with.

Note: Buffer pool memory protection works on a per-agent level; any particular agent has access to buffer pool pages only when that agent needs access.

The database manager protects buffer pools by restricting access to buffer pool memory. When an agent requires access to the buffer pools to perform its work, it is temporarily granted access to the buffer pool memory. When the agent no longer requires access to the buffer pools, access is revoked. This behavior ensures that agents are only allowed to modify buffer pool contents when needed, reducing the likelihood of buffer pool corruptions. Any illegal access to buffer pool memory results in a segmentation error. Tools to diagnose these errors are provided, such as the **db2diag**, **db2fodc**, **db2pdcfg**, and **db2support** commands.

To enable the buffer pool memory protection feature, in order to increase the resilience of the database engine, enable the **DB2_MEMORY_PROTECT** registry variable:

DB2_MEMORY_PROTECT registry variable

This registry variable enables and disables the buffer pool memory protection feature. When **DB2_MEMORY_PROTECT** is enabled (set to YES), and a DB2 engine thread tries to illegally access buffer pool memory, that engine thread traps. The default is NO.

Note: The buffer pool memory protection feature depends on the implementation of AIX Storage Protect Keys and it might not work with the pinned shared memory. If **DB2_MEMORY_PROTECT** is specified with **DB2_PINNED_BP** or **DB2_LARGE_PAGE_MEM** setting, AIX Storage Protect Keys may not be enabled. For more information about AIX Storage Protect Keys, see http://publib.boulder.ibm.com/infocenter/systems/scope/aix/index.jsp?topic=/com.ibm.aix.genprogc/doc/genprogc/storage_protect_keys.htm.

You cannot use the memory protection if **DB2_LGPAGE_BP** is set to YES. Even if **DB2_MEMORY_PROTECT** is set to YES, DB2 database manager will fail to protect the buffer pool memory and disable the feature.

Buffer pool monitoring in a DB2 pureScale environment

Examining the number of times that pages of data requested by a member can be found in group or local buffer pools, as opposed to the number of times they need to be read in from disk can tell you where you might have performance problems related to I/O.

Generally speaking, larger buffer pools increase the likelihood that a required page of data can be found in memory.

Viewing and comparing monitor elements related to buffer pool activity can help you understand the extent to which the group buffer pool (GBP) in the cluster caching facility, and the local buffer pools (LBPs) for each member are reducing the amount of disk I/O in your system.

Buffer pool hit rates and hit ratios in a DB2 pureScale environment

One way of measuring the extent to which pages required by members are found in memory as opposed to on disk is by calculating the buffer pool *hit ratio*. The buffer pool hit ratio indicates the number of times that the database manager found a requested page in a buffer pool (also known as the *hit rate*) as compared to the number of times it had to read it from disk. In a DB2 pureScale environment, both the local buffer pool and group buffer pool hit rates and hit ratios are important factors in assessing overall performance.

Local buffer pool (LBP) hit ratios reflect the extent to which pages that a member needs can be found in a valid state in the local buffer pool. A page in the LBP of a member is deemed to be in a valid state if that page has not been changed by another member since it was loaded into the LBP. If another member has changed the page, which might happen before the page has been cast out to disk, then the page is said to be *invalid*. If the member with the invalid page requires that page to perform a transaction, the member has to go to the CF to request a new, valid version of the page.

A low LBP hit ratio is an indication that the pages were not found locally, and had to be requested from the CF.

However, in a DB2 pureScale environment, looking at the LBP hit ratios provides only one side of the buffer pool story. You also need to consider the role that the group buffer pool (GBP) plays in retrieving pages, and the hit ratio for the GBP itself. If a member is unable to locate a valid copy of a page in its LBP, it makes a request to the CF to search the GBP for a valid copy of the page. The GBP does one of the following actions:

- If it has a valid copy of the page, the GBP provides it to the member making the request.
- Otherwise, the GBP tells the requesting member that it must read the page from disk.

An additional consideration for LBP usage is the concept of GBP-independent page. A GBP-independent page is a page that is only ever accessed through a LBP of a member, and never exists in the GBP. Pages might be GBP-independent because the operations using the page, or the objects where the pages come from, are only accessed by the local member.

Group buffer pool hit ratios reflect the extent to which pages required by members, for which they do not have a valid local copy, are found in the group buffer pool, as compared to having to be read in from disk. A low hit ratio for the GBP is an indication that relatively few of the pages required by members across the instance are available in the GBP. Increasing the size of the GBP can improve hit rates, and overall performance. Therefore, when calculating the hit ratios for data pages in the local buffer pool (LBP) for a member, you need to consider the number of times the member attempted to read pages from the LBP in comparison to the number of times attempted reads did not find a valid page in the LBP. See “Formulas for calculating buffer pool hit ratios” on page 61 for details on how LBP and GBP monitor elements are used to calculate the GBP hit rate.

Tip: Hit ratios can vary based on many factors, such as the nature of the data in your database, the queries that are run against it, as well as hardware and software configurations. Generally speaking, higher buffer pool hit ratios are reflective of better query performance. If you find hit ratios seem low, or are declining over

time, increasing the size of the buffer pools can help. To increase the size of the group buffer pool, adjust the **cf_gbp_sz** configuration parameter on the CF. To adjust local buffer pools, run the **ALTER BUFFERPOOL** statement on the member with the buffer pools that need correction.

Buffer pool monitor element reporting

In DB2 pureScale environments, as is the case with other DB2 environments, each member reports on its own local buffer pools. No aggregation of data across members takes place. You must take into account which member or members you are interested in, and interpret the data accordingly. In some cases, you might want to calculate the hit ratios for a specific member. In others cases, you might want to look at the data for all members together, to form an overall view of the hit rates and hit ratios for the DB2 pureScale environment as a whole.

For example, if you submit a query to return data for the number of times a data page was read into a local buffer pool from disk, because it was not found in the GBP (using the **pool_data_gbp_p_reads** monitor element) with the **MON_GET_BUFFERPOOL** table function, and you do not specify which member to return, you will see results like the ones that follow:

MEMBER BP_NAME	POOL_DATA_GBP_P_READS
0 IBMDEFAULTBP	408
0 IBMSYSTEMBP4K	0
0 IBMSYSTEMBP8K	0
0 IBMSYSTEMBP16K	0
0 IBMSYSTEMBP32K	0
1 IBMDEFAULTBP	108
1 IBMSYSTEMBP4K	0
1 IBMSYSTEMBP8K	0
1 IBMSYSTEMBP16K	0
1 IBMSYSTEMBP32K	0
2 IBMDEFAULTBP	112
2 IBMSYSTEMBP4K	0
2 IBMSYSTEMBP8K	0
2 IBMSYSTEMBP16K	0
2 IBMSYSTEMBP32K	0

15 record(s) selected.

Important: In the preceding example, you can see that the data reported for temporary buffer pools shows all zeros. This is not a coincidence; in DB2 pureScale instances, temporary objects and table spaces are local to the member they are associated with. They do not use the GBP on the CF.

If you are interested in the results across all members, you can use the SUM aggregate function to add the numbers for all members together:

```
SELECT VARCHAR(BP_NAME,15) AS BP_NAME,
       SUM(POOL_DATA_GBP_P_READS) AS TOTAL_P_READS
  FROM TABLE(MON_GET_BUFFERPOOL(' ', -2))
 GROUP BY BP_NAME
```

The preceding query returns results like the following output:

BP_NAME	TOTAL_P_READS
IBMDEFAULTBP	310
IBMSYSTEMBP16K	0
IBMSYSTEMBP32K	0
IBMSYSTEMBP4K	0
IBMSYSTEMBP8K	0

5 record(s) selected.

Formulas for calculating buffer pool hit ratios

Buffer pool hit ratios reflect the extent to which data needed for queries is found in memory, as opposed to having to be read in from external storage. You can calculate hit rates and ratios with formulas that are based on buffer pool monitor elements.

Local buffer pools

Table 5. Formulas for local buffer pool hit ratios. The formulas shown express the hit ratios as a percentage.

Type of page	Formula for calculating buffer pool hit ratio
Data pages	$((\text{pool_data_lbp_pages_found} - \text{pool_async_data_lbp_pages_found}) / (\text{pool_data_l_reads} + \text{pool_temp_data_l_reads})) \times 100$
Index pages	$((\text{pool_index_lbp_pages_found} - \text{pool_async_index_lbp_pages_found}) / (\text{pool_index_l_reads} + \text{pool_temp_index_l_reads})) \times 100$
XML storage object (XDA) pages	$((\text{pool_xda_lbp_pages_found} - \text{pool_async_xda_lbp_pages_found}) / (\text{pool_xda_l_reads} + \text{pool_temp_xda_l_reads})) \times 100$
Overall hit ratio	$((\text{pool_data_lbp_pages_found} + \text{pool_index_lbp_pages_found} + \text{pool_xda_lbp_pages_found} - \text{pool_async_data_lbp_pages_found} - \text{pool_async_index_lbp_pages_found} - \text{pool_async_xda_lbp_pages_found}) / (\text{pool_data_l_reads} + \text{pool_index_l_reads} + \text{pool_xda_l_reads} + \text{pool_temp_data_l_reads} + \text{pool_temp_xda_l_reads} + \text{pool_temp_index_l_reads})) \times 100$

Group buffer pools (DB2 pureScale environments)

The formulas used to calculate group buffer pool hit ratios in a DB2 pureScale environment are different from formulas for hit ratios used in other DB2 environments. This difference is because of how the group buffer pool in the cluster caching facility works with the local buffer pools in each member to retrieve pages of data. The following formulas, which are based on buffer pool monitor elements, can be used to calculate hit ratios for data, index, and XML storage object pages, for both the local and group buffer pools.

Table 6. Formulas for group buffer pool (GBP) hit ratios. The formulas shown express the hit ratios as a percentage.

Type of page	Formula for calculating buffer pool hit ratio
Data pages	$((\text{pool_data_gbp_l_reads} - \text{pool_data_gbp_p_reads}) / \text{pool_data_gbp_l_reads}) \times 100$
Index pages	$((\text{pool_index_gbp_l_reads} - \text{pool_index_gbp_p_reads}) / \text{pool_index_gbp_l_reads}) \times 100$
XML storage object (XDA) pages	$((\text{pool_xda_gbp_l_reads} - \text{pool_xda_gbp_p_reads}) / \text{pool_xda_gbp_l_reads}) \times 100$
Overall hit ratio	$((\text{pool_data_gbp_l_reads} + \text{pool_index_gbp_l_reads} + \text{pool_xda_gbp_l_reads} - \text{pool_data_gbp_p_reads} - \text{pool_index_gbp_p_reads} - \text{pool_xda_gbp_p_reads}) / (\text{pool_data_gbp_l_reads} + \text{pool_index_gbp_l_reads} + \text{pool_xda_gbp_l_reads})) \times 100$

In addition to the preceding formulas for calculating buffer pool hit ratios, you can also use the following formulas to show what percentage of the time pages that are prefetched are found in the GBP:

Prefetches for data pages

$$((\text{pool_async_data_gbp_l_reads} - \text{pool_async_data_gbp_p_reads}) / \text{pool_async_data_gbp_l_reads}) \times 100$$

Prefetches for index pages

$$((\text{pool_async_index_gbp_l_reads} - \text{pool_async_index_gbp_p_reads}) / \text{pool_async_index_gbp_l_reads}) \times 100$$

Prefetches for XML storage object (XDA) pages

$$\frac{((\text{pool_async_xda_gbp_l_reads} - \text{pool_async_xda_gbp_p_reads}) / \text{pool_async_xda_gbp_l_reads}) \times 100}{}$$

Calculating buffer pool hit ratios in a DB2 pureScale environment

Calculating buffer pool hit ratios for a DB2 pureScale instance can help you understand where there are opportunities to tune buffer pools to improve I/O efficiency.

Before you begin

Determine which ratio or ratios you are interested in. If you want to see a ratio across all members in an instance, consider formulating your SQL to aggregate data across members using the SUM aggregate function. If you are interested in seeing the data for a specific member only, you can use specify the member for which you want to see data in the MON_GET_BUFFERPOOL table function.

Procedure

To calculate buffer pool hit ratios, follow these steps:

1. Retrieve the information for the required monitor elements. This example uses the MON_GET_BUFFERPOOL table function to retrieve the monitor elements that contain the values needed to calculate the hit ratio for data pages for the GBP, **pool_data_gbp_l_reads** and **pool_data_gbp_p_reads**.

```
SELECT varchar(bp_name,20) AS bp_name,
       pool_data_gbp_l_reads,
       pool_data_gbp_p_reads,
       member
  FROM TABLE(MON_GET_BUFFERPOOL('','-2))
```

The preceding query returns data like the following example:

BP_NAME	POOL_DATA_GBP_L_READS	POOL_DATA_GBP_P_READS	MEMBER
IBMDFAULTBP	1814911	456990	1
IBMSYSTEMBP4K	0	0	1
IBMSYSTEMBP8K	0	0	1
IBMSYSTEMBP16K	0	0	1
IBMSYSTEMBP32K	0	0	1
IBMDFAULTBP	1807959	455287	3
IBMSYSTEMBP4K	0	0	3
IBMSYSTEMBP8K	0	0	3
IBMSYSTEMBP16K	0	0	3
IBMSYSTEMBP32K	0	0	3
IBMDFAULTBP	1813932	455225	2
IBMSYSTEMBP4K	0	0	2
IBMSYSTEMBP8K	0	0	2
IBMSYSTEMBP16K	0	0	2
IBMSYSTEMBP32K	0	0	2
IBMDFAULTBP	1113396	278845	0
IBMSYSTEMBP4K	0	0	0
IBMSYSTEMBP8K	0	0	0
IBMSYSTEMBP16K	0	0	0
IBMSYSTEMBP32K	0	0	0

20 record(s) selected.

Important: In the preceding example, you can see that the data reported for temporary buffer pools shows all zeros. This is not a coincidence; in DB2

pureScale instances, temporary objects and table spaces are local to the member they are associated with. They do not use the GBP on the CF.

2. Use the values returned for the monitor elements to calculate the hit ratio. The formula for calculating the hit ratio for the GBP (expressed as a percentage) is

$$((pool_data_gbp_l_reads - pool_data_gbp_p_reads) \div pool_data_gbp_l_reads) \times 100$$

So, using the data returned for the monitor elements in step 1 on page 62:

$$\begin{aligned} & (((1,814,911+1,807,959 + 1,813,932+1,113,396) - (456,990+455,287 + 455,225+278,845)) \\ & \quad \div (1,814,911+1,807,959 + 1,813,932+1,113,396)) \times 100 \\ & = ((6,550,198 - 1,646,347) \div 6,550,198) \times 100 \\ & = 74.9\% \end{aligned}$$

In this example, the hit ratio for the GBP is 74.9%

Note: The values shown in the output for queries are for illustrative purposes only.

Example

Example 1: Find the overall hit rates across all members

This example is similar to the one shown in the preceding procedure, except that it uses an aggregate function to provide overall hit rates across all members.

```
SELECT VARCHAR(BP_NAME,20) AS BP,
       SUM(POOL_DATA_GBP_L_READS) AS POOL_DATA_GBP_L_READS,
       SUM(POOL_DATA_GBP_P_READS) AS POOL_DATA_GBP_P_READS
  FROM TABLE(MON_GET_BUFFERPOOL('','-2))
 GROUP BY BP_NAME
```

Results:

BP	POOL_DATA_GBP_L_READS	POOL_DATA_GBP_P_READS
IBMDEFAULTBP	6550198	1646347
IBMSYSTEMBP16K	0	0
IBMSYSTEMBP32K	0	0
IBMSYSTEMBP4K	0	0
IBMSYSTEMBP8K	0	0

5 record(s) selected.

Example 2: Determining the GBP hit ratio for all data, index, and XML storage object (XDA) pages

To calculate the GBP hit ratio for all data, index, and XDA pages, use the following formula:

$$\begin{aligned} & ((pool_data_gbp_l_reads + pool_index_gbp_l_reads+pool_xda_gbp_l_reads) \\ & \quad - (pool_data_gbp_p_reads + pool_index_gbp_p_reads+pool_xda_gbp_p_reads)) \\ & \quad \div (pool_data_gbp_l_reads + pool_index_gbp_l_reads+pool_xda_gbp_l_reads)) \times 100 \end{aligned}$$

The following example uses the **MON_GET_BUFFERPOOL** table function to retrieve the data contained in the required monitor elements and calculates the hit ratio for each member:

```
WITH BPMETRICS AS (
  SELECT BP_NAME,
         POOL_DATA_GBP_L_READS +
         POOL_INDEX_GBP_L_READS +
         POOL_XDA_GBP_L_READS
```

```

        AS LOGICAL_READS,
        POOL_DATA_GBP_P_READS +
        POOL_INDEX_GBP_P_READS +
        POOL_XDA_GBP_P_READS
        AS PHYSICAL_READS,
        MEMBER
FROM TABLE(MON_GET_BUFFERPOOL('','-2)) AS METRICS)
SELECT VARCHAR(BP_NAME,20) AS BP_NAME,
       LOGICAL_READS,
       PHYSICAL_READS,
CASE WHEN LOGICAL_READS > 0
    THEN DEC((
        FLOAT(LOGICAL_READS) - FLOAT(PHYSICAL_READS)) /
        FLOAT(LOGICAL_READS))
        * 100,5,2)
    ELSE NULL END AS HIT_RATIO,
        MEMBER
FROM BPMETRICS

```

Results:

BP_NAME	LOGICAL_READS	PHYSICAL_READS	HIT_RATIO	MEMBER
IBMDFAULTBP	5730213	617628	89.22	1
IBMSYSTEMBP4K	0	0	-	1
IBMSYSTEMBP8K	0	0	-	1
IBMSYSTEMBP16K	0	0	-	1
IBMSYSTEMBP32K	0	0	-	1
IBMDFAULTBP	5724845	615395	89.25	3
IBMSYSTEMBP4K	0	0	-	3
IBMSYSTEMBP8K	0	0	-	3
IBMSYSTEMBP16K	0	0	-	3
IBMSYSTEMBP32K	0	0	-	3
IBMDFAULTBP	5731714	615814	89.25	2
IBMSYSTEMBP4K	0	0	-	2
IBMSYSTEMBP8K	0	0	-	2
IBMSYSTEMBP16K	0	0	-	2
IBMSYSTEMBP32K	0	0	-	2
IBMDFAULTBP	5024809	409159	91.85	0
IBMSYSTEMBP4K	0	0	-	0
IBMSYSTEMBP8K	0	0	-	0
IBMSYSTEMBP16K	0	0	-	0
IBMSYSTEMBP32K	0	0	-	0

20 record(s) selected.

Example 3: Using the SUM aggregate function to compute an overall hit ratio

You can also use the SUM aggregate function to compute an overall hit ratio across all members as follows:

```

WITH BPMETRICS AS (
    SELECT SUM(POOL_DATA_GBP_L_READS) +
           SUM(POOL_INDEX_GBP_L_READS) +
           SUM(POOL_XDA_GBP_L_READS)
        AS LOGICAL_READS,
           SUM(POOL_DATA_GBP_P_READS) +
           SUM(POOL_INDEX_GBP_P_READS) +
           SUM(POOL_XDA_GBP_P_READS)
        AS PHYSICAL_READS
FROM TABLE(MON_GET_BUFFERPOOL('','-2)) AS METRICS)
SELECT LOGICAL_READS,
       PHYSICAL_READS,
CASE WHEN LOGICAL_READS > 0
    THEN DEC(((FLOAT(LOGICAL_READS) - FLOAT(PHYSICAL_READS)) /
        FLOAT(LOGICAL_READS))
        * 100,5,2)
    ELSE NULL END AS HIT_RATIO
FROM BPMETRICS

```

```

Results:
LOGICAL_READS      PHYSICAL_READS      HIT_RATIO
-----
22211581          2255996           89.84
1 record(s) selected.

```

What to do next

If hit ratios seem low, or if they decline over time, you might want to increase the size of the buffer pools on either members, CFs, or both. If you are seeing lower than expected hit rates for the LBPs overall across the DB2 pureScale instance, look at the hit rates for each member individually, since the buffer pools on each member can have different sizes. A smaller sized LBP on one member might be unduly influencing the average hit rate for the instance.

Tip: Hit ratios can vary based on many factors, such as the nature of the data in your database, the queries that are run against it, as well as hardware and software configurations. Generally speaking, higher buffer pool hit ratios are reflective of better query performance. If you find hit ratios seem low, or are declining over time, increasing the size of the buffer pools can help. To increase the size of the group buffer pool, adjust the **cf_gbp_sz** configuration parameter on the CF. To adjust local buffer pools, run the **ALTER BUFFERPOOL** statement on the member with the buffer pools that need correction.

Creating buffer pools

Use the CREATE BUFFERPOOL statement to define a new buffer pool to be used by the database manager.

Before you begin

There needs to be enough real memory on the computer for the total of all the buffer pools that you created. The operating system also needs some memory to operate.

About this task

On partitioned databases, you can also define the buffer pool to be created differently, including different sizes, on each database partition. The default ALL DBPARTITIONNUMS clause creates the buffer pool on all database partitions in the database.

Procedure

To create a buffer pool using the command line:

1. Get the list of buffer pool names that exist in the database. Issue the following SQL statement:

```
SELECT BPNAME FROM SYSCAT.BUFFERPOOLS
```
2. Choose a buffer pool name that is not currently found in the result list.
3. Determine the characteristics of the buffer pool you are going to create.
4. Ensure that you have the correct authorization ID to run the CREATE BUFFERPOOL statement.
5. Issue the CREATE BUFFERPOOL statement. A basic CREATE BUFFERPOOL statement is:

```
CREATE BUFFERPOOL buffer-pool-name
  PAGESIZE 4096
```

Results

If there is sufficient memory available, the buffer pool can become active immediately. By default new buffer pools are created using the IMMEDIATE keyword, and on most platforms, the database manager is able to acquire more memory. The expected return is successful memory allocation. In cases where the database manager is unable to allocate the extra memory, the database manager returns a warning condition stating that the buffer pool could not be started. This warning is provided on the subsequent database startup. For immediate requests, you do not need to restart the database. When this statement is committed, the buffer pool is reflected in the system catalog tables, but the buffer pool does not become active until the next time the database is started. For more information about this statement, including other options, see the “CREATE BUFFERPOOL statement”.

If you issue a CREATE BUFFERPOOL DEFERRED, the buffer pool is not immediately activated; instead, it is created at the next database startup. Until the database is restarted, any new table spaces use an existing buffer pool, even if that table space is created to explicitly use the deferred buffer pool.

Example

In the following example, the optional DATABASE PARTITION GROUP clause identifies the database partition group or groups to which the buffer pool definition applies:

```
CREATE BUFFERPOOL buffer-pool-name
  PAGESIZE 4096
  DATABASE PARTITION GROUP db-partition-group-name
```

If this parameter is specified, the buffer pool is created only on database partitions in these database partition groups. Each database partition group must currently exist in the database. If the DATABASE PARTITION GROUP clause is not specified, this buffer pool is created on all database partitions (and on any database partitions that are later added to the database).

For more information, see the “CREATE BUFFERPOOL statement”.

Modifying buffer pools

There are a number of reasons why you might want to modify a buffer pool, for example, to enable self-tuning memory. To do this, you use the ALTER BUFFERPOOL statement.

Before you begin

The authorization ID of the statement must have SYSCTRL or SYSADM authority.

About this task

When working with buffer pools, you might need to do one of the following tasks:

- Enable self tuning for a buffer pool, allowing the database manager to adjust the size of the buffer pool in response to your workload.
- Modify the block area of the buffer pool for block-based I/O.

- Add this buffer pool definition to a new database partition group.
- Modify the size of the buffer pool on some or all database partitions.

To alter a buffer pool using the command line, do the following:

1. To get the list of the buffer pool names that already exist in the database, issue the following statement:

```
SELECT BPNAME FROM SYSCAT.BUFFERPOOLS
```

2. Choose the buffer pool name from the result list.
3. Determine what changes must be made.
4. Ensure that you have the correct authorization ID to run the ALTER BUFFERPOOL statement.

Note: Two key parameters are IMMEDIATE and DEFERRED. With IMMEDIATE, the buffer pool size is changed without having to wait until the next database activation for it to take effect. If there is insufficient database shared memory to allocate new space, the statement is run as DEFERRED.

With DEFERRED, the changes to the buffer pool will not be applied until the database is reactivated. Reserved memory space is not needed; the database manager allocates the required memory from the system at activation time.

5. Use the ALTER BUFFERPOOL statement to alter a single attribute of the buffer pool object. For example:

```
ALTER BUFFERPOOL buffer pool name SIZE number of pages
```

- The *buffer pool name* is a one-part name that identifies a buffer pool described in the system catalogs.
- The *number of pages* is the new number of pages to be allocated to this specific buffer pool. You can also use a value of -1, which indicates that the size of the buffer pool should be the value found in the **buffpage** database configuration parameter.

The statement can also have the DBPARTITIONNUM <db partition number> clause that specifies the database partition on which the size of the buffer pool is modified. If this clause is not specified, the size of the buffer pool is modified on all database partitions except those that have an exception entry in SYSCAT.BUFFERPOOLDBPARTITIONS. For details on using this clause for database partitions, see the ALTER BUFFERPOOL statement.

Changes to the buffer pool as a result of this statement are reflected in the system catalog tables when the statement is committed. However, no changes to the actual buffer pool take effect until the next time the database is started, except for successful ALTER BUFFERPOOL requests specified with the default IMMEDIATE keyword.

There must be enough real memory on the computer for the total of all the buffer pools that you have created. There also needs to be sufficient real memory for the rest of the database manager and for your applications.

Dropping buffer pools

When dropping buffer pools, ensure that no table spaces are assigned to those buffer pools.

You cannot drop the IBMDEFAULTBP buffer pool.

About this task

Disk storage might not be released until the next connection to the database. Storage memory is not released from a dropped buffer pool until the database is stopped. Buffer pool memory is released immediately, to be used by the database manager.

Procedure

To drop buffer pools, use the DROP BUFFERPOOL statement.

```
DROP BUFFERPOOL buffer-pool-name
```

Chapter 4. Storage groups

A storage group is a named set of storage paths where data can be stored. Storage groups are configured to represent different classes of storage available to your database system. You can assign table spaces to the storage group that best suits the data. Only automatic storage table spaces use storage groups.

A table space can be associated with only one storage group, but a storage group can have multiple table space associations. To manage storage group objects you can use the CREATE STOGROUP, ALTER STOGROUP, RENAME STOGROUP, DROP and COMMENT statements.

With the table partitioning feature, you can place table data in multiple table spaces. Using this feature, storage groups can store a subset of table data on fast storage while the remainder of the data is on one or more layers of slower storage. Use storage groups to support multi-temperature storage which prioritizes data based on classes of storage. For example, you can create storage groups that map to the different tiers of storage in your database system. Then the defined table spaces are associated with these storage groups.

When defining storage groups, ensure that you group the storage paths according to their quality of service characteristics. The common *quality of service* characteristics for data follow an aging pattern where the most recent data is frequently accessed and requires the fastest access time (*hot data*) while older data is less frequently accessed and can tolerate higher access time (*warm data* or *cold data*). The priority of the data is based on:

- Frequency of access
- Acceptable access time
- Volatility of the data
- Application requirements

Typically, the priority of data is inversely proportional to the volume, where there is significantly more cold and warm data and only a small portion of data is hot. You can use the DB2 Work Load Manager (WLM) to define rules about how activities are treated based on a tag that can be assigned to accessed data through the definition of a table space or a storage group.

Default storage groups

If a database has storage groups, the default storage group is used when an automatic storage managed table space is created without explicitly specifying the storage group.

When you create a database, a default storage group named IBMSTOGROUP is automatically created. However, a database created with the AUTOMATIC STORAGE NO clause, does not have a default storage group. The first storage group created with the CREATE STOGROUP statement becomes the designated default storage group. There can only be one storage group designated as the default storage group.

Note: Although, you can create a database specifying the AUTOMATIC STORAGE NO clause, the AUTOMATIC STORAGE clause is deprecated and might be removed from a future release.

You can designate a default storage group by using either the CREATE STOGROUP or ALTER STOGROUP statements. When you designate a different storage group as the default storage group, there is no impact to the existing table spaces using the old default storage group. To alter the storage group associated with a table space, use the ALTER TABLESPACE statement.

You can determine which storage group is the default storage group by using the SYSCAT.STOGROUPS catalog view.

You cannot drop the current default storage group. You can drop the IBMSTOGROUP storage group if it is not designated as the default storage group at that time. If you drop the IBMSTOGROUP storage group, you can create another storage group with that name.

Storage group and table space media attributes

Automatic storage table spaces inherit media attribute values, device read rate and data tag attributes, from the storage group that the table spaces are using by default.

When you create a storage group by using the CREATE STOGROUP statement, you can specify the following storage group attributes:

OVERHEAD

This attribute specifies the I/O controller time and the disk seek and latency time in milliseconds.

DEVICE READ RATE

This attribute specifies the device specification for the read transfer rate in megabytes per second. This value is used to determine the cost of I/O during query optimization. If this value is not the same for all storage paths, the number should be the average for all storage paths that belong to the storage group.

DATA TAG

This attribute specifies a tag on the data in a particular storage group, which WLM can use to determine the processing priority of database activities.

The default values for the storage group attributes are as follows:

Table 7. The default settings for storage group attributes

Attribute	Default setting
DATA TAG	NONE
DEVICE READ RATE	100 MB/sec
OVERHEAD	6.725 ms

When creating an automatic storage table space, you can specify a tag that identifies data contained in that table space. If that table space is associated with a storage group, then the data tag attribute on the table space overrides any data tag attribute that may be set on the storage group. If the user does not specify a data tag attribute on the table space and the table space is contained in a storage group, the table space inherits the data tag value from the storage group. The data tag attribute can be set for any regular or large table space except the catalog table space (SQL0109N). The data tag attribute cannot be set for temporary table spaces and returns the SQL0109N message error.

An automatic storage table space inherits the overhead and transferrate attributes from the storage group it uses. When a table space inherits the transferrate attribute from the storage group it uses, the storage group's device read rate is converted from milliseconds per page read, taking into account the pagesize setting of the table space, as follows:

$$\text{TRANSFERRATE} = (1 / \text{DEVICE READ RATE}) * 1000 / 1024000 * \text{PAGESIZE}$$

The pagesize setting for both an automatic storage table space and a nonautomatic table space has the corresponding default TRANSFERRATE values:

Table 8. Default TRANSFERRATE values

PAGESIZE	TRANSFERRATE
4 KB	0.04 milliseconds per page read
8 KB	0.08 milliseconds per page read
16 KB	0.16 milliseconds per page read
32 KB	0.32 milliseconds per page read

The data tag, device read rate, and overhead media attributes for automatic storage table spaces can be changed to dynamically inherit the values from its associated storage group. To have the media attributes dynamically updated, specify the INHERIT option for the CREATE TABLESPACE or ALTER TABLESPACE statement.

When a table space inherits the value of an attribute from a storage group, the SYSCAT.TABLESPACES catalog table view reports a value of -1 for that attribute. To view the actual values at run time for the overhead, transferrate and data tag attributes, you can use the following query:

```
select tbspace,
       cast(case when a.datatag = -1 then b.datatag else a.datatag end as smallint) eff_datatag,
       cast(case when a.overhead = -1 then b.overhead else a.overhead end as double) eff_overhead,
       cast(case when a.transferrate = -1 then
                  (1 / b.devicereadrate) / 1024 * a.pagesize else a.transferrate end as double) eff_transferrate
  from syscat.tablespace a left outer join syscat.stogroups b on a.sgid = b.sgid
```

If you upgrade to V10.1, the existing table spaces retain their overhead and transferrate settings, and the overhead and device read rate attributes for the storage group are set to undefined. The newly created table spaces in a storage group with device read rate set to undefined use the DB2 database defaults that were defined when the database was originally created. If the storage group's media settings have a valid value, then the newly created table space will inherit those values. You can set media attributes for the storage group by using the ALTER STOGROUP statement. For nonautomatic table spaces, the media attributes are retained.

Creating storage groups

Use the CREATE STOGROUP statement to create storage groups. Creating a storage group within a database assigns storage paths to the storage group.

Before you begin

If you create a database with the AUTOMATIC STORAGE NO clause it does not have a default storage group. You can use the CREATE STOGROUP statement to create a default storage group.

Note: Although, you can create a database specifying the AUTOMATIC STORAGE NO clause, the AUTOMATIC STORAGE clause is deprecated and might be removed from a future release.

Procedure

To create a storage group by using the command line, enter the following statement:

```
CREATE STOGROUP operational_sg ON '/filesystem1', '/filesystem2', '/filesystem3'...
```

where *operational_sg* is the name of the storage group and */filesystem1*, */filesystem2*, */filesystem3* , ... are the storage paths to be added.

Important: To help ensure predictable performance, all the paths that you assign to a storage group should have the same media characteristics: latency, device read rate, and size.

Altering storage groups

You can use the ALTER STOGROUP statement to alter the definition of a storage group, including setting media attributes, setting a data tag, or setting a default storage group. You can also add and remove storage paths from a storage group.

If you add storage paths to a storage group and you want to stripe the extents of their table spaces over all storage paths, you must use the ALTER TABLESPACE statement with the REBALANCE option for each table space that is associated with that storage group.

If you drop storage paths from a storage group, you must use the ALTER TABLESPACE statement with the REBALANCE option to move allocated extents off the dropped paths.

You can use the DB2 Work Load Manager (WLM) to define rules about how activities are treated based on a tag that is associated with accessed data. You associate the tag with data when defining a table space or a storage group.

Adding storage paths

You can add a storage path to a storage group by using the ALTER STOGROUP statement.

About this task

When you add a storage path for a multipartition database environment, the storage path must exist on each database partition. If the specified path does not exist on every database partition, the statement is rolled back.

Procedure

- To add storage paths to a storage group, issue the following ALTER STOGROUP statement:

```
ALTER STOGROUP sg ADD '/hdd/path1', '/hdd/path2', ...
```

where *sg* is the storage group and */hdd/path1*, */hdd/path2*, ... are the storage paths being added.

Important: All the paths that you assign to a storage group should have similar media characteristics: underlying disks, latency, device read rate, and size. If paths have non-uniform media characteristics, performance might be inconsistent.

- After adding one or more storage paths to the storage group, you can optionally use the ALTER TABLESPACE statement to rebalance table spaces to immediately start using the new storage paths. Otherwise, the new storage paths are used only when there is no space in the containers on the existing storage paths. To determine all of the affected permanent table spaces in the storage group, run the following statement:

```
SELECT TBSP_NAME
      FROM tabTe (MON_GET_TABLESPACE(' ', -2))
     WHERE TBSP_USING_AUTO_STORAGE = 1
       AND TBSP_CONTENT_TYPE IN ('ANY', 'LARGE')
       AND STORAGE_GROUP_NAME = 'sg'
    ORDER BY TBSP_ID
```

Once the table spaces have been identified, you can perform the following statement for each of the table spaces listed:

```
ALTER TABLESPACE tablespace_name REBALANCE
```

where *tablespace_name* is the table space.

Dropping storage paths

You can drop one or more storage paths from a storage group or you can move data off the storage paths and rebalance them.

Before you begin

To determine whether permanent table spaces are using the storage path, use the ADMIN_GET_STORAGE_PATHS administrative view. This view displays current information about the storage paths for each storage group. A storage path can be in one of three states:

NOT_IN_USE

The storage path has been added to the database but is not in use by any table space.

IN_USE

One or more table spaces have containers on the storage path.

DROP_PENDING

An ALTER STOGROUP *stogroup_name* DROP statement has been issued to drop the path, but table spaces are still using the storage path. The path is removed from the database when it is no longer being used by a table space.

If the storage path you dropped has data stored on it and is in the DROP_PENDING state, you must rebalance all permanent table spaces using the storage path before the database manager can complete the drop of the path.

To obtain information about table spaces on specific database partitions use the MON_GET_TABLESPACE administrative view.

Restrictions

A storage group must have at least one path. You cannot drop all paths in a storage group.

About this task

If you intend to drop a storage path, you must rebalance all permanent table spaces that use the storage path by using ALTER TABLESPACE tablespace-name REBALANCE, which moves data off the path to be dropped. In this situation, the rebalance operation moves data from the storage path that you intend to drop to the remaining storage paths and keeps the data striped consistently across those storage paths, maximizing I/O parallelism.

Procedure

1. To drop storage paths from a storage group, issue the following ALTER STOGROUP statement:

```
ALTER STOGROUP sg DROP '/db2/filesystem1', '/db2/filesystem2'
```

where *sg* is the storage group and */db2/filesystem1* and */db2/filesystem2* are the storage paths being dropped.

2. Rebalance the containers of the storage paths being dropped. To determine all the affected permanent table spaces in the database that have containers residing on a "Drop Pending" path, issue the following statement:

```
SELECT TBSP_NAME
  FROM table(MON_GET_TABLESPACE(' ', -2))
 WHERE TBSP_USING_AUTO_STORAGE = 1
   AND TBSP_CONTENT_TYPE IN ('ANY', 'LARGE')
   AND STORAGE_GROUP_NAME = 'sg'
 ORDER BY TBSP_ID
```

Once the table spaces have been identified, you can perform the following statement for each of the table spaces listed:

```
ALTER TABLESPACE tablespace_name REBALANCE
```

where *tablespace_name* is a table space.

After the last rebalance operation is complete, */db2/filesystem1* and */db2/filesystem2* are removed from the storage group.

3. Drop the temporary table spaces using the storage group. A table space in DROP_PENDING state is not dropped if there is a temporary table space on it.
4. Re-create the temporary table spaces that were using the storage group.

What to do next

Query the ADMIN_GET_STORAGE_PATHS administrative view to verify that the storage path that was dropped is no longer listed. If it is, then one or more table spaces are still using it.

Monitoring storage paths

You can use administrative views and table functions to get information about the storage paths used.

The following administrative views and table functions can be used:

- Use the ADMIN_GET_STORAGE_PATHS administrative view to get a list of storage paths for each storage group and the file system information for each storage path.
- Use the TBSP_USING_AUTOMATIC_STORAGE and STORAGE_GROUP_NAME monitor elements in the MON_GET_TABLESPACE table function to understand if a table space is using automatic storage and to identify which storage group the table space is using.
- Use the DB_STORAGE_PATH_ID monitor element in the MON_GET_CONTAINER table function to understand which storage path in a storage group a container is defined on.

Replacing the paths of a storage group

Replace the storage paths in a storage group with new storage paths.

Procedure

To replace the existing storage paths in a storage group:

1. Add the new storage paths to an existing storage group.

```
ALTER STOGROUP sg_default ADD '/hdd/path3', '/hdd/path4'
```

2. Drop the old storage paths.

```
ALTER STOGROUP sg_default DROP '/hdd/path1', '/hdd/path2'
```

Note: All storage groups must have at least one path and that last path cannot be dropped.

This marks the dropped storage paths as DROP PENDING.

3. Determine the affected non-temporary table spaces.

```
SELECT TBSP_NAME
      FROM table (MON_GET_TABLESPACE(' ', -2))
     WHERE TBSP_USING_AUTO_STORAGE = 1
       AND TBSP_CONTENT_TYPE IN ('ANY', 'LARGE')
       AND STORAGE_GROUP_NAME = 'sg_default'
    ORDER BY TBSP_ID
```

4. Perform the following statement for each of the affected non-temporary table spaces returned.

```
ALTER TABLESPACE tablespace-name REBALANCE
```

5. If there are any temporary table spaces defined on the dropped storage paths, you must create the new temporary table spaces first before dropping the old ones.

```
SELECT TBSP_NAME
      FROM table (MON_GET_TABLESPACE(' ', -2))
     WHERE TBSP_USING_AUTO_STORAGE = 1
       AND TBSP_CONTENT_TYPE IN ('USRTEMP', 'SYSTEMP')
       AND STORAGE_GROUP_NAME = 'sg_default'
    ORDER BY TBSP_ID
```

Renaming storage groups

Use the RENAME STOGROUP statement to rename a storage group.

Procedure

Use the following statement to rename a storage group:

```
RENAME STOGROUP sg_hot TO sg_warm
```

where `sg_warm` is the new name of the storage group.

Example

When the first storage group is created at database creation time, the default storage group name is `IBMSTOGROUP`. You can use the following statement to change the designated default name:

```
RENAME STOGROUP IBMSTOGROUP TO DEFAULT_SG
```

where `DEFAULT_SG` is the new default name of the storage group.

Dropping storage groups

You can remove a storage group by using the `DROP` statement.

About this task

You must determine whether there are any table spaces that use the storage group before dropping it. If there are, you must change the storage group that the table spaces use and complete the rebalance operation before dropping the original storage group.

Restrictions

You cannot drop the current default storage group.

Procedure

To drop a storage group:

- Find the table spaces that are using the storage group.

```
SELECT TBSP_NAME, TBSP_CONTENT_TYPE  
  FROM table (MON_GET_TABLESPACE(' ', -2))  
 WHERE TBSP_USING_AUTO_STORAGE = 1  
   AND STORAGE_GROUP_NAME = STO_GROUP  
 ORDER BY TBSP_ID
```

where `STO_GROUP` is the storage group that you want to drop.

- If there are regular or large table spaces that use the storage group, assign them to a different storage group:

```
ALTER TABLESPACE tablespace_name USING STOGROUP sto_group_new
```

where `sto_group_new` is a different storage group.

- If there are temporary table spaces that use the storage group that you want to drop, perform these steps:

- Determine what temporary table spaces use the storage group that you want to drop:

```
SELECT TBSP_NAME  
  FROM table (MON_GET_TABLESPACE(' ', -2))  
 WHERE TBSP_USING_AUTO_STORAGE = 1  
   AND TBSP_CONTENT_TYPE IN ('USRTEMP', 'SYSTEMP')  
   AND STORAGE_GROUP_NAME = 'STO_GROUP'  
 ORDER BY TBSP_ID
```

- Drop the temporary table spaces using the storage group:

```
DROP TABLESPACE table_space
```

- Re-create the temporary table spaces that were using the storage group.

- Monitor the rebalance activity for the storage group to be dropped.

```
SELECT * from table (MON_GET_REBALANCE_STATUS( ' ', -2))
    WHERE REBALANCER_SOURCE_STORAGE_GROUP_NAME = sto_group_old
```

An empty result state indicates that all table spaces have finished moving to the new storage group.

5. Drop the storage group when all table space extents have been successfully moved to the target storage group.

```
DROP STOGROUP STO_GROUP
```

where *STO_GROUP* is the name of the storage group to be dropped.

Associating a table space to a storage group

Using the CREATE TABLESPACE statement or ALTER TABLESPACE statement, you can specify or change the storage group a table space uses. If a storage group is not specified when creating a table space, then the default storage group is used.

About this task

When you change the storage group a table space uses, an implicit REBALANCE operation is issued when the ALTER TABLESPACE statement is committed. It moves the data from the source storage group to the target storage group.

When using the IBM DB2 pureScale Feature, REBALANCE is not supported and you cannot change the assigned storage group. The REBALANCE operation is asynchronous and does not affect the availability of data. You can use the monitoring table function MON_GET_REBALANCE_STATUS to monitor the progress of the REBALANCE operation.

During the ALTER TABLESPACE operation, compiled objects that are based on old table space attributes are *soft invalidated*. Any new compilations after the ALTER TABLESPACE commits use the new table space attributes specified in the ALTER TABLESPACE statement. *Soft invalidation* support is limited to dynamic SQL only, you must manually detect and recompile any static SQL dependencies for the new values to be used.

Any table spaces that use the same storage group can have different PAGESIZE and EXTENTSIZE values. These attributes are related to the table space definition and not to the storage group.

Procedure

To associate a table space with a storage group, issue the following statement:

```
CREATE TABLESPACE tbspc USING STOGROUP storage_group
```

where *tbspc* is the new table space, and *storage_group* is the associated storage group.

Scenario: Moving a table space to a new storage group

This scenarios shows how a table space can be moved from one storage group to a different storage group.

The assumption in this scenario is that the table space data is in containers on storage paths in a storage group. An ALTER TABLESPACE statement is used to move the table space data to the new storage group.

When the table space is moved to the new storage group, the containers in the old storage group are marked as drop pending. After the ALTER TABLESPACE statement is committed, containers are allocated on the new storage group's storage paths, the existing containers residing in the old storage groups are marked as drop pending, and an implicit REBALANCE operation is initiated. This operation allocates containers on the new storage path and rebalances the data from the existing containers into the new containers. The number and size of the containers to create depend on both the number of storage paths in the target storage group and on the amount of free space on the new storage paths. The old containers are dropped, after all the data is moved.

The following diagram is an example of moving the table space from a storage group to a different storage group, where:

1. New containers are allocated on the target storage group's storage paths.
2. All original containers are marked drop pending and new allocation request are satisfied from the new containers.
3. A reverse rebalance is preformed, moving data off of the containers on the paths being dropped.
4. The containers are physically dropped.

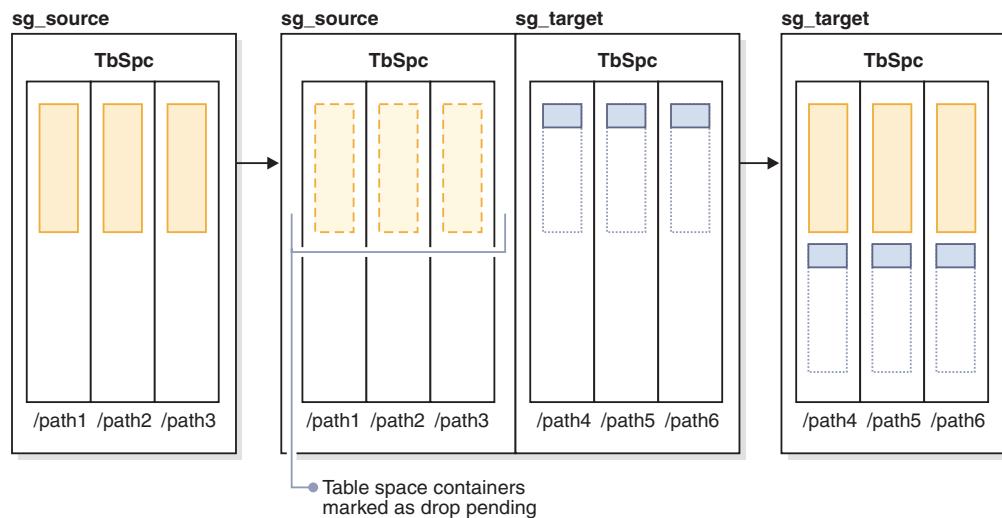


Figure 12. Moving a table space to a new storage group

To move a table space to a different storage group, do the following:

1. Create two storage groups, sg_source and sg_target:


```
CREATE STOGROUP sg_source ON '/path1', '/path2', '/path3'
CREATE STOGROUP sg_target ON '/path4', '/path5', '/path6'
```
2. After creating the database, create an automatic storage table space that initially uses the sg_source storage group:


```
CREATE TABLESPACE TbSpc USING STOGROUP sg_source
```
3. Move the automatic storage table space to the sg_target storage group:


```
ALTER TABLESPACE TbSpc USING sg_target
```

Chapter 5. Multi-temperature storage

You can configure your databases so that frequently accessed data (*hot data*) is stored on fast storage, infrequently accessed data (*warm data*) is stored on slightly slower storage, and rarely accessed data (*cold data*) is stored on slow, less-expensive storage. As hot data cools down and is accessed less frequently, you can dynamically move it to the slower storage.

In database systems, there is a strong tendency for a relatively small proportion of data to be hot data and the majority of the data to be warm or cold data. These sets of *multi-temperature data* pose considerable challenges if you want to optimize the use of fast storage by trying not to store cold data there. As a data warehouse consumes increasing amounts of storage, optimizing the use of fast storage becomes increasingly important in managing storage costs.

Storage groups are groups of storage paths with similar qualities. Some critical attributes of the underlying storage to consider when creating or altering a storage group are available storage capacity, latency, data transfer rates, and the degree of RAID protection. You can create storage groups that map to different classes of storage in your database management system. You can assign automatic storage table spaces to these storage groups, based on which table spaces have hot, warm, or cold data. To convert database-managed table spaces to use automatic storage, you must issue an ALTER TABLESPACE statement specifying the MANAGED BY AUTOMATIC STORAGE option and then perform a rebalance operation.

Because current data is often considered to be hot data, it typically becomes warm and then cold as it ages. You can dynamically reassign a table space to a different storage group by using the ALTER TABLESPACE statement, with the USING STOGROUP option.

The following example illustrates the use of storage groups with multi-temperature data. Assume that you are the DBA for a business that does most of its processing on current-fiscal-quarter data. As shown in Figure 13 on page 80, this business has enough solid-state drive (SSD) capacity to hold data for an entire quarter and enough Fibre Channel-based (FC) and Serial Attached SCSI (SAS) drive capacity to hold data for the remainder of the year. The data that is older than one year is stored on a large Serial ATA (SATA) RAID array that, while stable, does not perform quickly enough to withstand a heavy query workload. You can define three storage groups: one for the SSD storage (`sg_hot`), one for the FC and SAS storage (`sg_warm`), and the other for the SATA storage (`sg_cold`). You then take the following actions:

- Assign the table space containing the data for the current quarter to the `sg_hot` storage group
- Assign the table space containing the data for the previous three quarters to the `sg_warm` storage group
- Assign the table space containing all older data to the `sg_cold` storage group

After the current quarter passes, you take the following actions:

- Assign a table space for the new quarter to the `sg_hot` storage group
- Move the table space for the quarter that just passed to the `sg_warm` storage group

- Move the data for the oldest quarter in the sg_warm storage group to the sg_cold storage group

You can do all this work while the database is online.

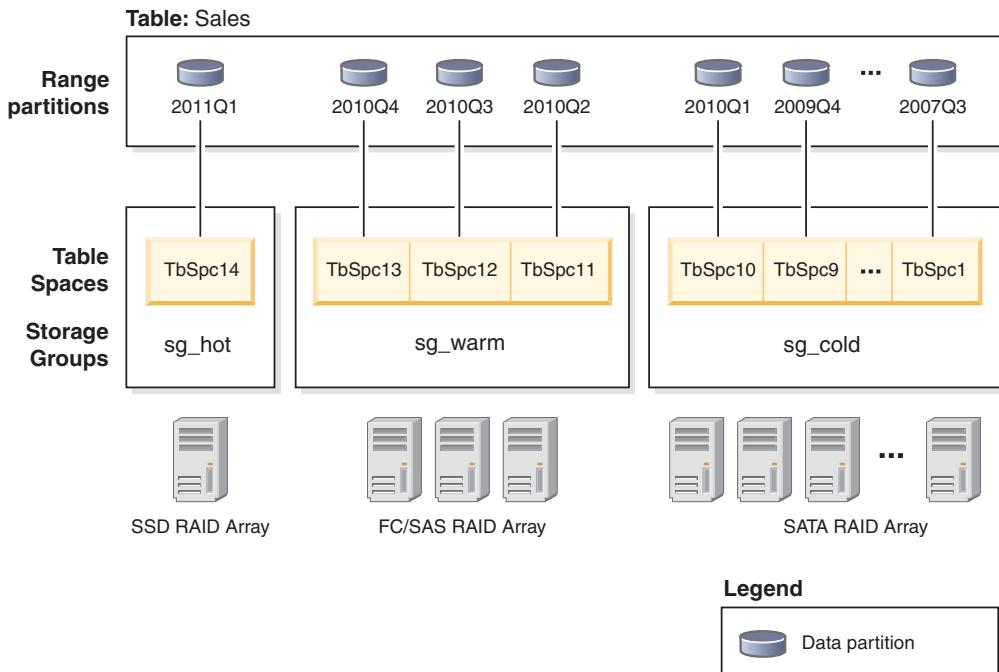


Figure 13. Managing Sales data using multi-temperature data storage

The following steps provide more details on how to set up multi-temperature data storage for the sales data in the current fiscal year:

- Create two storage groups to reflect the two classes of storage, a storage group to store hot data and a storage group to store warm data.

```
CREATE STOGROUP sg_hot ON '/ssd/path1', '/ssd/path2' DEVICE READ RATE 100
OVERHEAD 6.725;
CREATE STOGROUP sg_warm ON '/hdd/path1', '/hdd/path2';
```

These statements define an SSD storage group (sg_hot) to store hot data and an FC and SAS storage group (sg_warm) to store warm data.

- Create four table spaces, one per quarter of data in a fiscal year, and assign the table spaces to the storage groups.

```
CREATE TABLESPACE tbsp_2010q2 USING STOGROUP sg_warm;
CREATE TABLESPACE tbsp_2010q3 USING STOGROUP sg_warm;
CREATE TABLESPACE tbsp_2010q4 USING STOGROUP sg_warm;
CREATE TABLESPACE tbsp_2011q1 USING STOGROUP sg_hot;
```

This association results in table spaces inheriting the storage group properties.

- Set up your range partitions in your sales table.

```
CREATE TABLE sales (order_date DATE, order_id INT, cust_id BIGINT)
PARTITION BY RANGE (order_date)
(PART "2010Q2" STARTING ('2010-04-01') ENDING ('2010-06-30') in "tbsp_2010q2",
 PART "2010Q3" STARTING ('2010-07-01') ENDING ('2010-09-30') in "tbsp_2010q3",
 PART "2010Q4" STARTING ('2010-10-01') ENDING ('2010-12-31') in "tbsp_2010q4",
 PART "2011Q1" STARTING ('2011-01-01') ENDING ('2011-03-31') in "tbsp_2011q1");
```

The 2011Q1 data represents the current fiscal quarter and is using the sg_hot storage group.

4. After the current quarter passes, create a table space for a new quarter, and assign the table space to the sg_hot storage group.
`CREATE TABLESPACE tbsp_2011q2 USING STOGROUP sg_hot;`
5. Move the table space for the quarter that just passed to the sg_warm storage group. To change the storage group association for the tbsp_2011q1 table space, issue the ALTER TABLESPACE statement with the USING STOGROUP option.
`ALTER TABLESPACE tbsp_2011q1 USING STOGROUP sg_warm;`

Chapter 6. IBM Data Studio

IBM Data Studio provides application developers with a single integrated development environment that can be used to create, deploy, and debug data-centric applications. Built to extend the Eclipse framework and SQL model components, it combines Eclipse technology and shared repository extensions for database development.

IBM Data Studio consist of the following components:

- The IBM Data Studio client, which is an Eclipse-based tool that provides an integrated development environment for database and instance administration, routine and Java application development, and query tuning tasks. It can be installed with other IBM software products to share a common environment.
- The IBM Data Studio web console, which is a web-based tool with health and availability monitoring, job creation, and database administration tasks.

If you previously used the Control Center tools, review the mapping between the recommended Optim™ tools and Control Center tools that is available at “Table of recommended tools versus Control Center tools” in *What's New for DB2 Version 10.1 Version 9.7*.

Related information:

- ➡ [IBM Data Studio documentation](#)
- ➡ [Features in IBM Data Studio](#)
- ➡ [IBM Data Studio product Web page](#)
- ➡ [Download IBM Data Studio](#)

Using IBM Data Studio for key tasks

IBM Data Studio includes support for key tasks across the data management lifecycle, including administration, application development, and query tuning.

Administer databases, monitor health, and run jobs

IBM Data Studio provides database management and maintenance support, including object, change, and authorization management, scripting, basic health and availability monitoring, and job scheduling to automate changes to the database.

Develop database applications

IBM Data Studio helps developers and database administrators develop, debug, and deploy database applications and database routines (SQL or Java stored procedures and user-defined functions).

Data access development support

If data access development support is enabled for Java applications, developers and database administrators can use IBM Data Studio to understand the relationship between database objects, source code, and SQL statements that are in the source code. Data access development support also provides client metrics for SQL statements.

pureQuery support

If pureQuery support is enabled, developers can use the integrated

InfoSphere Optim pureQuery Runtime and the pureQuery APIs to create Java applications. With the APIs, developers can use the integrated Java editor and simple pureQuery syntax to create a simple Java data access layer with the data access object (DAO) pattern.

Tune queries

IBM Data Studio includes basic query tuning tools, such as query formatting, access path graphs, and statistics advice to help developers and SQL tuners create higher performance queries. You can also use IBM Data Studio to access the tuning features of IBM InfoSphere Optim Query Workload Tuner when you connect to a DB2 database or subsystem on which a license for InfoSphere Optim Query Workload Tuner is active.

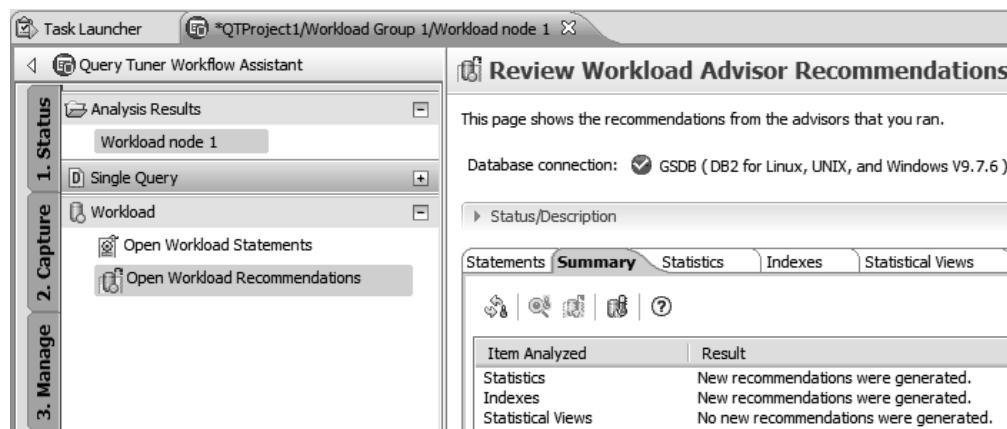


Figure 14. The review section of the Query Tuner Workflow Assistant in the IBM Data Studio client.

Scenario: IBM Data Studio in a team environment

You can install multiple instances of the Data Studio components to mirror your organization and support the structure of your enterprise. For example, in an organization with users with different roles and access privileges, your team can install multiple instances of the Data Studio client.

The following illustration shows a complex use scenario that consists of a database designer and multiple database administrators and application developers who all have different access privileges to the test and production database and to the Data Studio web console.

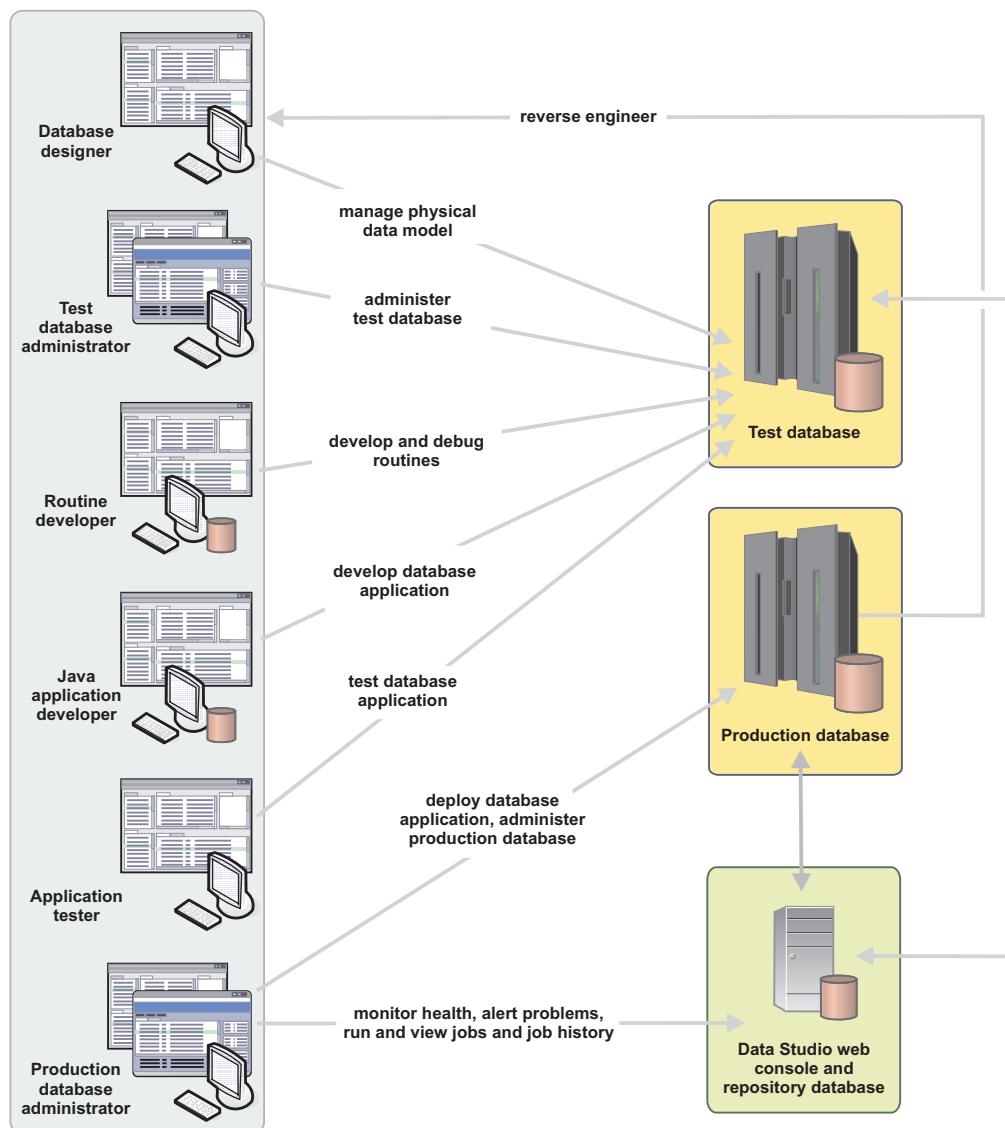


Figure 15. Topology diagram of a complex installation and use scenario.

Most users in this scenario have access to only the test database, but the test database administrator has additional access to the Data Studio web console. Similarly, the production database administrator has access to both the production database and the Data Studio web console. The administrators with access to the Data Studio web console, can monitor database health, manage jobs, and manage and share database connection information across the organization.

For information about other tools and solutions that can help you with the tasks and responsibilities throughout the data management lifecycle, see InfoSphere Optim Data Management Solutions.

IBM Data Studio client

The IBM Data Studio client is built on Eclipse technology and provides an integrated development environment for database and instance administration, routine and Java™ application development, and query tuning.

The IBM Data Studio client is one of two IBM Data Studio components: the client and the web console. For information about the IBM Data Studio web console component, which you use for job management and health monitoring, see “IBM Data Studio web console” on page 88.

The IBM Data Studio client includes the following features:

Activities and perspectives

The tools that you use as a database administrator or application developer depend on your role. Data Studio provides two primary ways to access these tool sets: activities and perspectives.

Activities

Use the activity menu in the toolbar to switch between Data Studio activities. These preconfigured activities correspond to a subset of the Data Studio perspectives. The following activities are available:

- Administer Databases
- Develop SQL and Routines
- Develop Java Applications
- Tune Queries
- Run SQL

You can return to your preset home activity from any perspective by clicking the home activity button in the toolbar.

Perspectives

You can also access other tools that you need for your role by switching to other perspectives from the **Window > Open Perspective** menu. For example, the primary perspective for database administration is the Database Administration perspective, and the perspectives for application development are the Database Development and Java perspectives.

Depending on your role, other perspectives that you can use include the Data, Java, SQL and Routine Development, and Query Tuning perspectives.

Getting started tools

The following tools can help you get started with the IBM Data Studio client:

- If you are new to the Eclipse development environment, view the Eclipse Basic tutorial. Click the link, then click the **Show in Table of Contents** icon in the information center to see the entire tutorial.
- Several tutorials are available to help you get started with IBM Data Studio. To access tutorials, expand the Tutorials category in the Contents pane of information center.
- Use the Task Launcher in the IBM Data Studio client to view and start many getting started tasks and other key tasks in the product. Task Launcher opens when you start IBM Data Studio, or you can open it by clicking **Help > Task Launcher**.

Database object management

Changing database objects requires determining which changes need to be made, specifying those changes, evaluating the effects of those changes, and then deploying them.

An editable Properties view and the Review and Deploy dialog box provides a consistent way to create, alter, and drop objects. You can also manage the privileges objects for various types of database servers. After you define your changes in the Properties view, IBM Data Studio automatically generates the commands that can make the changes. The generated commands are displayed in the Review and Deploy dialog box, where you can review the commands and run them.

More robust change management features are provided for DB2 for Linux, UNIX, and Windows databases because a change plan is used to manage the changes. A change plan makes more complex changes possible and you can use it to change more than one object at a time. IBM Data Studio manages the dependent objects and takes resulting actions to address any side effects that are caused by your database object changes. With change plans, you can also preserve your data across database changes, undo your database changes, and track your changes with a version control system.

Data application developer features

For data application developers, IBM Data Studio provides the following key features. Working in a data development project in the Data Project Explorer, you can:

- Create, test, debug, and deploy routines, such as stored procedures and user-defined functions. See [Developing database routines](#).
- Create, edit, and run SQL queries. Use the SQL Query builder and the SQL and XQuery editor .
- View and capture performance data for SQL statements. See [Import and view SQL performance data](#) and [Run SQL statements and capture performance data](#).
- Use Visual Explain to diagram access plans. See [Diagramming access plans with Visual Explain](#).
- Use query tuning features to improve the performance of SQL statements in applications. See [Tuning single SQL statements](#).
- Debug stored procedures. See [Use the Routine debugger](#).
- Create web services that expose database operations (SQL SELECT and DML statements, XQuery expressions, or calls to stored procedures) to client applications. See [Developing and deploying Web services](#).
- Develop SQLJ applications in a Java project. See [Developing SQLJ applications](#).
- Develop XML applications. See [Use wizards and editors to develop XML applications](#).

Query tuning features

With the IBM Data Studio client, you can format SQL statements so that they are easier to read, generate visual representations of access plans, and get recommendations for collecting statistics on the objects that a statement references. You can also generate a report that summarizes information about the access plan and includes the recommendations.

If you connect the IBM Data Studio client to a DB2 database or subsystem on which a license for InfoSphere® Optim Query Workload Tuner is active, you can use the full set of tuning features.

Team features

If you are working on a large team, you can share data development projects by using supported code control systems, and you can share database connection information.

For more information, see:

- Supported source code control systems.
- Share database connection information by importing and exporting this information to XML files.

Shell-sharing with other Eclipse-based products

Shell-sharing (sharing a common environment) with other Eclipse-based products makes it easy to share the functions between products from one interface. If you install the IBM Data Studio client into the same product group as a compatible product, you install only one version of Eclipse that shares the components of each product. Shell-sharing saves disk space and avoids duplicating components that are already built into other products.

Another benefit of shell sharing is the ability to have products interact with each other, which makes each product stronger than if they were run alone. For example, the following shell-sharing scenario shows the strength of using IBM InfoSphere Data Architect and IBM Data Studio together:

1. Shell-share InfoSphere Data Architect with the IBM Data Studio client.
2. Create glossary models to standardize your naming conventions by using InfoSphere Data Architect.
3. Use the database administration features of the IBM Data Studio client to ensure that those naming conventions are followed.

To shell-share products, the base Eclipse versions must be the same. For example, you cannot shell-share an Eclipse version 3.6-based product with an Eclipse version 4.2-based product.

For more information, see:

- Coexistence considerations
- Limitations for sharing a common environment between IBM products based on Eclipse
- IBM Software products installed together that share a common environment

For more details about the installation packages that are available for IBM Data Studio, see the product web page.

IBM Data Studio web console

IBM Data Studio web console provides health and availability monitoring features and job creation and management functions for DB2 for Linux, UNIX, and Windows and DB2 for z/OS® databases. Use the health pages of the web console to view alerts, applications, utilities, storage, and related information and use the job manager pages to create and manage script-based jobs across your databases.

IBM Data Studio web console is available as a stand-alone web interface or integrated with the IBM Data Studio client.

Tip: You can use IBM Data Studio web console in single-user mode to test the product in a controlled environment, or in multi-user mode to share monitoring features across your production servers.

- In single-user mode, log in to the IBM Data Studio web console as a single user with full administrative rights.
- In multi-user mode, configure access to the web console and permissions to monitor and perform other actions on the specific databases for users and groups.

Database administration with IBM Data Studio

You can run database administration commands for hosts, instances, and databases that are displayed in the Administration Explorer and for databases, table spaces, tables, and indexes that are displayed in the Object List.

As a database administrator, you might be responsible for maintaining, managing, and administering DB2 instances, databases, and database objects such as table spaces, tables, and views. For example, your backup and recovery strategy might require you to take periodic backups of your databases. As another example, over time, the data in your tables might become fragmented, which can increase the size of your tables and indexes as the records are distributed over more and more data pages. To reclaim wasted space and improve data access, you likely will need to reorganize your tables and indexes.

Managing and maintaining your database systems might require you to run database administration commands, which include:

- DB2 commands
- System commands
- Utilities
- SQL statements

When the Database Administration feature of IBM Data Studio is installed, task assistants are available for DB2 for Linux, UNIX, and Windows databases. These task assistants guide you through typical database administration tasks. From the Administration Explorer, you can do the following types of administration tasks:

- Maintenance mode management for DB2 pureScale hosts
- Instance management, including starting and stopping instances
- Database management, including creating and dropping databases, configuring logging or automatic maintenance, setting up and managing the DB2 High Availability Disaster Recovery (HADR) feature, and backing up, restarting, and recovering databases

When you click a data object folder (also called a *flat folder* in the Administration Explorer to display database objects in the Object List, you can do the following types of administration tasks:

- Table space management, including backing up, restoring, and recovering table spaces
- Table management, including unloading and loading data, and setting the integrity of tables
- Index management, including reorganizing indexes
- Package management, including rebinding packages

You can also manage databases in the Object List.

When you right-click an object in the Administration Explorer or the Object List, a context-sensitive menu displays the list of the database administration commands that are available for that object. When you select a database administration command for that object, a database administration task assistant is displayed. The task assistant guides you through the process of setting any options for the database administration command, previewing the commands that are automatically generated, and running the commands for the object.

For databases that are not DB2 for Linux, UNIX, and Windows, you can use the SQL and XQuery editor to create and run your database administration commands.

Database administration for partitioned databases

The DB2 Database Partitioning feature (DPF) allows the partitioning of a database into two or more partitions that can reside on either the same server or on a different server.

For partitioned databases, the task assistants that guide you through the process of setting up the options for the commands include the ability to specify whether to run the commands against all of the partitions, one or more specific partitions, or partition groups. You can also choose to run the commands against the partitions in parallel, which is particularly useful for long-running commands. If you save the commands to a script, the commands will be run sequentially.

The control to run commands on individual partitions gives you more flexibility in managing your databases and resources. The granularity of an operation can determine how long it will take. For example, if a database has hundreds of partitions, backing up sets of partitions at different times or on different days might make more sense than backing up all of the partitions at the same time. As another example, depending on the system resources that are available to each partition, you might want to set certain configuration parameters across non-catalog partitions and customize the parameters for the catalog partition to optimize performance.

Database administration for the DB2 pureScale Feature for DB2 Enterprise Server Edition

In a DB2 pureScale environment, task assistants provide these additional database administration operations for members and cluster caching facilities (CFs):

Start To start members or CFs, select **Start** on the context-sensitive menu for the instance object. You can start selected members or CFs, all members and CFs, or an instance on a host.

Stop To stop members or CFs that are currently active, select **Stop** on the context-sensitive menu for the instance object. You can stop selected members or CFs, all the members and CFs, or an instance on a host.

Quiesce

To quiesce members, select **Stop** on the context-sensitive menu for the instance object and select the option to **Quiesce member with timeout**.

Configure

To change the configuration parameters for one or more members, select **Configure** on the context-sensitive menu for the instance object.

In addition, you can manage the maintenance mode for a DB2 pureScale host.

Administering databases with task assistants

IBM Data Studio provides dialogs that are called task assistants that help you create and run database administration commands for objects in DB2 for Linux, UNIX, and Windows databases. For example, you can use task assistants to start and stop databases and instances, configure database parameters, reorganize tables and indexes, back up and restore databases or table spaces, and import and export table data.

Before you begin

To run a database administration command for an object, you must have the appropriate permission or authorization for the object, and you must have a connection to the database that contains the object.

For Linux operating systems

Before users who do not have DB2 instance level privileges can use the DB2 client CLP to run commands, including commands for importing, loading, exporting, or unloading tables, the DB2INSTANCE system environment variable must be set.

Note: You must set the DB2INSTANCE system environment variable for each time that you log in or open a command terminal.

To set the DB2INSTANCE system environment variable:

1. Ensure that the Data Studio client is closed and not running before you run the script.
2. Set the environment variable at the instance level by running one of the following scripts:
 - For a Bourne or Korn shell, run: db2profile
 - For a C shell, run: db2cshrc
3. Start IBM Data Studio client.

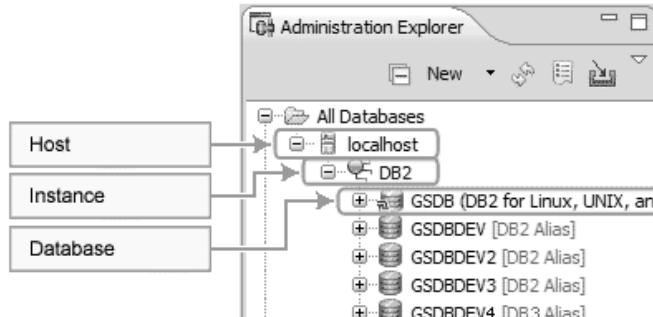
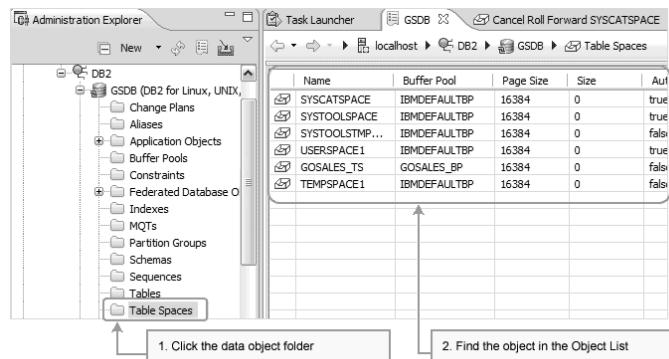
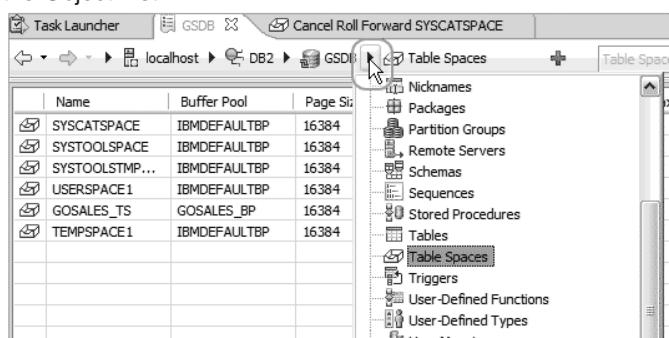
For detailed information about setting the system environment variable, see the *Setting environment variables outside the profile registries on Linux and UNIX operating systems* topic in the IBM DB2 online information for your version of the DB2 database.

Procedure

To open the task assistant for the command, specify additional settings for the command, and run the generated commands, complete the following steps:

1. Find the object that you want to work with. You can find the object either in the Administration Explorer or the Object List.

Table 9. Which view to use to find objects

View and objects	Example
Administration Explorer Hosts Instances Databases	From the Administration Explorer, you can open a task assistant for hosts, instances, and databases. 
Object List Databases Table spaces Indexes Views Aliases Packages	When you click a database or a data object folder in the Administration Explorer, the list of objects is displayed in the Object List.  <p>Figure 17. Example of selecting the Table Spaces folder to display the table spaces in the Object List</p> <p>Tip: If the Object List is already open for the database, you can use the drop-down arrow that is displayed after the database name in navigation breadcrumb trail to display other objects in the Object List.</p>
	 <p>Figure 18. Example of using a drop-down arrow in the Object List to select other objects to display</p>

- Right-click the object and select the command to run from the context-sensitive menu.

For example, the following figure shows how to back up the GSDB database.

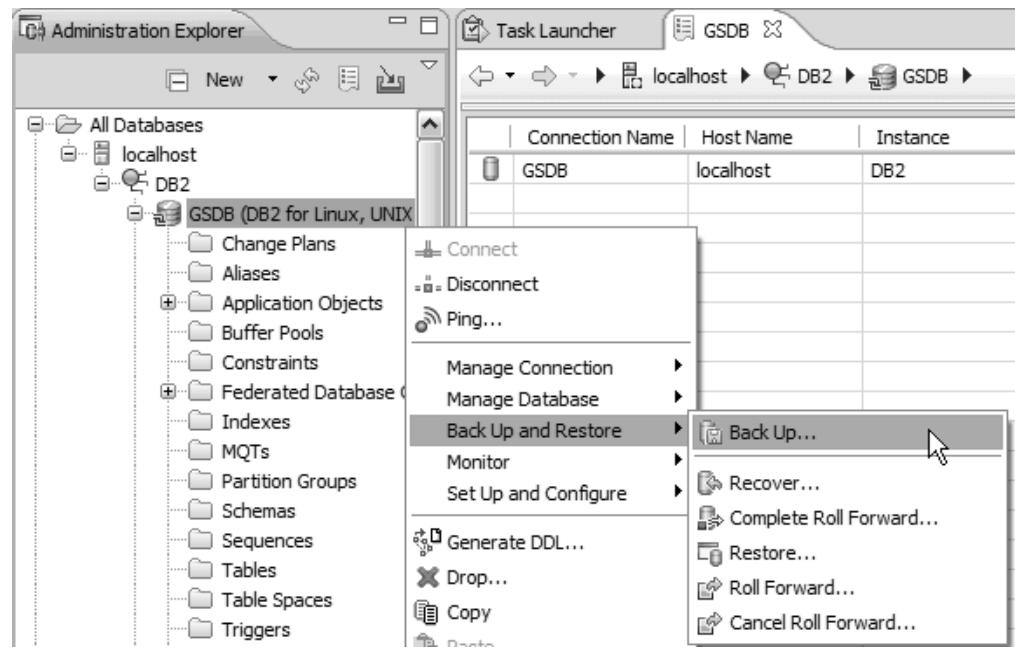


Figure 19. Example of the context-sensitive menu for databases and selecting to back up a database

The task assistant opens for the database administration task that you selected. Each task assistant has four sections: **Connection**, **Settings**, **Command**, and **Messages**. The following graphic shows how the **Connection**, **Command**, and **Messages** sections are expandable. The **Settings** section is always expanded.

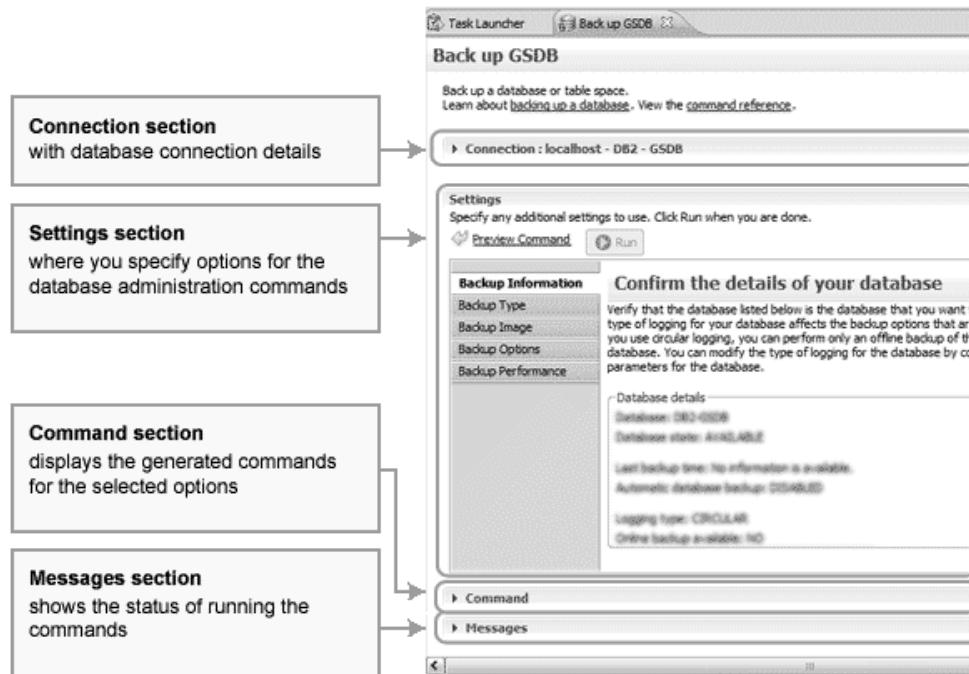


Figure 20. Example of a task assistant with its four sections

3. In the **Settings** section, specify the options for the command:
 - a. Click each of the tabs to step through the process of specifying the settings and options to use in the command.
 - b. Click **Preview Command** to shift down to and expand the **Command** section, where the generated commands that are based on the options that you specified are displayed.

For example, the following figure shows selecting the **Backup Performance** tab to specify options that improve the performance of the backup operation.

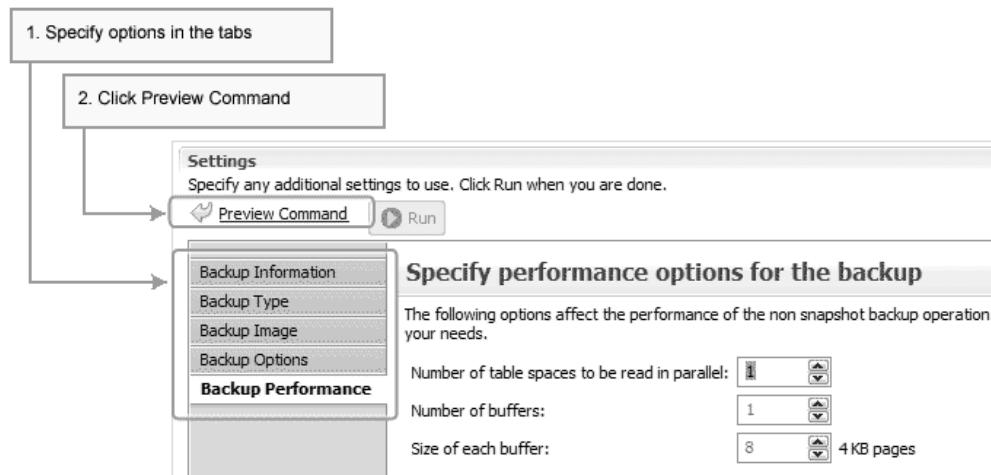


Figure 21. Example of selecting a tab and specifying options on that tab

4. In the **Command** section, review the commands that were generated and then run them.

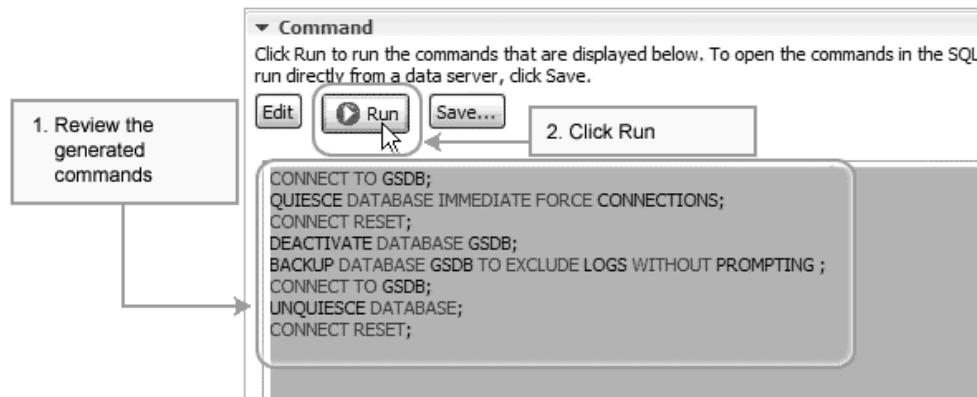


Figure 22. Example of the commands being displayed in the Command section

If you are satisfied with the displayed commands, click **Run** to run them.

Tip: In some cases, you might want to edit the displayed commands. Click **Edit** to open the SQL and XQuery editor where you can edit and run the commands. You might also want to schedule a time to run the commands.

The focus of the task assistant shifts to the **Messages** section.

5. In the **Messages** section, monitor the progress of the commands that are being run in the progress bar and review any messages that are issued.



Deployment status for the current command

[Display in SQL Results view](#)

Running Back up GSDB

Figure 23. Example of the Messages section while the commands are being run

To view detailed information about any command that does not run successfully, click the message number or SQL code that is displayed.

Results

The commands to perform your database administration task were run.

Database administration commands that you can run from task assistants

Use the numerous task assistants to guide you through the process of setting options for common database administration commands, reviewing the automatically generated commands, and running the commands.

For each object, the following table shows which database administration commands are supported by a task assistant. Task assistants are available only for databases on DB2 for Linux, UNIX, and Windows.

Table 10. Task assistant support for DB2 for Linux, UNIX, and Windows database administration commands

Action	Database administration command	Description
Hosts		
Manage DB2 pureScale Host Maintenance Mode	db2cluster with one of the following options <ul style="list-style-type: none">• -cm -enter -maintenance [-all]• -cm -exit -maintenance [-all]• -cfs -enter -maintenance [-all]• -cfs -exit -maintenance [-all]	<p>Either puts or removes the target host or all hosts from maintenance mode. The target host must be in maintenance mode to apply software updates to DB2 cluster services.</p> <p>The host must have at least one DB2 pureScale instance, and at least one connection profile was created and connected.</p>
Verify DB2 cluster services status	db2cluster with one of the following options <ul style="list-style-type: none">• -cm -verify -resources• -cfs -verify -configuration• -cm -verify -maintenance• -cfs -verify -maintenance• -cfs -verify -configuration -filesystem <i>fsname</i>• -cfs -list -filesystem	<p>Verify the status and configuration of DB2 cluster services. The task assistant can do the following tasks:</p> <ul style="list-style-type: none">• Verify that the resource model for the instance is correct, and that there are no inconsistencies in the resource model.• Verify the configuration of the current file system cluster.• Display if the cluster manager is offline on the host so that the binaries can be updated.• Display if the shared file system cluster host is offline to allow for updates to the binaries.• Verify the configuration of the current file system cluster.• List the current file systems.

Table 10. Task assistant support for DB2 for Linux, UNIX, and Windows database administration commands (continued)

Action	Database administration command	Description
Manage DB2 cluster services configuration	db2cluster with one of the following options <ul style="list-style-type: none"> • -cm -set -option -pprimary <i>host name</i> • -cm -set -option HostFailureDetection Time [1..10] • -cm -set -option autofailback [ON/OFF] • -cm -set -tiebreaker -disk [<i>disk name</i> PVID=<i>pvid</i>] • -cfs -set -option -tiebreaker -disk <i>disk name</i> 	Manage the configuration of the DB2 cluster services. The task assistant can do the following tasks: <ul style="list-style-type: none"> • Set a new preferred primary value for DB2 cluster services. The option specifies which cluster caching facility DB2 cluster services will attempt to start in the primary role. Attention: Make sure to check that the preferred primary value was changed after you run the command by looking at the output of the command in the SQLResults view. The db2cluster command is not recognized by DB2 CLP, so the command will run as an external command. DB2 CLP always reports success for any external command irrespective of the actual result of the external command. This behavior will be addressed in future releases. • Set how long it takes to detect a host failure or network partition. The value specifies the interval for detecting host failure. • Enable or disable automatic failback for the cluster. • Set the cluster manager quorum type to a disk tiebreaker.
Instances		
Configure	UPDATE DATABASE MANAGER CONFIGURATION	Modifies individual entries in the database manager configuration file.
Quiesce	QUIESCE	Forces all users off the specified instance and puts the instance into quiesced mode.
Start	db2start	Starts the DB2 instance.
Stop	db2stop	Stops the DB2 instance.
Unquiesce	UNQUIESCE	Restores user access to instances that were quiesced for maintenance or other reasons.
Databases		
Backup	BACKUP DATABASE	Creates a backup copy of a database or table space.
Configure	UPDATE DATABASE CONFIGURATION	Modifies individual entries in a specific database configuration file.
Configure Automatic Maintenance	UPDATE DATABASE CONFIGURATION	Enables or disables the various automatic maintenance activities that can be performed and defines a maintenance interval and window in which the activities can occur. Maintenance activities can occur during the maintenance window only if DB2 determines that the maintenance is required.

Table 10. Task assistant support for DB2 for Linux, UNIX, and Windows database administration commands (continued)

Action	Database administration command	Description
Configure Database Logging	UPDATE CONFIGURATION LOGGING	Modifies the data logging options for your database, such as the type of logging to use, the size of the log files, and the location where log files will be stored.
Create	CREATE DATABASE	Creates a database with either automatic or manual storage.
Drop	DROP DATABASE	Deletes the database contents and all log files for the database, uncatalogs the database, and deletes the database subdirectory.
Revalidate	SYSCAT.INVALIDOBJECTS ADMIN_REVALIDATE_DB_OBJECTS	<p>Lists all invalid objects for your database, and lets you revalidate these objects. The following object types are supported:</p> <p>DB2 V10.1 and higher</p> <ul style="list-style-type: none"> • COLUMN MASK • GLOBAL VARIABLE • ROUTINE • ROW PERMISSION • TRIGGER • USER-DEFINED DATA TYPE • VIEW <p>DB2 V9.7</p> <ul style="list-style-type: none"> • GLOBAL VARIABLE • ROUTINE • TRIGGER • USER-DEFINED DATA TYPE • VIEW
HADR Setup	Various	Sets up the High Availability Disaster Recovery (HADR) feature for your database. The HADR feature ensures that changes to the database can be replicated to one or more standby databases. A standby database takes over in the event of a failure on the primary system.
HADR Manage	START HADR, STOP HADR, TAKEOVER HADR	Starts and stops HADR operations on either the primary database or a standby database. You can also instruct a standby database to take over as the primary database in the event of a failure on the primary system.
List or Force Applications	FORCE APPLICATIONS	Forces local or remote users or applications off of the system to allow for maintenance on a server.
Manage Storage	ALTER DATABASE ADD STORAGE	Specifies that one or more new storage locations are to be added to the collection of storage locations that are used for automatic storage table spaces.
Quiesce	QUIESCE	Forces all users off of the specified database and puts the database into quiesced mode.

Table 10. Task assistant support for DB2 for Linux, UNIX, and Windows database administration commands (continued)

Action	Database administration command	Description
Recover	RECOVER DATABASE	Restores and rolls forward a database to a particular point in time or to the end of the logs.
Restart	RESTART DATABASE	Restarts a database that has been abnormally terminated and left in an inconsistent state.
Restore	RESTORE DATABASE	Re-creates a damaged or corrupted database that was backed up with the DB2 backup utility.
Roll Forward	ROLLFORWARD DATABASE	Recovers a database by applying transactions that were recorded in the database log files.
Cancel Roll Forward	ROLLFORWARD DATABASE with CANCEL option	Cancels the roll-forward recovery operation. The database is put in restore pending status.
Complete Roll Forward	ROLLFORWARD DATABASE with COMPLETE option	For databases that were archived and restored, but did not have the logs rolled forward, rolls forward the logs. The logs can be rolled forward to a point in time or to the end of the log.
Start	ACTIVATE DATABASE	Activates the specified database and starts all necessary database services so that the database is available for connection and use by any application.
Stop	DEACTIVATE DATABASE	Deactivates the specified database.
Unquiesce	UNQUIESCE	Restores user access databases that were quiesced for maintenance or other reasons.
Table spaces		
Backup	BACKUP	Creates a backup copy of a table space.
Restore	RESTORE	Re-creates a damaged or corrupted table space that was backed up with the DB2 backup utility. The task assistant does not support restoring multiple table spaces.
Rollforward	ROLLFORWARD DATABASE	Recovers a table space by applying transactions that were recorded in the log files.
Cancel Roll Forward	ROLLFORWARD DATABASE with CANCEL option	Cancels the roll-forward recovery operation. One or more table spaces are put in restore pending status.
Complete Roll Forward	ROLLFORWARD DATABASE with COMPLETE option	For table spaces that were archived and restored, but did not have the logs rolled forward, rolls forward the logs. The logs can be rolled forward to a point in time or to the end of the log.
Tables		
Convert Table	ADMIN_MOVE_TABLE	Converts row-organized tables to column organization.
Export Table	EXPORT	Exports data from a table to one of several external file formats.
Import Table	IMPORT	Inserts data from an external file with a supported file format into a table.
Load Table	LOAD	Loads data into a DB2 table.

Table 10. Task assistant support for DB2 for Linux, UNIX, and Windows database administration commands (continued)

Action	Database administration command	Description
Optim High Performance Unload	db2hpu	<p>Uses Optim High Performance Unload commands to unload data from a DB2 table or to copy data from source tables to target tables by using temporary files to store the data. You can also migrate data from one database system to another.</p> <p>To specify Optim High Performance Unload as the unload method, Optim High Performance Unload for Linux, UNIX, and Windows must be installed on each database system that is involved in the unload or migration.</p>
Reorg Table	REORG TABLE	Reorganizes a table.
Reorg Index	REORG INDEX	Reorganizes all of the indexes that are defined for the table.
Run Statistics	RUNSTATS	Updates the statistics about the characteristics of a table, its indexes, or both.
Set Integrity	SET INTEGRITY	Brings tables out of set integrity pending state, places tables in set integrity pending state, places tables into full access state, or prunes the contents of staging tables.
Packages		
Rebind	REBIND PACKAGE	Re-creates a package without needing the original bind file.

Tip: To create a database by using the context-sensitive menu, another database must exist. To create the first database in an instance, you can use the **New Database** icon in the toolbar at the top of the Administration Explorer.

Managing jobs in IBM Data Studio

IBM Data Studio web console provides job creation, job scheduling, and job management for your DB2 for Linux, UNIX, and Windows and DB2 for z/OS databases.

With the Data Studio web console job manager you can:

- Create and schedule jobs directly from the IBM Data Studio client workbench.
 - Use the workbench script editor to create your script and then schedule the script to run as a job in the job manager.
 - Access the Data Studio web console either embedded in the workbench or in a stand-alone web browser window.
 - Access the job history for a database directly from the Administration Explorer in the workbench.
- Create and manage jobs by using the web console graphical user interface.
 - View jobs, schedules, and notifications filtered by criteria such as database, job ID, or job type.
- Create jobs based on database scripts:

SQL-only scripts

The SQL-only scripts are run by the job manager by running the SQL commands that are outlined in the script part of the job directly against the database.

DB2 CLP scripts

The DB2 CLP script jobs are run on the database server by the job manager, which logs in to the database server by using SSH. For multiple databases, the job manager logs in as the user ID that is defined in the database connection. For a single database, based on the user's selection, the job manager logs in by using SSH credentials that the user supplies or the user ID that is defined in the database connection. When logged in, the job manager runs command line processor commands directly on the DB2 console of the server.

Important: To be able to run DB2 CLP script jobs on a database, the user ID that is used to run the job must have permission to log in to the database server by using SSH.

Executable/shell Scripts

The Executable/Shell script jobs are run on the database server by the job manager, which logs in to the database server by using SSH. For multiple databases, the job manager logs in as the user ID that is defined in the database connection. For a single database, based on the user's selection, the job manager logs in by using SSH credentials that the user supplies or the user ID that is defined in the database connection. When logged in, the job manager runs shell commands directly on the server.

Important: To be able to run Executable/Shell script jobs on a database, the user ID that is used to run the job must have permission to log in to the database server by using SSH.

- Schedule jobs to run at a specific time, or to repeat at certain intervals for one or more databases.
- Run jobs for multiple databases as the default user stored in the database connection, or specify a user ID to run the job as when running a job on one database.
- Add jobs together in chains, where the main job is followed by a secondary job dependent on the outcome of the main job, and where a finishing job, such as **RUNSTATS** and **BACKUP**, is run last.
- Configure email notifications to be sent to one or more users depending on the success or failure of the job.
- View the history of all jobs that run on your databases.
 - The job history view gives you a high-level overview of the job results and the option to drill down into each job.
 - You can configure the job manager to retain job history for all jobs that were run, or for a subset depending on the success or failure of the job.
- Manage user access to job manager tasks across your databases.
 - Enable or disable job management privileges requirements for the users of the web console.
 - For each database, grant or revoke job management privileges for each user of the web console.

Creating and managing jobs

With Data Studio web console job manager, you can create and manage your database jobs from the web console.

You create and manage your jobs by using the following tabs of the Job Manager page:

Job List

From this tab, you can create jobs for your databases or run existing jobs directly against a database without scheduling.

When you create a job or open an existing job, the job details open in the job editor. Use the tabs in the job editor to move between jobs, or use the job section view selector to drill down into the script, schedule, notification, and chain component of each job.

Tip: If you have configured your IBM Data Studio client to connect to IBM Data Studio web console you can create jobs directly from the SQL script editor.

Schedules

From this tab, you can create and manage schedules for the jobs that you created for your databases.

A schedule defines when a job will be run, whether the job is repeating, and whether the schedule is limited in number of runs or in time. The schedule also defines one or more databases on which to run the job.

Notifications

Use this tab to manage email notifications for the execution of the jobs that you created for your databases.

Job manager notifications help you monitor the execution results for your jobs across multiple databases and schedules without requiring access to the web console.

Each job can have any number of notifications configured, and each notification can be set up with different conditions, a different set of users to notify, and different collections of databases to monitor.

History

On this tab, you can view the status of jobs that ran on your databases. The job history is displayed for jobs that ran according to a schedule in addition to jobs that you ran manually over the last few days.

Tip: If you have configured your IBM Data Studio client to connect to IBM Data Studio web console you can view job history for a database directly from the Administration Explorer.

Scenario: Creating and scheduling a job

In this scenario, Alan, a database administrator with the Sample Company, uses the job manager to create and schedule a job based on a script provided by Doug, a developer, on the Sales database owned by Becky, a database administrator.

To complete the parts of the scenario, Alan uses the following web console pages of Data Studio web console:

- Databases
- Job Manager

- Job List tab
- Schedules tab
- Notifications tab
- History tab
- Console Security

Alan is a database manager for Sample Company, and is responsible for scheduling database jobs. Alan works with the database script developers for the script content of the jobs and with the database owners to get the required credentials to access the databases. Alan owns the repository database that is used by Data Studio web console to manage user access to the web console.

Alan is approached by Doug, a script developer who asks Alan to schedule a script to be run on the Sales database monthly, and to notify Doug and Doug's manager if the job fails. In addition, each time the script runs, an existing Cleanup job must be run directly afterward.

First Alan verifies with Doug that the script has been tested and verified by development, and that it runs without problems on their test databases. Doug uses other IBM Data Studio tools to verify the scripts.

Next, Alan opens the Databases page in the web console to verify that the Sales database exists as a database connection. If needed, he adds a database connection to the Sales database with information from Becky, the owner of the Sales database. Becky wants to restrict the running of jobs on the **Sales** database to a specific subset of users, so Alan configures the database connection to connect with a user ID that has the minimum required authority of **CONNECT**. To schedule the job on the Sales database Alan also needs the user credentials of a user ID that has the authorizations on the database required by the actions that the script runs. That user ID also needs the required authority to run the cleanup job afterward.

Alan then opens the Job Manager page in the web console, and clicks **Add Job** in the Job List tab to create the job. After filling out the basic information, such as a job name and a description of the job, Alan selects the correct type of job to match the script and verifies that the job is enabled for scheduling.

Working through the new job wizard, Alan pastes in the script that Doug provided into the Script component of the job, making sure that the closing character defined for the job matches what is in the script.

Alan then creates a schedule from the Schedules component of the job, setting a date and time for the first job run, and configuring it to run monthly on the **Sales** database. As the user ID used in the database connection does not have the correct authority to run some of the commands in the script, Alan selects to run the job as the specific user ID with the correct authority that was provided by the database owner.

Alan also adds the requested cleanup job to the job in the Chain component. As the only required chained job is the cleanup, Alan adds it to run at the end of the chain.

Finally, Alan adds the email addresses of Doug and Doug's manager to the Notifications component of the job, and configures notifications to be sent if the job fails.

The job is now scheduled, and Alan can view the job, schedule, and notification information for the job in the corresponding job manager tabs. Once the job has been run, any user with access to the web console can use the History page to view the job history for the job, and get a detailed view by looking at the log entry for the job. If Doug does not have access to the web console, Alan adds Doug as a repository database user and uses the Console Security page to grant Doug access to the web console.

Importing tasks from DB2 Task Center

Use the Data Studio web console to import existing tasks from the Task Center in the DB2 Control Center. Imported tasks are saved as jobs in the job manager.

About this task

The imported tasks are mapped to the appropriate job manager type as shown in the following table:

Table 11. Mapping of Task Center script type to Job Manager job type

Task Center script type	Job Manager job type
DB2 command script	DB2 CLP script
OS command script	Shell/Executable script

Restrictions: The following restrictions apply to importing tasks from the DB2 Task Center:

- Task types from DB2 Task Center:

Table 12. Restrictions for task types from DB2 Task Center

Task type	Restrictions
MVS shell script	Not supported.
Grouping	Not supported.
OS command script	The script interpreters and command execution parameters are not supported. The default script interpreter is used instead.
DB2 command script	Supported.

- Schedules that are associated with tasks from DB2 Task Center:

Table 13. Restrictions for schedules from DB2 Task Center

Schedule type	Restrictions
Weekly	Only schedules set for 1 to 4 weeks are supported.
Monthly	Only schedules set for 1 month and schedules set to a specific date or last date are supported.
Yearly	Only schedules set for 1 year are supported.
Expired (that is, schedules with a starting or ending time that is earlier than the current time)	Expired schedules will be imported but marked as inactive.

- Task actions that are associated with tasks from DB2 Task Center:

Table 14. Restrictions for task actions from DB2 Task Center

Task action	Restrictions
Run task	Only the first Run task task action associated with the task will be imported.
Enable schedule of	Not supported.
Disable schedule of	Not supported.
Delete this task	Not supported.

- The success code sets that are used by the DB2 Task Center when running tasks are ignored by the job manager.
- If the tools catalog database contains a task that was previously imported to the Data Studio web console and you choose to import the task again, the task is saved as a new job with a new job ID.
- Contact lists are not imported from the DB2 Task Center.

Procedure

To import tasks from the DB2 Task Center:

1. Open the Data Studio web console in a web browser.
2. To open the Import Tasks page, from the **Open** menu, click **Product Setup > Import Tasks**.
3. Follow the instructions on the Import Tasks page to start importing tasks. You must specify a valid tools catalog database that contains the DB2 Task Center metadata, and then select the tasks to import. Only supported tasks from the tools catalog database are enabled in the Import Tasks page.

Results

If the task is imported successfully, a new job is created for the imported task in the job manager with a job name that is identical to the task name of the imported task. The job name is prefixed by "TC_toolsdb_" where *toolsdb* is the name of the DB2 tools database. The script of the imported task is not modified.

If the imported task is associated with a schedule in the Task Center, a new schedule is created for the corresponding job by the job manager and the tools catalog database is associated with the schedule by default. The schedule date format for the imported task is converted to the job manager schedule format.

What to do next

If the job that was generated from the imported task is not associated with a schedule, create a schedule and add the job to the schedule.

Diagramming access plans with Visual Explain

You can generate a diagram of the current access plan for an SQL or XPATH statement to find out how your data server processes the statement. You can use the information available from the graph to tune your SQL statements for better performance.

Before you begin

If you want to create access plan diagrams for DB2 for z/OS, you must configure the DB2 subsystem that you are using. The steps are identical to the steps for configuring a subsystem for use with the no-charge tuning features that are in IBM Data Studio.

Restriction: For IBM Informix® Dynamic Server, Visual Explain cannot explain SELECT statements that contain parameter markers or host variables.

About this task

You can use Visual Explain to:

- View the statistics that were used at the time of optimization. You can then compare these statistics to the current catalog statistics to help you determine whether rebinding the package might improve performance.
- Determine whether or not an index was used to access a table. If an index was not used, Visual Explain can help you determine which columns might benefit from being indexed.
- Obtain information about each operation in the access plan, including the total estimated cost and number of rows retrieved (cardinality).

Procedure

To generate the diagram of the current access plan for a query:

1. Optional: Set preferences for how Visual Explain operates and for how it displays diagrams.
2. Follow one of these steps:
 - In the Data Project Explorer, right-click an SQL statement, SQL stored procedure, or SQL user-defined function, and select **Open Visual Explain**.
 - In the Data Source Explorer, right-click a view or right-click an SQL stored procedure or SQL user-defined function that contains an INSERT, UPDATE, DELETE, or SELECT statement. Select **Open Visual Explain**. If the workbench finds more than one SQL statement or XQUERY statement, the workbench uses the first statement.
 - In an SQL, Routine, or Java editor, highlight and right-click the INSERT, UPDATE, DELETE, or SELECT statement, XPATH, or XQUERY statement and select **Open Visual Explain**.

Attempts to open Visual Explain from an SQL statement in a Java editor fail if the SQL statement contains variables that are declared in your application. For example, this SQL statement cannot be analyzed by Visual Explain because of the two variables in the predicate:

```
select count(*), sum(order.price)
from order
where order.date > var_date_1
and order.date < var_date_2
```

However, after you bind or deploy the application, you can use InfoSphere Optim Query Tuner or the single-query tuning features in Data Studio to capture the SQL statement from a DB2 package or from the dynamic statement cache and then tune it.

Note: Visual Explain is disabled or throws an exception if the selected SQL statement or object is not explainable. Only the SQL statements in the following list can be explained by Visual Explain:

- For DB2 for Linux, UNIX, and Windows: CALL, Compound SQL (Dynamic), DELETE, INSERT, MERGE, REFRESH, SELECT, SELECT INTO, SET INTEGRITY, UPDATE, VALUES, or VALUES INTO.
 - For DB2 for z/OS: SELECT, INSERT, or the searched form of an UPDATE or DELETE statement.
3. On the first page of the wizard, specify the terminator of the SQL, XPATH, or XQUERY statement that you want to diagram the access plan for.
 4. Optional: On the first page of the wizard, you can also specify settings for various options.
 - a. Specify whether you want to store the collected explain data in explain tables. If you choose this option, Visual Explain does not have to collect explain data the next time that you want to diagram the access plan for the same statement.
 - b. Specify the directory that you want Visual Explain to use as a working directory.
 - c. If IBM Support needs a trace, specify whether to trace the creation of the diagram of the access plan and whether to trace the collection of the explain data.
 - d. Specify whether to save your settings as the defaults for all diagrams that you create with Visual Explain. You can change these defaults with the Preferences window.
 5. On the second page of the wizard, set values for the special registers to customize the runtime environment to influence the collection of explain data. When Visual Explain runs the statement to gather explain data, it uses the values that you specify.
- Attention:** Please be aware of the following information regarding DB2 data servers.
- **For DB2 for z/OS:** If you specify different values for CURRENT SCHEMA and CURRENT SQLID, Visual Explain searches for explain tables that are qualified by the value of CURRENT SQLID. If Visual Explain does not find explain tables that are qualified by the value of CURRENT SQLID, Visual Explain attempts to create the explain tables under that value.
 - **For DB2 for Linux, UNIX, and Windows:** If you change the value of CURRENT SCHEMA to a value that contains special characters, you must delimit the value with single quotation marks.
 - **For DB2 for Linux, UNIX, and Windows:** Select the **Collect column and column group statistics** check box if you want Visual Explain to collect detailed statistics about clustered columns and columns that participate in a GROUP BY clause.
6. Optional: On the second page of the wizard, specify whether to save your settings as the defaults for all diagrams that you create with Visual Explain. You can change these defaults with the Preferences window.
 7. Click **Finish** to close the wizard and to generate the diagram.

Results

The workbench displays the diagram in the Access Plan Diagram view. In this view, you can navigate through the diagram, view descriptions of the nodes in the diagram, and search for nodes.

Diagrams of access plans

When DB2 processes a query, the DB2 optimizer generates several alternative plans for accessing the requested data. The optimizer estimates the execution cost of each plan and chooses the lowest-cost plan to execute. This plan is called the access plan.

Visual Explain graphically displays the access plan for any explainable statement. This display is called an access plan diagram, and it illustrates how DB2 accesses the data for a specified SQL statement.

The access plan diagram consists of nodes and lines that connect those nodes. The nodes represent data sources, operators, SQL statements, and query blocks. Nodes can have only one parent node, but they can have unlimited child nodes. The arrows on the edges indicate the direction of the flow. Usually, a table node is at the bottom of the graph, and the access plan proceeds upward from there.

Some operations in the access plan, such as nested loop joins or index scans, are represented in the graph by groups of nodes, which are called constructs. Many of these constructs have a defining node that indicates the operation. For example, the HBJOIN node indicates that a hybrid join operation is taking place, but the entire hybrid join is represented in the graph by a group of nodes. This group of nodes represents all of the other data sources and operations that are involved in the hybrid join.

Query blocks

An SQL statement can consist of several subqueries, which are represented in the access plan diagram by query blocks.

The subquery can be a SELECT, INSERT, UPDATE, or DELETE. A subquery can contain other subqueries in the FROM clause, the WHERE clause, or a subselect of a UNION or UNION ALL. A subquery within another subquery is called a child subquery. A subquery that contains another subquery is called a parent subquery. This parent-child relationship can be represented by a tree hierarchy.

If a subquery references at least one column of its parent subquery or of any parent subqueries that are higher up in the tree hierarchy, the subquery is a correlated subquery; otherwise it is a non-correlated subquery. A non-correlated subquery can run at the same time as the highest parent subquery that is also non-correlated. This highest parent subquery is called the "do-at-open parent subquery" in terms of its relationship to the non-correlated subquery. The execution of a correlated subquery is bound to the execution of its parent subquery. Such relationships between the relative executions of parents and children can be represented by separate trees hierarchies in the access plan graph.

Non-correlated subquery

For a non-correlated subquery, the query block node is connected to the right of the query block node for the highest parent subquery that is also non-correlated.

Correlated subquery

For a correlated subquery, the query block node is connected to the part within its parent subquery where the correlated subquery is executed.

Setting preferences for Visual Explain

Use the Preferences window to set default values for settings that determine how Visual Explain operates and how it displays diagrams.

Procedure

To set preferences for Visual Explain:

1. Select **Window > Preferences**.
2. In the tree view of the Preferences window, select **Data > Visual Explain**.
3. On the Visual Explain page, set the following options:
 - a. Specify whether to launch the Visual Explain wizard when you right-click an SQL statement, view, stored procedure, or user-defined function and select **Visual Explain**. The wizard allows you to override preferences. If you clear this option, Visual Explain uses the preferences.
 - b. If your project is associated with a DB2 data server, specify whether Visual Explain saves in the explain tables the explain data that it collects for the statement.
4. On the **Query Explain Settings** page, specify default values for special registers. Changing these values modifies how Visual Explain gathers explain data to use when generating the access plan diagram.

Attention: Please be aware of the following information regarding DB2 data servers.

- **For DB2 for z/OS:** If you specify different values for CURRENT SCHEMA and CURRENT SQLID, Visual Explain searches for explain tables that are qualified by the value of CURRENT SQLID. If Visual Explain does not find explain tables that are qualified by the value of CURRENT SQLID, Visual Explain attempts to create the explain tables under that value.
- **For DB2 for Linux, UNIX, and Windows:** If you change the value of CURRENT SCHEMA to a value that contains special characters, you must delimit the value with single quotation marks.
- **For DB2 for Linux, UNIX, and Windows:** Select the **Collect column and column group statistics** check box if you want Visual Explain to collect detailed statistics about clustered columns and columns that participate in a GROUP BY clause.

5. On the **Viewer** page, change various behaviors and colors of diagrams.
6. On the **Nodes** page, change the default appearance of nodes. You can change the text, color, and shape of the different types of nodes. You can also choose whether to highlight selected nodes, shadow nodes, or show information about nodes when you move your mouse cursor over them.

Part 2. Data partitioning and clustering

With the introduction of table partitioning, a DB2 database offers a three-level data organization scheme. There are three clauses of the CREATE TABLE statement that include an algorithm to indicate how the data is to be organized.

The following three clauses demonstrate the levels of data organization that can be used together in any combination:

- **DISTRIBUTE BY** to spread data evenly across database partitions (to enable intraquery parallelism and to balance the load across each database partition) (database partitioning)
- **PARTITION BY** to group rows with similar values of a single dimension in the same data partition (table partitioning)
- **ORGANIZE BY** to group rows with similar values on multiple dimensions in the same table extent (multidimensional clustering) or to group rows according to the time of the insert operation (insert time clustering table).

This syntax allows consistency between the clauses and allows for future algorithms of data organization. Each of these clauses can be used in isolation or in combination with one another. By combining the DISTRIBUTE BY and PARTITION BY clauses of the CREATE TABLE statement data can be spread across database partitions spanning multiple table spaces. This approach allows for similar behavior to the Informix Dynamic Server and Informix Extended Parallel Server hybrid.

In a single table, you can combine the clauses used in each data organization scheme to create more sophisticated partitioning schemes. For example, partitioned database environments are not only compatible, but also complementary to table partitioning.

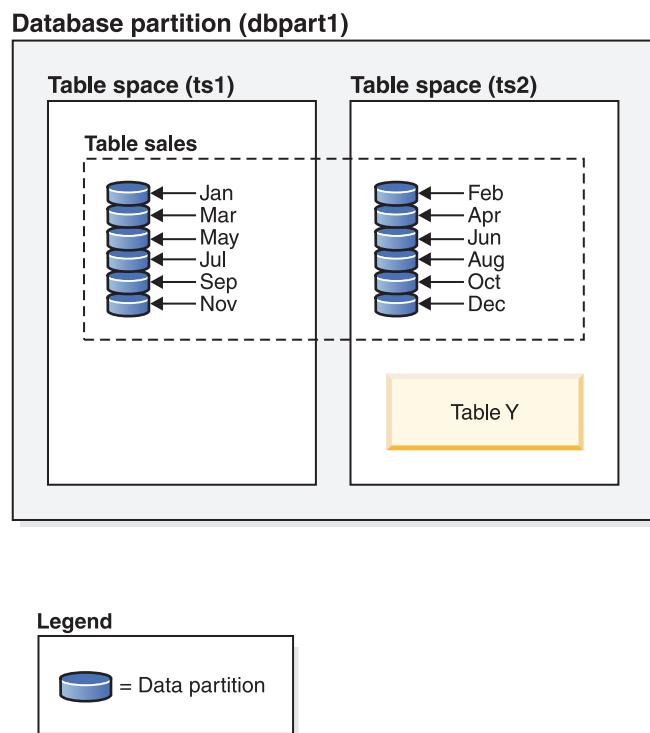


Figure 24. Demonstrating the table partitioning organization scheme where a table representing monthly sales data is partitioned into multiple data partitions. The table also spans two table spaces (ts1 and ts2).

Database partition group (dbgroup1)

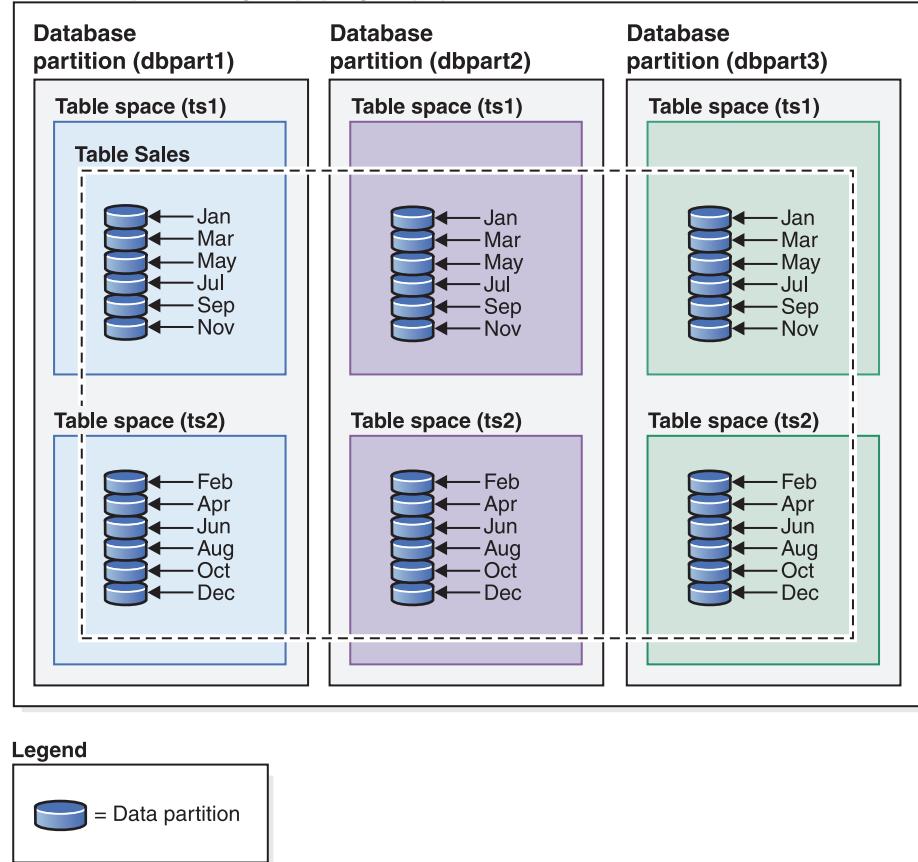


Figure 25. Demonstrating the complementary organization schemes of database partitioning and table partitioning. A table representing monthly sales data is partitioned into multiple data partitions, spanning two table spaces (ts1 and ts2) that are distributed across multiple database partitions (dbpart1, dbpart2, dbpart3) of a database partition group (dbgroup1).

The salient distinction between multidimensional clustering (MDC) and table partitioning is multi-dimension versus single dimension. MDC is suitable to cubes (that is, tables with multiple dimensions), and table partitioning works well if there is a single dimension which is central to the database design, such as a DATE column. MDC and table partitioning are complementary when both of these conditions are met. This is demonstrated in Figure 26 on page 114.

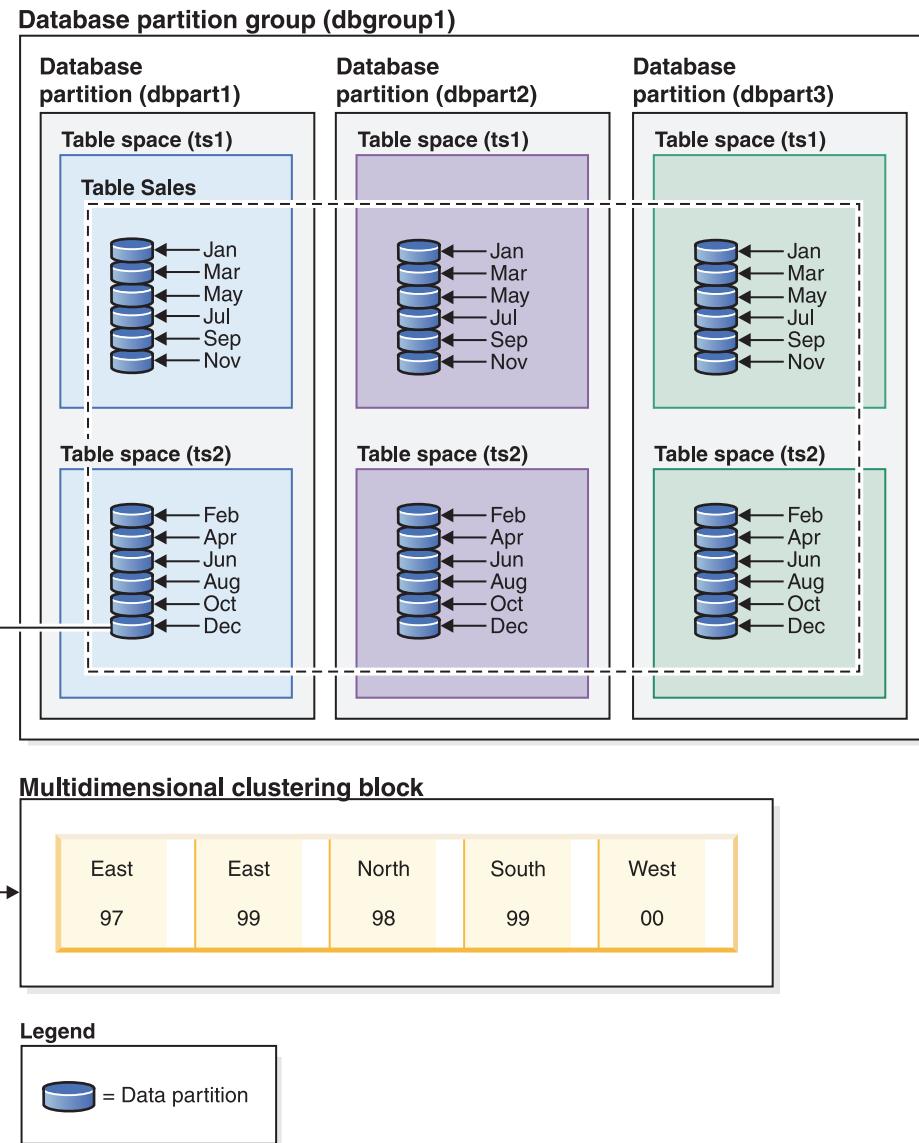


Figure 26. A representation of the database partitioning, table partitioning and multidimensional organization schemes where data from table SALES is not only distributed across multiple database partitions, partitioned across table spaces ts1 and ts2, but also groups rows with similar values on both the date and region dimensions.

There is another data organization scheme which cannot be used with any of the schemes that were listed previously. This scheme is ORGANIZE BY KEY SEQUENCE. It is used to insert each record into a row that was reserved for that record at the time of table creation (Range-clustered table).

Data organization terminology

Database partitioning

A data organization scheme in which table data is divided across multiple database partitions based on the hash values in one or more distribution key columns of the table, and based on the use of a distribution map of the database partitions. Data from a given table is distributed based on the specifications provided in the **DISTRIBUTE BY HASH** clause of the **CREATE TABLE** statement.

Database partition

A portion of a database on a database partition server consisting of its own user data, indexes, configuration file, and transaction logs. Database partitions can be logical or physical.

Table partitioning

A data organization scheme in which table data is divided across multiple data partitions according to values in one or more partitioning columns of the table. Data from a given table is partitioned into multiple storage objects based on the specifications provided in the PARTITION BY clause of the CREATE TABLE statement. These storage objects can be in different table spaces.

Data partition

A set of table rows, stored separately from other sets of rows, grouped by the specifications provided in the PARTITION BY RANGE clause of the CREATE TABLE statement.

Multidimensional clustering (MDC)

A table whose data is physically organized into blocks along one or more dimensions, or clustering keys, specified in the ORGANIZE BY DIMENSIONS clause.

Insert time clustering (ITC)

A table whose data is physically clustered based on row insert time, specified by the ORGANIZE BY INSERT TIME clause.

Benefits of each data organization scheme

Understanding the benefits of each data organization scheme can help you to determine the best approach when planning, designing, or reassessing your database system requirements. Table 15 provides a high-level view of common customer requirements and shows how the various data organization schemes can help you to meet those requirements.

Table 15. Using table partitioning with the Database Partitioning Feature

Issue	Recommended scheme	Explanation
Data roll-out	Table partitioning	Uses detach to roll out large amounts of data with minimal disruption
Parallel query execution (query performance)	Database Partitioning Feature	Provides query parallelism for improved query performance
Data partition elimination (query performance)	Table partitioning	Provides data partition elimination for improved query performance
Maximization of query performance	Both	Maximum query performance when used together: query parallelism and data partition elimination are complementary
Heavy administrator workload	Database Partitioning Feature	Execute many tasks for each database partition

Table 16. Using table partitioning with MDC tables

Issue	Recommended scheme	Explanation
Data availability during roll-out	Table partitioning	Use the DETACH PARTITION clause to roll out large amounts of data with minimal disruption.
Query performance	Both	MDC is best for querying multiple dimensions. Table partitioning helps through data partition elimination.
Minimal reorganization	MDC	MDC maintains clustering, which reduces the need to reorganize.

Note: Table partitioning is now recommended over UNION ALL views.

Chapter 7. Partitioned database environments

A partitioned database environment is a database installation that supports the distribution of data across database partitions.

- A *database partition* is a part of a database that consists of its own data, indexes, configuration files, and transaction logs. A partitioned database environment is a database installation that supports the distribution of data across database partitions.
- A *single-partition database* is a database having only one database partition. All data in the database is stored in that single database partition. In this case, database partition groups, although present, provide no additional capability.
- A *multi-partition database* is a database with two or more database partitions. Tables can be located in one or more database partitions. When a table is in a database partition group consisting of multiple database partitions, some of its rows are stored in one database partition, and other rows are stored in other database partitions.

Usually, a single database partition exists on each physical machine, and the processors on each system are used by the database manager at each database partition to manage its part of the total data in the database.

Because data is distributed across database partitions, you can use the power of multiple processors on multiple physical machines to satisfy requests for information. Data retrieval and update requests are decomposed automatically into sub-requests, and executed in parallel among the applicable database partitions. The fact that databases are split across database partitions is transparent to users issuing SQL statements.

User interaction occurs through one database partition, known as the *coordinator partition* for that user. The coordinator partition runs on the same database partition as the application, or in the case of a remote application, the database partition to which that application is connected. Any database partition can be used as a coordinator partition.

The database manager allows you to store data across several database partitions in the database. This means that the data is physically stored across more than one database partition, and yet can be accessed as though it were located in the same place. Applications and users accessing data in a multi-partition database are unaware of the physical location of the data.

Although the data is physically split, it is used and managed as a logical whole. Users can choose how to distribute their data by declaring distribution keys. Users can also determine across which and over how many database partitions their data is distributed by selecting the table space and the associated database partition group in which the data is to be stored. Suggestions for distribution and replication can be done using the DB2 Design Advisor. In addition, an updatable distribution map is used with a hashing algorithm to specify the mapping of distribution key values to database partitions, which determines the placement and retrieval of each row of data. As a result, you can spread the workload across a multi-partition database for large tables, and store smaller tables on one or more database partitions. Each database partition has local indexes on the data it stores, resulting in increased performance for local data access.

Note: You are not restricted to having all tables divided across all database partitions in the database. The database manager supports *partial declustering*, which means that you can divide tables and their table spaces across a subset of database partitions in the system.

An alternative to consider when you want tables to be positioned on each database partition, is to use materialized query tables and then replicate those tables. You can create a materialized query table containing the information that you need, and then replicate it to each database partition.

A non-root installation of a DB2 database product does not support database partitioning. Do not manually update the db2nodes.cfg file. A manual update returns an error (SQL6031N).

Database partitioning across multiple database partitions

The database manager allows great flexibility in spreading data across multiple database partitions of a partitioned database.

Users can choose how to distribute their data by declaring distribution keys, and can determine which and how many database partitions their table data can be spread across by selecting the database partition group and table space in which the data is to be stored.

In addition, a distribution map (which is updatable) specifies the mapping of distribution key values to database partitions. This makes it possible for flexible workload parallelization across a partitioned database for large tables, while allowing smaller tables to be stored on one or a small number of database partitions if the application designer so chooses. Each local database partition can have local indexes on the data it stores to provide high performance local data access.

In a partitioned database, the distribution key is used to distribute table data across a set of database partitions. Index data is also partitioned with its corresponding tables, and stored locally at each database partition.

Before database partitions can be used to store data, they must be defined to the database manager. Database partitions are defined in a file called db2nodes.cfg.

The distribution key for a table in a table space on a partitioned database partition group is specified in the CREATE TABLE statement or the ALTER TABLE statement. If not specified, a distribution key for a table is created by default from the first column of the primary key. If no primary key is defined, the default distribution key is the first column defined in that table that has a data type other than a long or a LOB data type. Tables in partitioned databases must have at least one column that is neither a long nor a LOB data type. A table in a table space that is in a single partition database partition group will have a distribution key only if it is explicitly specified.

Rows are placed in a database partition as follows:

1. A hashing algorithm (database partitioning function) is applied to all of the columns of the distribution key, which results in the generation of a distribution map index value.
2. The database partition number at that index value in the distribution map identifies the database partition in which the row is to be stored.

The database manager supports *partial declustering*, which means that a table can be distributed across a subset of database partitions in the system (that is, a database partition group). Tables do not have to be distributed across all of the database partitions in the system.

The database manager has the capability of recognizing when data being accessed for a join or a subquery is located at the same database partition in the same database partition group. This is known as *table collocation*. Rows in colocated tables with the same distribution key values are located on the same database partition. The database manager can choose to perform join or subquery processing at the database partition in which the data is stored. This can have significant performance advantages.

Collocated tables must:

- Be in the same database partition group, one that is not being redistributed. (During redistribution, tables in the database partition group might be using different distribution maps - they are not colocated.)
- Have distribution keys with the same number of columns.
- Have the corresponding columns of the distribution key be database partition-compatible.
- Be in a single partition database partition group defined on the same database partition.

Database partition groups

A database partition group is a named set of one or more database partitions that belong to a database.

A database partition group that contains more than one database partition is known as a *multiple partition database partition group*. Multiple partition database partition groups can only be defined with database partitions that belong to the same instance.

Figure 27 on page 120 shows an example of a database with five database partitions.

- Database partition group 1 contains all but one of the database partitions.
- Database partition group 2 contains one database partition.
- Database partition group 3 contains two database partitions.
- The database partition in Group 2 is shared (and overlaps) with Group 1.
- A single database partition in Group 3 is shared (and overlaps) with Group 1.

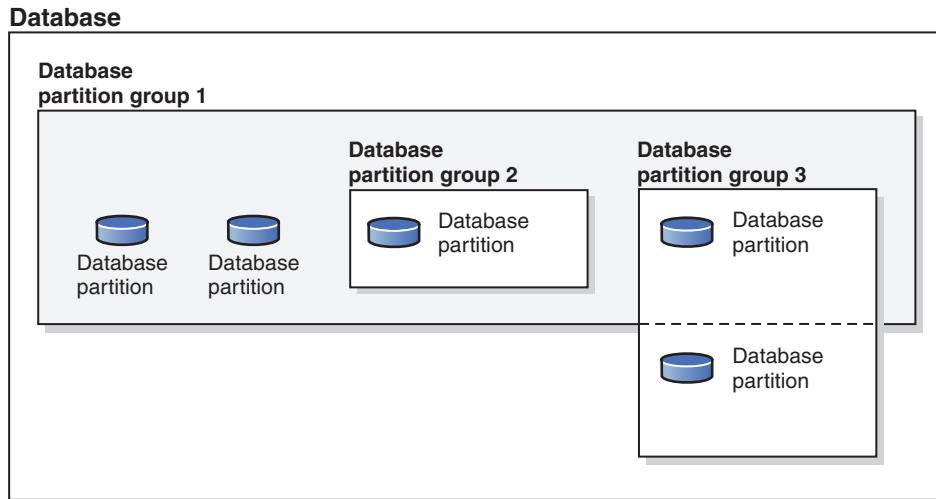


Figure 27. Database partition groups in a database

When a database is created, all database partitions that are specified in the *database partition configuration file* named db2nodes.cfg are created as well. Other database partitions can be added or removed with the **ADD DBPARTITIONNUM** or **DROP DBPARTITIONNUM VERIFY** command, respectively. Data is divided across all of the database partitions in a database partition group.

When a database partition group is created, a *distribution map* is associated with the group. The distribution map, along with a *distribution key* and a hashing algorithm are used by the database manager to determine which database partition in the database partition group will store a given row of data.

Default database partition groups

Three database partition groups are defined automatically at database creation time:

- IBMCATGROUP for the SYSCATSPACE table space, holding system catalog tables
- IBMTEMPGROUP for the TEMPSPACE1 table space, holding temporary tables created during database processing
- IBMDEFAULTGROUP for the USERSPACE1 table space, holding user tables and indexes. A user temporary table space for a declared temporary table or a created temporary table can be created in IBMDEFAULTGROUP or any user-created database partition group, but not in IBMTEMPGROUP.

Table spaces in database partition groups

When a table space is associated with a multiple partition database partition group (during execution of the CREATE TABLESPACE statement), all of the tables within that table space are partitioned across each database partition in the database partition group. A table space that is associated with a particular database partition group cannot later be associated with another database partition group.

Creating a database partition group

Create a database partition group by using the CREATE DATABASE PARTITION GROUP statement. This statement specifies the set of database partitions on which the table space containers and table data are to reside. This statement also performs the following actions:

- It creates a distribution map for the database partition group.
- It generates a distribution map ID.
- It inserts records into the following catalog views:
 - SYSCAT.DBPARTITIONGROUPDEF
 - SYSCAT.DBPARTITIONGROUPS
 - SYSCAT.PARTITIONMAPS

Altering a database partition group

Use the ALTER DATABASE PARTITION GROUP statement to add database partitions to (or drop them from) a database partition group. After adding or dropping database partitions, use the **REDISTRIBUTE DATABASE PARTITION GROUP** command to redistribute the data across the set of database partitions in the database partition group.

Database partition group design considerations

Place small tables in single-partition database partition groups, except when you want to take advantage of collocation with a larger table. *Collocation* is the placement of rows from different tables that contain related data in the same database partition. Collocated tables help the database manager to use more efficient join strategies. Such tables can exist in a single-partition database partition group. Tables are considered to be collocated if they are in a multiple partition database partition group, have the same number of columns in the distribution key, and if the data types of corresponding columns are compatible. Rows in collocated tables with the same distribution key value are placed on the same database partition. Tables can be in separate table spaces in the same database partition group, and still be considered collocated.

Avoid extending medium-sized tables across too many database partitions. For example, a 100-MB table might perform better on a 16-partition database partition group than on a 32-partition database partition group.

You can use database partition groups to separate online transaction processing (OLTP) tables from decision support (DSS) tables. This will help to ensure that the performance of OLTP transactions is not adversely affected.

If you are using a multiple partition database partition group, consider the following points:

- In a multiple partition database partition group, you can only create a unique index if the index is a superset of the distribution key.
- Each database partition must be assigned a unique number, because the same database partition might be found in one or more database partition groups.
- To ensure fast recovery of a database partition containing system catalog tables, avoid placing user tables on the same database partition. Place user tables in database partition groups that do not include the database partition in the IBMCATGROUP database partition group.

Distribution maps

In a partitioned database environment, the database manager must know where to find the data that it needs. The database manager uses a map, called a *distribution map*, to find the data.

A distribution map is an internally generated array containing either 32 768 entries for multiple-partition database partition groups, or a single entry for single-partition database partition groups. For a single-partition database partition group, the distribution map has only one entry containing the number of the database partition where all the rows of a database table are stored. For multiple-partition database partition groups, the numbers of the database partition group are specified in a way such that each database partition is used one after the other to ensure an even distribution across the entire map. Just as a city map is organized into sections using a grid, the database manager uses a *distribution key* to determine the location (the database partition) where the data is stored.

For example, assume that you have a database on four database partitions (numbered 0-3). The distribution map for the IBMDEFAULTGROUP database partition group of this database is:

0 1 2 3 0 1 2 ...

If a database partition group had been created in the database using database partitions 1 and 2, the distribution map for that database partition group is:

1 2 1 2 1 2 1 ...

If the distribution key for a table to be loaded into the database is an integer with possible values between 1 and 500 000, the distribution key is hashed to a number between 0 and 32 767. That number is used as an index into the distribution map to select the database partition for that row.

Figure 28 shows how the row with the distribution key value (c1, c2, c3) is mapped to number 2, which, in turn, references database partition n5.

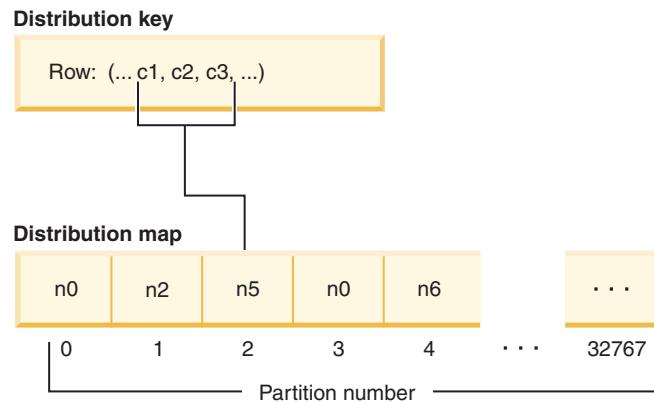


Figure 28. Data distribution using a distribution map

A distribution map is a flexible way of controlling where data is stored in a multi-partition database. If you must change the data distribution across the database partitions in your database, you can use the data redistribution utility. This utility allows you to rebalance or introduce skew into the data distribution.

You can use the db2GetDistMap API to obtain a copy of a distribution map that you can view. If you continue to use the sqlugtpi API to obtain the distribution information, this API might return error message SQL2768N, because it can only retrieve distribution maps containing 4096 entries.

Distribution keys

A *distribution key* is a column (or group of columns) that is used to determine the database partition in which a particular row of data is stored.

A distribution key is defined on a table using the CREATE TABLE statement. If a distribution key is not defined for a table in a table space that is divided across more than one database partition in a database partition group, one is created by default from the first column of the primary key.

If no primary key is specified, the default distribution key is the first non-long field column defined on that table. (*Long* includes all long data types and all large object (LOB) data types). If you are creating a table in a table space associated with a single-partition database partition group, and you want to have a distribution key, you must define the distribution key explicitly. One is not created by default.

If no columns satisfy the requirement for a default distribution key, the table is created without one. Tables without a distribution key are only allowed in single-partition database partition groups. You can add or drop distribution keys later, using the ALTER TABLE statement. Altering the distribution key can only be done to a table whose table space is associated with a single-partition database partition group.

Choosing a good distribution key is important. Take into consideration:

- How tables are to be accessed
- The nature of the query workload
- The join strategies employed by the database system

If collocation is not a major consideration, a good distribution key for a table is one that spreads the data evenly across all database partitions in the database partition group. The distribution key for each table in a table space that is associated with a database partition group determines if the tables are collocated. Tables are considered collocated when:

- The tables are placed in table spaces that are in the same database partition group
- The distribution keys in each table have the same number of columns
- The data types of the corresponding columns are partition-compatible.

These characteristics ensure that rows of collocated tables with the same distribution key values are located on the same database partition.

An inappropriate distribution key can cause uneven data distribution. Do not choose columns with unevenly distributed data or columns with a small number of distinct values for the distribution key. The number of distinct values must be great enough to ensure an even distribution of rows across all database partitions in the database partition group. The cost of applying the distribution algorithm is proportional to the size of the distribution key. The distribution key cannot be more than 16 columns, but fewer columns result in better performance. Do not include unnecessary columns in the distribution key.

Consider the following points when defining a distribution key:

- Creation of a multiple-partition table that contains only BLOB, CLOB, DBCLOB, LONG VARCHAR, LONG VARGRAPHIC, XML, or structured data types is not supported.
- The distribution key definition cannot be altered.
- Include the most frequently joined columns in the distribution key.
- Include columns that often participate in a GROUP BY clause in the distribution key.
- Any unique key or primary key must contain all of the distribution key columns.
- In an online transaction processing (OLTP) environment, ensure that all columns in the distribution key participate in a transaction through equality predicates. For example, assume that you have an employee number column, EMP_NO, that is often used in transactions such as:

```
UPDATE emp_table SET ... WHERE  
emp_no = host-variable
```

In this case, the EMP_NO column makes a good single column distribution key for EMP_TABLE.

Database partitioning is the method by which the placement of each row in the table is determined. The method works as follows:

1. A hashing algorithm is applied to the value of the distribution key, and generates a number between zero (0) and 32 767.
2. The distribution map is created when a database partition group is created. Each of the numbers is sequentially repeated in a round-robin fashion to fill the distribution map.
3. The number is used as an index into the distribution map. The number at that location in the distribution map is the number of the database partition where the row is stored.

Table collocation

If two or more tables frequently contribute data in response to certain queries, you will want related data from these tables to be physically located as close together as possible. In a partitioned database environment, this process is known as *table collocation*.

Tables are collocated when they are stored in the same database partition group, and when their distribution keys are compatible. Placing both tables in the same database partition group ensures a common distribution map. The tables might be in different table spaces, but the table spaces must be associated with the same database partition group. The data types of the corresponding columns in each distribution key must be *partition-compatible*.

When more than one table is accessed for a join or a subquery, the database manager determines whether the data to be joined is located at the same database partition. When this happens, the join or subquery is performed at the database partition where the data is stored, instead of having to move data between database partitions. This ability has significant performance advantages.

Partition compatibility

The base data types of corresponding columns of distribution keys are compared and can be declared *partition-compatible*. Partition-compatible data types have the

property that two variables, one of each type, with the same value, are mapped to the same number by the same partitioning algorithm.

Partition-compatibility has the following characteristics:

- A base data type is compatible with another of the same base data type.
- Internal formats are used for DATE, TIME, and TIMESTAMP data types. They are not compatible with each other, and none are compatible with character or graphic data types.
- Partition compatibility is not affected by the nullability of a column.
- Partition-compatibility is affected by collation. Locale-sensitive UCA-based collations require an exact match in collation, except that the strength (S) attribute of the collation is ignored. All other collations are considered equivalent for the purposes of determining partition compatibility.
- Character columns defined with FOR BIT DATA are only compatible with character columns without FOR BIT DATA when a collation other than a locale-sensitive UCA-based collation is used.
- NULL values of compatible data types are treated identically; those of non-compatible data types might not be.
- Base data types of a user-defined type are used to analyze partition-compatibility.
- Decimals of the same value in the distribution key are treated identically, even if their scale and precision differ.
- Trailing blanks in character strings (CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC) are ignored by the hashing algorithm.
- BIGINT, SMALLINT, and INTEGER are compatible data types.
- When a locale-sensitive UCA-based collation is used, CHAR, VARCHAR, GRAPHIC, and VARGRAPHIC are compatible data types. When another collation is used, CHAR and VARCHAR of different lengths are compatible types and GRAPHIC and VARGRAPHIC are compatible types, but CHAR and VARCHAR are not compatible types with GRAPHIC and VARGRAPHIC.
- Partition-compatibility does not apply to LONG VARCHAR, LONG VARGRAPHIC, CLOB, DBCLOB, and BLOB data types, because they are not supported as distribution keys.

Setting up partitioned database environments

The decision to create a multi-partition database must be made before you create your database. As part of the database design decisions you make, you will have to determine if you should take advantage of the performance improvements database partitioning can offer.

About this task

In a partitioned database environment, you still use the **CREATE DATABASE** command or the `sqlecrea()` function to create a database. Whichever method is used, the request can be made through any of the partitions listed in the `db2nodes.cfg` file. The `db2nodes.cfg` file is the database partition server configuration file.

Except on the Windows operating system environment, any editor can be used to view and update the contents of the database partition server configuration file (`db2nodes.cfg`). On the Windows operating system environment, use **db2ncrt** and **db2nchg** commands to create and change the database partition server configuration file.

Before creating a multi-partition database, you must select which database partition will be the catalog partition for the database. You can then create the database directly from that database partition, or from a remote client that is attached to that database partition. The database partition to which you attach and execute the **CREATE DATABASE** command becomes the *catalog partition* for that particular database.

The catalog partition is the database partition on which all system catalog tables are stored. All access to system tables must go through this database partition. All federated database objects (for example, wrappers, servers, and nicknames) are stored in the system catalog tables at this database partition.

If possible, you should create each database in a separate instance. If this is not possible (that is, you must create more than one database per instance), you should spread the catalog partitions among the available database partitions. Doing this reduces contention for catalog information at a single database partition.

Note: You should regularly do a backup of the catalog partition and avoid putting user data on it (whenever possible), because other data increases the time required for the backup.

When you create a database, it is automatically created across all the database partitions defined in the db2nodes.cfg file.

When the first database in the system is created, a system database directory is formed. It is appended with information about any other databases that you create. When working on UNIX, the system database directory is sqldbdirect and is located in the sql1lib directory under your home directory, or under the directory where DB2 database was installed. When working on UNIX, this directory must reside on a shared file system, (for example, NFS on UNIX platforms) because there is only one system database directory for all the database partitions that make up the partitioned database environment. When working on Windows, the system database directory is located in the instance directory.

Also resident in the sqldbdirect directory is the system intention file. It is called sqldbins, and ensures that the database partitions remain synchronized. The file must also reside on a shared file system since there is only one directory across all database partitions. The file is shared by all the database partitions making up the database.

Configuration parameters have to be modified to take advantage of database partitioning. Use the **GET DATABASE CONFIGURATION** and the **GET DATABASE MANAGER CONFIGURATION** commands to find out the values of individual entries in a specific database, or in the database manager configuration file. To modify individual entries in a specific database, or in the database manager configuration file, use the **UPDATE DATABASE CONFIGURATION** and the **UPDATE DATABASE MANAGER CONFIGURATION** commands respectively.

The database manager configuration parameters affecting a partitioned database environment include **conn_elapse**, **fcm_num_buffers**, **fcm_num_channels**, **max_connretries**, **max_coordagents**, **max_time_diff**, **num_poolagents**, and **start_stop_time**.

Adding database partition servers to an instance (Windows)

On Windows, use the **db2ncrt** command to add a database partition server to an instance.

About this task

Note: Do not use the **db2ncrt** command if the instance already contains databases. Instead, use the **START DBM ADD DBPARTITIONNUM** command. This ensures that the database is correctly added to the new database partition server. **DO NOT EDIT** the db2nodes.cfg file, since changing the file might cause inconsistencies in the partitioned database environment.

The command has the following required parameters:

```
db2ncrt /n:partition_number  
          /u:username,password  
          /p:logical_port
```

/n:partition_number

The unique database partition number to identify the database partition server. The number can be from 1 to 999 in ascending sequence.

/u:username,password

The logon account name and password of the DB2 service.

/p:logical_port

The logical port number used for the database partition server if the logical port is not zero (0). If not specified, the logical port number assigned is 0.

The logical port parameter is only optional when you create the first database partition on a computer. If you create a logical database partition, you must specify this parameter and select a logical port number that is not in use. There are several restrictions:

- On every computer there must be a database partition server with a logical port 0.
- The port number cannot exceed the port range reserved for FCM communications in the services file in %SystemRoot%\system32\drivers\etc directory. For example, if you reserve a range of four ports for the current instance, then the maximum port number would be 3 (ports 1, 2, and 3; port 0 is for the default logical database partition). The port range is defined when **db2icrt** is used with the /r:base_port, end_port parameter.

There are also several optional parameters:

/g:network_name

Specifies the network name for the database partition server. If you do not specify this parameter, DB2 uses the first IP address it detects on your system.

Use this parameter if you have multiple IP addresses on a computer and you want to specify a specific IP address for the database partition server. You can enter the network_name parameter using the network name or IP address.

/h:host_name

The TCP/IP host name that is used by FCM for internal communications if the host name is not the local host name. This parameter is required if you add the database partition server on a remote computer.

/i:instance_name

The instance name; the default is the current instance.

/m:computer_name

The computer name of the Windows workstation on which the database partition resides; the default name is the computer name of the local computer.

/o:instance_owning_computer

The computer name of the computer that is the instance-owning computer; the default is the local computer. This parameter is required when the **db2ncrt** command is invoked on any computer that is not the instance-owning computer.

For example, if you want to add a new database partition server to the instance TESTMPP (so that you are running multiple logical database partitions) on the instance-owning computer MYMACHIN, and you want this new database partition to be known as database partition 2 using logical port 1, enter:

```
db2ncrt /n:2 /p:1 /u:my_id,my_pword /i:TESTMPP  
/M:TEST /o:MYMACHIN
```

Setting up multiple logical partitions

There are several situations in which it is advantageous to have several database partition servers running on the same computer.

This means that the configuration can contain more database partitions than computers. In these cases, the computer is said to be running *multiple logical partitions* if they participate in the same instance. If they participate in different instances, this computer is not hosting multiple logical partitions.

With multiple logical partition support, you can choose from three types of configurations:

- A standard configuration, where each computer has only one database partition server
- A multiple logical partition configuration, where a computer has more than one database partition server
- A configuration where several logical partitions run on each of several computers

Configurations that use multiple logical partitions are useful when the system runs queries on a computer that has symmetric multiprocessor (SMP) architecture. The ability to configure multiple logical partitions on a computer is also useful if a computer fails. If a computer fails (causing the database partition server or servers on it to fail), you can restart the database partition server (or servers) on another computer using the **START DBM DBPARTITIONNUM** command. This ensures that user data remains available.

Another benefit is that multiple logical partitions can use SMP hardware configurations. In addition, because database partitions are smaller, you can obtain better performance when performing such tasks as backing up and restoring database partitions and table spaces, and creating indexes.

Configuring multiple logical partitions

There are two methods of configuring multiple logical partitions.

About this task

- Configure the logical partitions (database partitions) in the db2nodes.cfg file. You can then start all the logical and remote partitions with the **db2start** command or its associated API.

Note: For Windows, you must use **db2ncrt** to add a database partition if there is no database in the system; or, **db2start addnode** command if there is one or more databases. Within Windows, the db2nodes.cfg file should never be manually edited.

- Restart a logical partition on another processor on which other logical partitions are already running. This allows you to override the hostname and port number specified for the logical partition in db2nodes.cfg.

To configure a logical database partition in db2nodes.cfg, you must make an entry in the file to allocate a logical port number for the database partition. Following is the syntax you should use:

```
nodenumber hostname logical-port netname
```

For the IBM DB2 pureScale Feature, ensure there is a member with "nodenumber 0".

Note: For Windows, you must use **db2ncrt** to add a database partition if there is no database in the system; or, **db2start addnode** command if there is one or more databases. Within Windows, the db2nodes.cfg file should never be manually edited.

The format for the db2nodes.cfg file on Windows is different when compared to the same file on UNIX. On Windows, the column format is:

```
nodenumber hostname computername logical_port netname
```

Use the fully-qualified name for the hostname. The /etc/hosts file also should use the fully-qualified name. If the fully-qualified name is not used in the db2nodes.cfg file and in the /etc/hosts file, you might receive error message SQL30082N RC=3.

You must ensure that you define enough ports in the services file of the etc directory for FCM communications.

Enabling inter-partition query parallelism

Inter-partition parallelism occurs automatically based on the number of database partitions and the distribution of data across these database partitions.

About this task

You must modify configuration parameters to take advantage of parallelism within a database partition or within a non-partitioned database. For example, intra-partition parallelism can be used to take advantage of the multiple processors on a symmetric multi-processor (SMP) machine.

Procedure

- To enable parallelism when loading data:

The load utility automatically makes use of parallelism, or you can use the following parameters on the **LOAD** command:

- **CPU_PARALLELISM**

- **DISK_PARALLELISM**

In a partitioned database environment, inter-partition parallelism for data loading occurs automatically when the target table is defined on multiple database partitions. Inter-partition parallelism for data loading can be overridden by specifying **OUTPUT_DBPARTNUMS**. The load utility also intelligently enables database partitioning parallelism depending on the size of the target database partitions. **MAX_NUM_PART_AGENTS** can be used to control the maximum degree of parallelism selected by the load utility. Database partitioning parallelism can be overridden by specifying **PARTITIONING_DBPARTNUMS** when **ANYORDER** is also specified.

- To enable parallelism when creating an index:
 - The table must be large enough to benefit from parallelism
 - Multiple processors must be enabled on an SMP computer.
- To enable I/O parallelism when backing up a database or table space:
 - Use more than one target media.
 - Configure table spaces for parallel I/O by defining multiple containers, or use a single container with multiple disks, and the appropriate use of the **DB2_PARALLEL_IO** registry variable. If you want to take advantage of parallel I/O, you must consider the implications of what must be done before you define any containers. This cannot be done whenever you see a need; it must be planned for before you reach the point where you need to backup your database or table space.
 - Use the **PARALLELISM** parameter on the **BACKUP** command to specify the degree of parallelism.
 - Use the **WITH num-buffers BUFFERS** parameter on the **BACKUP** command to ensure that enough buffers are available to accommodate the degree of parallelism. The number of buffers should equal the number of target media you have plus the degree of parallelism selected plus a few extra.
- Also, use a backup buffer size that is:
 - As large as feasible. 4 MB or 8 MB (1024 or 2048 pages) is a good rule of thumb.
 - At least as large as the largest (extent size * number of containers) product of the table spaces being backed up.
- To enable I/O parallelism when restoring a database or table space:
 - Use more than one source media.
 - Configure table spaces for parallel I/O. You must decide to use this option before you define your containers. This cannot be done whenever you see a need; it must be planned for before you reach the point where you need to restore your database or table space.
 - Use the **PARALLELISM** parameter on the **RESTORE** command to specify the degree of parallelism.
 - Use the **WITH num-buffers BUFFERS** parameter on the **RESTORE** command to ensure that enough buffers are available to accommodate the degree of parallelism. The number of buffers should equal the number of target media you have plus the degree of parallelism selected plus a few extra.
- Also, use a restore buffer size that is:
 - As large as feasible. 4 MB or 8 MB (1024 or 2048 pages) is a good rule of thumb.
 - At least as large as the largest (extent size * number of containers) product of the table spaces being restored.
 - The same as, or an even multiple of, the backup buffer size.

Enabling intrapartition parallelism for queries

To enable intrapartition query parallelism, modify one or more database or database manager configuration parameters, precompile or bind options, or a special register. Alternatively, use the MAXIMUM DEGREE option on the CREATE or ALTER WORKLOAD statement, or the ADMIN_SET_INTRA_PARALLEL procedure to enable or disable intrapartition parallelism at the transaction level.

Before you begin

Use the following controls to specify what degree of intrapartition parallelism the optimizer is to use:

- CURRENT DEGREE special register (for dynamic SQL)
- DEGREE bind option (for static SQL)
- **dft_degree** database configuration parameter (provides the default value for the previous two parameters)

Use the following controls to limit the degree of intrapartition parallelism at run time. The runtime settings override the optimizer settings.

- **max_querydegree** database manager configuration parameter
- SET RUNTIME DEGREE command
- MAXIMUM DEGREE workload option

Use the following controls to enable or disable intrapartition parallelism:

- **intra_parallel** database manager configuration parameter
- ADMIN_SET_INTRA_PARALLEL stored procedure
- MAXIMUM DEGREE workload option (set to 1)

About this task

Use the **GET DATABASE CONFIGURATION** or the **GET DATABASE MANAGER CONFIGURATION** command to find the values of individual entries in a specific database or instance configuration file. To modify one or more of these entries, use the **UPDATE DATABASE CONFIGURATION** or the **UPDATE DATABASE MANAGER CONFIGURATION** command.

intra_parallel

Database manager configuration parameter that specifies whether or not the database manager can use intrapartition parallelism. The default is NO, which means that applications in this instance are run without intrapartition parallelism. For example:

```
update dbm cfg using intra_parallel yes;
get dbm cfg;
```

max_querydegree

Database manager configuration parameter that specifies the maximum degree of intrapartition parallelism that is used for any SQL statement running on this instance. An SQL statement does not use more than this value when running parallel operations within a database partition. The default is -1, which means that the system uses the degree of intrapartition parallelism that is determined by the optimizer, not the user-specified value. For example:

```
update dbm cfg using max_querydegree any;
get dbm cfg;
```

The **intra_parallel** database manager configuration parameter must also be set to YES for the value of **max_querydegree** to be used.

dft_degree

Database configuration parameter that specifies the default value for the **DEGREE** precompile or bind option and the CURRENT DEGREE special register. The default is 1. A value of -1 (or ANY) means that the system uses the degree of intrapartition parallelism that is determined by the optimizer. For example:

```
connect to sample;
update db cfg using dft_degree -1;
get db cfg;
connect reset;
```

DEGREE Precompile or bind option that specifies the degree of intrapartition parallelism for the execution of static SQL statements on a symmetric multiprocessing (SMP) system. For example:

```
connect to prod;
prep demoapp.sqc bindfile;
bind demoapp.bnd degree 2;
...
```

CURRENT DEGREE

Special register that specifies the degree of intrapartition parallelism for the execution of dynamic SQL statements. Use the SET CURRENT DEGREE statement to assign a value to the CURRENT DEGREE special register. For example:

```
connect to sample;
set current degree = '1';
connect reset;
```

The **intra_parallel** database manager configuration parameter must also be set to YES to use intrapartition parallelism. If it is set to NO, the value of this special register is ignored, and the statement will not use intrapartition parallelism. The value of the **intra_parallel** database manager configuration parameter and the CURRENT DEGREE special register can be overridden in a workload by setting the MAXIMUM DEGREE workload attribute.

MAXIMUM DEGREE

CREATE WORKLOAD statement (or ALTER WORKLOAD statement) option that specifies the maximum runtime degree of parallelism for a workload.

For example, suppose that `bank_trans` is a packaged application that mainly executes short OLTP transactions, and `bank_report` is another packaged application that runs complex queries to generate a business intelligence (BI) report. Neither application can be modified, and both are bound with degree 4 to the database. While `bank_trans` is running, it is assigned to workload `trans`, which disables intrapartition parallelism. This OLTP application will run without any performance degradation associated with intrapartition parallelism overhead. While `bank_report` is running, it is assigned to workload `bi`, which enables intrapartition parallelism and specifies a maximum runtime degree of 8. Because the compilation degree for the package is 4, the static SQL statements in this application run with only a degree of 4. If this BI application contains dynamic SQL statements, and the CURRENT DEGREE special register is set to 16, these statements run with a degree of 8.

```

connect to sample;

create workload trans
  applname('bank_trans')
  maximum degree 1
  enable;

create workload bi
  applname('bank_report')
  maximum degree 8
  enable;

connect reset;

```

ADMIN_SET_INTRA_PARALLEL

Procedure that enables or disables intrapartition parallelism for a database application. Although the procedure is called in the current transaction, it takes effect starting with the next transaction. For example, assume that the following code is part of the demoapp application, which uses the ADMIN_SET_INTRA_PARALLEL procedure with both static and dynamic SQL statements:

```

EXEC SQL CONNECT TO prod;

// Disable intrapartition parallelism:
EXEC SQL CALL SYSPROC.ADMIN_SET_INTRA_PARALLEL('NO');
// Commit so that the effect of this call
// starts in the next statement:
EXEC SQL COMMIT;

// All statements in the next two transactions run
// without intrapartition parallelism:
strcpy(stmt, "SELECT deptname FROM org");
EXEC SQL PREPARE rstmt FROM :stmt;
EXEC SQL DECLARE c1 CURSOR FOR rstmt;
EXEC SQL OPEN c1;
EXEC SQL FETCH c1 INTO :deptname;
EXEC SQL CLOSE c1;
...
// New section for this static statement:
EXEC SQL SELECT COUNT(*) INTO :numRecords FROM org;
...
EXEC SQL COMMIT;

// Enable intrapartition parallelism:
EXEC SQL CALL SYSPROC.ADMIN_SET_INTRA_PARALLEL('YES');
// Commit so that the effect of this call
// starts in the next statement:
EXEC SQL COMMIT;

strcpy(stmt, "SET CURRENT DEGREE='4'");
// Set the degree of parallelism to 4:
EXEC SQL EXECUTE IMMEDIATE :stmt;

// All dynamic statements in the next two transactions
// run with intrapartition parallelism and degree 4:
strcpy(stmt, "SELECT deptname FROM org");
EXEC SQL PREPARE rstmt FROM :stmt;
EXEC SQL DECLARE c2 CURSOR FOR rstmt;
EXEC SQL OPEN c2;
EXEC SQL FETCH c2 INTO :deptname;
EXEC SQL CLOSE c2;
...
// All static statements in the next two transactions

```

```
// run with intrapartition parallelism and degree 2:  
EXEC SQL SELECT COUNT(*) INTO :numRecords FROM org;  
...  
EXEC SQL COMMIT;
```

The degree of intrapartition parallelism for dynamic SQL statements is specified through the CURRENT DEGREE special register, and for static SQL statements, it is specified through the DEGREE bind option. The following commands are used to prepare and bind the demoapp application:

```
connect to prod;  
prep demoapp.sqc bindfile;  
bind demoapp.bnd degree 2;  
...
```

Adding database partitions in partitioned database environments

You can add database partitions to the partitioned database system either when it is running, or when it is stopped. Because adding a new server can be time consuming, you might want to do it when the database manager is already running.

Use the **ADD DBPARTITIONNUM** command to add a database partition to a system. This command can be invoked in the following ways:

- As an option on the **START DBM** command
- With the **ADD DBPARTITIONNUM** command
- With the sqleaddn API
- With the sqlepstart API

If your system is stopped, use the **START DBM** command. If it is running, you can use any of the other choices.

When you use the **ADD DBPARTITIONNUM** command to add a new database partition to the system, all existing databases in the instance are expanded to the new database partition. You can also specify which containers to use for temporary table spaces for the databases. The containers can be:

- The same as those defined for the catalog partition for each database. (This is the default.)
- The same as those defined for another database partition.
- Not created at all. You must use the ALTER TABLESPACE statement to add temporary table space containers to each database before the database can be used.

Note: Any uncataloged database is not recognized when adding a new database partition. The uncataloged database will not be present on the new database partition. An attempt to connect to the database on the new database partition returns the error message SQL1013N.

You cannot use a database on the new database partition to contain data until one or more database partition groups are altered to include the new database partition.

You cannot change from a single-partition database to a multi-partition database by adding a database partition to your system. This is because the redistribution of data across database partitions requires a distribution key on each affected table. The distribution keys are automatically generated when a table is created in a

multi-partition database. In a single-partition database, distribution keys can be explicitly created with the CREATE TABLE or ALTER TABLE SQL statements.

Note: If no databases are defined in the system and you are running Enterprise Server Edition on a UNIX operating system, edit the db2nodes.cfg file to add a new database partition definition; do not use any of the procedures described, as they apply only when a database exists.

Windows Considerations: If you are using Enterprise Server Edition on a Windows operating system and have no databases in the instance, use the **db2ncrt** command to scale the database system. If, however, you already have databases, use the **START DBM ADD DBPARTITIONNUM** command to ensure that a database partition is created for each existing database when you scale the system. On Windows operating systems, do not manually edit the database partition configuration file (db2nodes.cfg), because this can introduce inconsistencies to the file.

Adding an online database partition

You can add new database partitions that are online to a partitioned database environment while it is running and while applications are connected to databases.

Procedure

To add an online database partition to a running database manager using the command line:

1. On any existing database partition, run the **START DBM** command.

On all platforms, specify the new database partition values for **DBPARTITIONNUM**, **ADD DBPARTITIONNUM**, **HOSTNAME**, **PORT**, and **NETNAME** parameters. On the Windows platform, you also specify the **COMPUTER**, **USER**, and **PASSWORD** parameters.

You can also specify the source for any temporary table space container definitions that must be created with the databases. If you do not provide table space information, temporary table space container definitions are retrieved from the catalog partition for each database.

For example, to add three new database partitions to an existing database, issue the following commands:

```
START DBM DBPARTITIONNUM 3 ADD DBPARTITIONNUM HOSTNAME HOSTNAME3  
PORT PORT3;  
START DBM DBPARTITIONNUM 4 ADD DBPARTITIONNUM HOSTNAME HOSTNAME4  
PORT PORT4;  
START DBM DBPARTITIONNUM 5 ADD DBPARTITIONNUM HOSTNAME HOSTNAME5  
PORT PORT5;
```

2. Optional: Alter the database partition group to incorporate the new database partition. This action can also be an option when redistributing the data to the new database partition.
3. Optional: Redistribute data to the new database partition. This action is not really optional if you want to take advantage of the new database partitions. You can also include the alter database partition group option as part of the redistribution operation. Otherwise, altering the database partition group to incorporate the new database partitions must be done as a separate action before redistributing the data to the new database partition.
4. Optional: Back up all databases on the new database partition. Although optional, this would be helpful to have for the new database partition and for the other database partitions particularly if you redistributed the data across both the old and the new database partitions.

Restrictions when working online to add a database partition

The status of the new database partition following its addition to the instance depends on the status of the original database partition. Applications may or may not be aware of the new database partition following its addition to the instance if the application uses WITH HOLD cursors.

When adding a new database partition to a single-partition database instance:

- If the original database partition is up when the database partition is added, then the new database partition is down when the add database partition operation completes.
- If the original database partition is down when the database partition is added, then the new database partition is up when the add database partition operation completes.

Applications using WITH HOLD cursors that are started before the add database partition operation runs are not aware of the new database partition when the add database partition operation completes. If the WITH HOLD cursors are closed before the add database partition operation runs, then applications are aware of the new database partition when the add database partition operation completes.

Adding a database partition offline (Linux and UNIX)

You can add new database partitions that are offline to a partitioned database system. The newly added database partition becomes available to all databases when the database manager is started again.

Before you begin

- Install the new server if it does not exist before you can create a database partition on it.
- Make the executables accessible using shared filesystem mounts or local copies.
- Synchronize operating system files with those on existing processors.
- Ensure that the sqllib directory is accessible as a shared file system.
- Ensure that the relevant operating system parameters (such as the maximum number of processes) are set to the appropriate values.
- Register the host name with the name server or in the hosts file in the /etc directory on all database partitions. The host name for the computer must be registered in .rhosts to run remote commands using **rsh** or **rah**.
- Set the default value of the **DB2_FORCE_OFFLINE_ADD_PARTITION** registry variable to TRUE to enforce that the added database partitions is offline.

Procedure

- To add a database partition to a stopped partitioned database server using the command line:
 1. Issue **STOP DBM** to stop all the database partitions.
 2. Run the **ADD DBPARTITIONNUM** command on the new server.

A database partition is created locally for every database that exists in the system. The database parameters for the new database partitions are set to the default value, and each database partition remains empty until you move data to it. Update the database configuration parameter values to match those on the other database partitions.

3. Run the **START DBM** command to start the database system. Note that the database partition configuration file (db2nodes.cfg) has already been updated by the database manager to include the new server during the installation of the new server.

4. Update the configuration file on the new database partition as follows:

- a. On any existing database partition, run the **START DBM** command.

Specify the new database partition values for **DBPARTITIONNUM**, **ADD DBPARTITIONNUM**, **HOSTNAME**, **PORT**, and **NETNAME** parameters as well as the **COMPUTER**, **USER**, and **PASSWORD** parameters.

You can also specify the source for any temporary table space container definitions that must be created with the databases. If you do not provide table space information, temporary table space container definitions are retrieved from the catalog partition for each database.

For example, to add three new database partitions to an existing database, issue the following commands:

```
START DBM DBPARTITIONNUM 3 ADD DBPARTITIONNUM HOSTNAME HOSTNAME3  
PORT PORT3;
```

```
START DBM DBPARTITIONNUM 4 ADD DBPARTITIONNUM HOSTNAME HOSTNAME4  
PORT PORT4;
```

```
START DBM DBPARTITIONNUM 5 ADD DBPARTITIONNUM HOSTNAME HOSTNAME5  
PORT PORT5;
```

When the **START DBM** command is complete, the new server is stopped.

- b. Stop the entire database manager by running the **STOP DBM** command.

When you stop all the database partitions in the system, the node configuration file is updated to include the new database partitions. The node configuration file is not updated with the new server information until **STOP DBM** is executed. This ensures that the **ADD DBPARTITIONNUM** command, which is called when you specify the **ADD DBPARTITIONNUM** parameter to the **START DBM** command, runs on the correct database partition. When the utility ends, the new server partitions are stopped.

5. Start the database manager by running the **START DBM** command.

The newly added database partition is now started with the rest of the system.

When all the database partitions in the system are running, you can run system-wide activities, such as creating or dropping a database.

Note: You might have to issue the **START DBM** command twice for all database partition servers to access the new db2nodes.cfg file.

6. Optional: Alter the database partition group to incorporate the new database partition. This action might also be an option when redistributing the data to the new database partition.

7. Optional: Redistribute data to the new database partition. This action is not really optional if you want to take advantage of the new database partition. You can also include the alter database partition group option as part of the redistribution operation. Otherwise, altering the database partition group to incorporate the new database partition must be done as a separate action before redistributing the data to the new database partition.

8. Optional: Back up all databases on the new database partition. Although optional, this would be helpful to have for the new database partition and for the other database partitions particularly if you redistributed the data across both the old and the new database partitions.

- You can also update the configuration file manually, as follows:

1. Edit the db2nodes.cfg file and add the new database partition to it.
2. Issue the following command to start the new database partition: START DBM DBPARTITIONNUM *partitionnum*
Specify the number you are assigning to the new database partition as the value of *partitionnum*.
3. If the new server is to be a logical partition (that is, it is not database partition 0), use **db2set** command to update the **DBPARTITIONNUM** registry variable. Specify the number of the database partition you are adding.
4. Run the **ADD DBPARTITIONNUM** command on the new database partition.
This command creates a database partition locally for every database that exists in the system. The database parameters for the new database partitions are set to the default value, and each database partition remains empty until you move data to it. Update the database configuration parameter values to match those on the other database partitions.
5. When the **ADD DBPARTITIONNUM** command completes, issue the **START DBM** command to start the other database partitions in the system.
Do not perform any system-wide activities, such as creating or dropping a database, until all database partitions are successfully started.

Adding a database partition offline (Windows)

You can add new database partitions to a partitioned database system while it is stopped. The newly added database partition becomes available to all databases when the database manager is started again.

Before you begin

- You must install the new server before you can create a database partition on it.
- Set the default value of the **DB2_FORCE_OFFLINE_ADD_PARTITION** registry variable to TRUE to enforce that any added database partitions is offline.

Procedure

To add a database partition to a stopped partitioned database server using the command line:

1. Issue **STOP DBM** to stop all the database partitions.
2. Run the **ADD DBPARTITIONNUM** command on the new server.
A database partition is created locally for every database that already exists in the system. The database parameters for the new database partitions are set to the default value, and each database partition remains empty until you move data to it. Update the database configuration parameter values to match those on the other database partitions.
3. Run the **START DBM** command to start the database system. Note that the database partition configuration file has already been updated by the database manager to include the new server during the installation of the new server.
4. Update the configuration file on the new database partition as follows:
 - a. On any existing database partitions, run the **START DBM** command.

Specify the new database partition values for **DBPARTITIONNUM**, **ADD DBPARTITIONNUM**, **HOSTNAME**, **PORT**, and **NETNAME** parameters as well as the **COMPUTER**, **USER**, and **PASSWORD** parameters.

You can also specify the source for any temporary table space container definitions that need to be created with the databases. If you do not provide

table space information, temporary table space container definitions are retrieved from the catalog partition for each database.

For example, to add three new database partitions to an existing database, issue the following commands:

```
START DBM DBPARTITIONNUM 3 ADD DBPARTITIONNUM HOSTNAME HOSTNAME3
PORT PORT3;
START DBM DBPARTITIONNUM 4 ADD DBPARTITIONNUM HOSTNAME HOSTNAME4
PORT PORT4;
START DBM DBPARTITIONNUM 5 ADD DBPARTITIONNUM HOSTNAME HOSTNAME5
PORT PORT5;
```

When the **START DBM** command is complete, the new server is stopped.

- b. Stop the database manager by running the **STOP DBM** command.

When you stop all the database partitions in the system, the node configuration file is updated to include the new database partitions. The node configuration file is not updated with the new server information until **STOP DBM** is executed. This ensures that the **ADD DBPARTITIONNUM** command, which is called when you specify the **ADD DBPARTITIONNUM** parameter to the **START DBM** command, runs on the correct database partitions. When the utility ends, the new server partitions are stopped.

5. Start the database manager by running the **START DBM** command.

The newly added database partitions are now started with the rest of the system.

When all the database partitions in the system are running, you can run system-wide activities, such as creating or dropping a database.

Note: You might have to issue the **START DBM** command twice for all database partition servers to access the new db2nodes.cfg file.

6. Optional: Alter the database partition group to incorporate the new database partition. This action could also be an option when redistributing the data to the new database partition.
7. Optional: Redistribute data to the new database partition. This action is not really optional if you want to take advantage of the new database partition. You can also include the alter database partition group option as part of the redistribution operation. Otherwise, altering the database partition group to incorporate the new database partition must be done as a separate action before redistributing the data to the new database partition.
8. Optional: Back up all databases on the new database partition. Although optional, this would be helpful to have for the new database partition and for the other database partitions particularly if you have redistributed the data across both the old and the new database partitions.

Error recovery when adding database partitions

Adding database partitions does not fail as a result of nonexistent buffer pools, because the database manager creates system buffer pools to provide default automatic support for all buffer pool page sizes.

However, if one of these system buffer pools is used, performance might be seriously affected, because these buffer pools are very small. If a system buffer pool is used, a message is written to the administration notification log. System buffer pools are used in database partition addition scenarios in the following circumstances:

- You add database partitions to a partitioned database environment that has one or more system temporary table spaces with a page size that is different from

the default of 4 KB. When a database partition is created, only the IBMDEFAULTDP buffer pool exists, and this buffer pool has a page size of 4 KB. Consider the following examples:

1. You use the **START DBM** command to add a database partition to the current multi-partition database:

```
START DBM DBPARTITIONNUM 2 ADD DBPARTITIONNUM HOSTNAME newhost PORT 2
```

2. You use the **ADD DBPARTITIONNUM** command after you manually update the db2nodes.cfg file with the new database partition description.

One way to prevent these problems is to specify the WITHOUT TABLESPACES clause on the **ADD DBPARTITIONNUM** or the **START DBM** commands. After doing this, use the CREATE BUFFERPOOL statement to create the buffer pools using the appropriate SIZE and PAGESIZE values, and associate the system temporary table spaces to the buffer pool using the ALTER TABLESPACE statement.

- You add database partitions to an existing database partition group that has one or more table spaces with a page size that is different from the default page size, which is 4 KB. This occurs because the non-default page-size buffer pools created on the new database partition have not been activated for the table spaces.

Note: In previous versions, this command used the NODEGROUP keyword instead of the DATABASE PARTITION GROUP keywords.

Consider the following example:

- You use the ALTER DATABASE PARTITION GROUP statement to add a database partition to a database partition group, as follows:

```
START DBM  
CONNECT TO mpp1  
ALTER DATABASE PARTITION GROUP ng1 ADD DBPARTITIONNUM (2)
```

One way to prevent this problem is to create buffer pools for each page size and then to reconnect to the database before issuing the following ALTER DATABASE PARTITION GROUP statement:

```
START DBM  
CONNECT TO mpp1  
CREATE BUFFERPOOL bp1 SIZE 1000 PAGESIZE 8192  
CONNECT RESET  
CONNECT TO mpp1  
ALTER DATABASE PARTITION GROUP ng1 ADD DBPARTITIONNUM (2)
```

Note: If the database partition group has table spaces with the default page size, message SQL1759W is returned.

Enabling communication between database partitions using FCM communications

In a partitioned database environment, most communication between database partitions is handled by the fast communications manager (FCM).

To enable the FCM at a database partition and allow communication with other database partitions, you must create a service entry in the database partition's services file of the etc directory as shown later in this section. The FCM uses the specified port to communicate. If you have defined multiple database partitions on the same host, you must define a range of ports, as shown later in this section.

Before attempting to manually configure memory for the fast communications manager (FCM), it is recommended that you start with the automatic setting,

which is also the default setting, for the number of FCM Buffers (**fcm_num_buffers**) and for the number of FCM Channels (**fcm_num_channels**). Use the system monitor data for FCM activity to determine if this setting is appropriate.

Windows Considerations

The TCP/IP port range is automatically added to the services file by:

- The install program when it creates the instance or adds a new database partition
- The **db2icrt** utility when it creates a new instance
- The **db2ncrt** utility when it adds the first database partition on the computer

The syntax of a service entry is as follows:

`DB2_instance port/tcp #comment`

DB2_instance

The value for *instance* is the name of the database manager instance. All characters in the name must be lowercase. Assuming an instance name of DB2PUSER, you specify DB2_db2puser.

port/tcp

The TCP/IP port that you want to reserve for the database partition.

#comment

Any comment that you want to associate with the entry. The comment must be preceded by a pound sign (#).

If the services file of the etc directory is shared, you must ensure that the number of ports allocated in the file is either greater than or equal to the largest number of multiple database partitions in the instance. When allocating ports, also ensure that you account for any processor that can be used as a backup.

If the services file of the etc directory is not shared, the same considerations apply, with one additional consideration: you must ensure that the entries defined for the DB2 database instance are the same in all services files of the etc directory (though other entries that do not apply to your partitioned database environment do not have to be the same).

If you have multiple database partitions on the same host in an instance, you must define more than one port for the FCM to use. To do this, include two lines in the services file of the etc directory to indicate the range of ports that you are allocating. The first line specifies the first port, and the second line indicates the end of the block of ports. In the following example, five ports are allocated for the SALES instance. This means no processor in the instance has more than five database partitions. For example:

```
DB2_sales      9000/tcp  
DB2_sales_END  9004/tcp
```

Note: You must specify END in uppercase only. You must also ensure that you include both underscore (_) characters.

Managing database partitions

You can start or stop partitions, drop partitions, or trace partitions.

Before you begin

To work with database partitions, you need authority to attach to an instance. Anyone with SECADM or ACCESSCTRL authority can grant you the authority to access a specific instance.

Procedure

- To start or to stop a specific database partition, use the **START DATABASE MANAGER** command or the **STOP DATABASE MANAGER** command with the **DBPARTITIONNUM** parameter.
- To drop a specific database partition from the db2nodes.cfg configuration file, use the **STOP DATABASE MANAGER** command with the **DROP DBPARTITIONNUM** parameter. Before using the **DROP DBPARTITIONNUM** parameter, run the **DROP DBPARTITIONNUM VERIFY** command to ensure that there is no user data on this database partition.
- To trace the activity on a database partition, use the options specified by IBM Support.

Attention: Use the trace utility only when directed to do so by IBM Support or by a technical support representative.

The trace utility records and formats information about DB2 for Linux, UNIX, and Windows operations. For more details, see the “db2trc - Trace command” topic.

Listing database partition servers in an instance (Windows)

On Windows, use the **db2nlist** command to obtain a list of database partition servers that participate in an instance.

About this task

The command is used as follows:

```
db2nlist
```

When using this command as shown, the default instance is the current instance (set by the **DB2INSTANCE** environment variable). To specify a particular instance, you can specify the instance using:

```
db2nlist /i:instName
```

where *instName* is the particular instance name you want.

You can also optionally request the status of each database partition server by using:

```
db2nlist /s
```

The status of each database partition server might be one of: starting, running, stopping, or stopped.

Eliminating duplicate entries from a list of machines in a partitioned database environment

If you are running multiple logical database partition servers on one computer, your db2nodes.cfg file contains multiple entries for that computer.

About this task

In this situation, the **rah** command needs to know whether you want the command to be executed only once on each computer or once for each logical database partition listed in the db2nodes.cfg file. Use the **rah** command to specify computers. Use the **db2_all** command to specify logical database partitions.

Note: On Linux and UNIX operating systems, if you specify computers, **rah** normally eliminates duplicates from the computer list, with the following exception: if you specify logical database partitions, **db2_all** prepends the following assignment to your command:

```
export DB2NODE=nnn (for Korn shell syntax)
```

where *nnn* is the database partition number taken from the corresponding line in the db2nodes.cfg file, so that the command is routed to the desired database partition server.

When specifying logical database partitions, you can restrict the list to include all logical database partitions except one, or specify only one using the <<-*nnn*< and <<+*nnn*< prefix sequences. You might want to do this if you want to run a command to catalog the database partition first, and when that has completed, run the same command at all other database partition servers, possibly in parallel. This is usually required when running the **RESTART DATABASE** command. You need to know the database partition number of the catalog partition to do this.

If you execute **RESTART DATABASE** using the **rah** command, duplicate entries are eliminated from the list of computers. However if you specify the " prefix, then duplicates are not eliminated, because it is assumed that use of the " prefix implies sending to each database partition server, rather than to each computer.

Specifying the list of machines in a partitioned database environment

By default, the list of computers is taken from the database partition configuration file, db2nodes.cfg.

About this task

Note: On Windows, to avoid introducing inconsistencies into the database partition configuration file, do *not* edit it manually. To obtain the list of computers in the instance, use the **db2nlist** command.

Procedure

To override the list of computers in db2nodes.cfg:

- Specify a path name to the file that contains the list of computers by exporting (on Linux and UNIX operating systems) or setting (on Windows) the environment variable **RAHOSTFILE**.
- Specify the list explicitly, as a string of names separated by spaces, by exporting (on Linux and UNIX operating systems) or setting (on Windows) the environment variable **RAHOSTLIST**.

Note: If both of these environment variables are specified, **RAHOSTLIST** takes precedence.

Changing the database configuration across multiple database partitions

When you have a database that is distributed across more than one database partition, the database configuration file should be the same on all database partitions.

About this task

Consistency is required since the SQL compiler compiles distributed SQL statements based on information in the database partition configuration file and creates an access plan to satisfy the needs of the SQL statement. Maintaining different configuration files on database partitions could lead to different access plans, depending on which database partition the statement is prepared. Use `db2_all` to maintain the configuration files across all database partitions.

Adding containers to SMS table spaces on database partitions

You can add a container to an SMS table space only on a database partition that currently has no containers.

Procedure

To add a container to an SMS table space using the command line, enter the following:

```
ALTER TABLESPACE name
  ADD ('path')
  ON DBPARTITIONNUM (database_partition_number)
```

The database partition specified by number, and every partition in the range of database partitions, must exist in the database partition group on which the table space is defined. A *database_partition_number* might only appear explicitly or within a range in exactly one *db-partitions-clause* for the statement.

Example

The following example shows how to add a new container to database partition number 3 of the database partition group used by table space “plans” on a UNIX operating system:

```
ALTER TABLESPACE plans
  ADD ('/dev/rhdisk0')
  ON DBPARTITIONNUM (3)
```

Using database partition expressions

In most cases, you must use the same storage paths for each partition in a partitioned database environment, and all of the storage paths must exist before you issue a statement. One exception is when you use database partition expressions within the storage path.

Doing this allows the database partition number to be reflected in the storage path such that the resulting path name is different on each partition.

You can specify a database partition expression for container string syntax when creating either SMS or DMS containers. You typically specify the database partition expression when using multiple logical database partitions in a partitioned database system. The expression ensures that container names are unique across

database partition servers. If you specify an expression, the database partition number is part of the container name or, if you specify additional arguments, the result of the argument is part of the container name.

Important: The SMS table space type has been deprecated in Version 10.1 for user-defined permanent table spaces and might be removed in a future release. The SMS table space type is not deprecated for catalog and temporary table spaces. For more information, see "SMS permanent table spaces have been deprecated" in *What's New for DB2 Version 10.1*.

Important: Starting with Version 10.1 Fix Pack 1, the DMS table space type is deprecated for user-defined permanent table spaces and might be removed in a future release. The DMS table space type is not deprecated for catalog and temporary table spaces. For more information, see "DMS permanent table spaces have been deprecated" in *What's New for DB2 Version 10.1*.

Use the argument "\$N" ([blank]\$N) to indicate a database partition expression. You can use a database partition expression anywhere in the storage path name, and you can specify multiple database partition expressions. Terminate the database partition expression with a space character; whatever follows the space is appended to the storage path name after the database partition expression is evaluated. If there is no space character in the storage path name after the database partition expression, it is assumed that the rest of the string is part of the expression. If you specify a number before the N argument, (\$[number]N), the partition number is formatted with leading zeros.

You must specify the argument by using one of the forms in the following table. Operators are evaluated from left to right. A percent sign (%) represents the modulus operator. The database partition number in the following examples is assumed to be 10.

Table 17. Database partition expressions

Syntax	Example	Value
[blank]\$N	"\$N"	10
[blank]\$[number]N	"\$4N"	0010
[blank]\$N+[number]	"\$N+100"	110
[blank]\$N%[number]	"\$N%5"	0
[blank]\$N+[number] %[number]	"\$N+1%5"	1
[blank]\$N %[number]+[number]	"\$N%4+2"	4

If you specified a storage path by using a database partition expression, you must use the same storage path string, including the database partition expression, to drop the path. This path string is in the DB_STORAGE_PATH_WITH_DPE field of the ADMIN_GET_STORAGE_PATHS table function. This element is not shown if you did not include a database partition expression in the original path.

Examples

1. On a system with two database partitions:

```
CREATE TABLESPACE TS1 MANAGED BY DATABASE USING  
(device '/dev/rcont $N' 20000)
```

The following containers are created:

```
/dev/rcont0 - on database partition 0  
/dev/rcont1 - on database partition 1
```

2. On a system with three database partitions:

```
ALTER STOGROUP IBMSTOGROUP ADD '/DB2/path $N'
```

The following paths are added:

```
/DB2/path0 - on database partition 0  
/DB2/path1 - on database partition 1  
/DB2/path2 - on database partition 2
```

3. On a system with four database partitions:

```
CREATE TABLESPACE TS2 MANAGED BY DATABASE USING  
(file '/DB2/containers/TS2/container $N+100' 10000)
```

The following containers are created:

```
/DB2/containers/TS2/container100 - on database partition 0  
/DB2/containers/TS2/container101 - on database partition 1  
/DB2/containers/TS2/container102 - on database partition 2  
/DB2/containers/TS2/container103 - on database partition 3
```

4. On a system with two database partitions:

```
CREATE TABLESPACE TS3 MANAGED BY SYSTEM USING  
('/TS3/cont $N%2','/TS3/cont $N%2+2')
```

The following containers are created:

```
/TS3/cont0 - On database partition 0  
/TS3/cont2 - On database partition 0  
/TS3/cont1 - On database partition 1  
/TS3/cont3 - On database partition 1
```

5. If there are 10 database partitions, the containers use the following syntax:

```
'/dbdir/node $N /cont1'  
'/ $N+1000 /file1'  
' $N%10 /container'  
'/dir/ $N2000 /dmscont'
```

The containers are created as:

```
'/dbdir/node5/cont1'  
'/1005/file1'  
'5/container'  
'/dir/2000/dmscont'
```

Changing database partitions (Windows)

On Windows, use the **db2nchg** command to change database partitions.

About this task

- Move the database partition from one computer to another.
- Change the TCP/IP host name of the computer.
If you are planning to use multiple network adapters, you must use this command to specify the TCP/IP address for the "netname" field in the db2nodes.cfg file.
- Use a different logical port number.
- Use a different name for the database partition server.

The command has the following required parameter:

```
db2nchg /n:node_number
```

The parameter **/n:** is the number of the database partition server that you want to change. This parameter is required.

Optional parameters include:

/i:instance_name

Specifies the instance that this database partition server participates in. If you do not specify this parameter, the default is the current instance.

/u:username,password

Changes the logon account name and password for the DB2 database service. If you do not specify this parameter, the logon account and password remain the same.

/p:logical_port

Changes the logical port for the database partition server. This parameter must be specified if you move the database partition server to a different computer. If you do not specify this parameter, the logical port number remains unchanged.

/h:host_name

Changes the TCP/IP host name used by FCM for internal communications. If you do not specify this parameter, the host name is unchanged.

/m:computer_name

Moves the database partition server to another computer. The database partition server can be moved only if there are no existing databases in the instance.

/g:network_name

Changes the network name for the database partition server.

Use this parameter if you have multiple IP addresses on a computer and you want to use a specific IP address for the database partition server. You can enter the *network_name* using the network name or the IP address.

For example, to change the logical port assigned to database partition 2, which participates in the instance TESTMPP, to use the logical port 3, enter the following command:

```
db2nchg /n:2 /i:TESTMPP /p:3
```

The DB2 database manager provides the capability of accessing DB2 database system registry variables at the instance level on a remote computer. Currently, DB2 database system registry variables are stored in three different levels: computer or global level, instance level, and database partition level. The registry variables stored at the instance level (including the database partition level) can be redirected to another computer by using **DB2REMOTEPREG**. When **DB2REMOTEPREG** is set, the DB2 database manager accesses the DB2 database system registry variables from the computer pointed to by **DB2REMOTEPREG**. The **db2set** command would appear as:

```
db2set DB2REMOTEPREG=remote_workstation
```

where *remote_workstation* is the remote workstation name.

Note:

- Care must be taken in setting this option since all DB2 database instance profiles and instance listings will be located on the specified remote computer name.

- If your environment includes users from domains, ensure that the logon account associated with the DB2 instance service is a domain account. This ensures that the DB2 instance has the appropriate privileges to enumerate groups at the domain level.

This feature might be used in combination with setting **DBINSTPROF** to point to a remote LAN drive on the same computer that contains the registry.

Redistributing data in a database partition group

To create an effective redistribution plan for your database partition group and redistribute your data, issue the **REDISTRIBUTE DATABASE PARTITION GROUP** command or call the sqludrtd API.

Before you begin

To work with database partition groups, you must have SYSADM, SYSCTRL, or DBADM authority.

Procedure

To redistribute data in a database partition group:

- Issue a **REDISTRIBUTE DATABASE PARTITION GROUP** command in the command line processor (CLP).
- Issue the **REDISTRIBUTE DATABASE PARTITION GROUP** command by using the ADMIN_CMD procedure.
- Call the sqludrtd API

Issuing commands in partitioned database environments

In a partitioned database environment, you might want to issue commands to be run on computers in the instance, or on database partition servers. You can do so using the **rah** command or the **db2_a11** command. The **rah** command allows you to issue commands that you want to run at computers in the instance.

If you want the commands to run at database partition servers in the instance, you run the **db2_a11** command. This section provides an overview of these commands. The information that follows applies to partitioned database environments only.

On Windows, to run the **rah** command or the **db2_a11** command, you must be logged on with a user account that is a member of the Administrators group.

On Linux and UNIX operating systems, your login shell can be a Korn shell or any other shell; however, there are differences in the way the different shells handle commands containing special characters.

Also, on Linux and UNIX operating systems, **rah** uses the remote shell program specified by the **DB2RSHCMD** registry variable. You can select between the two remote shell programs: ssh (for additional security), or rsh (or remsh for HP-UX). If **DB2RSHCMD** is not set, rsh (or remsh for HP-UX) is used. The ssh remote shell program is used to prevent the transmission of passwords in clear text in UNIX operating system environments.

If a command runs on one database partition server and you want it to run on all of them, use **db2_a11**. The exception is the **db2trc** command, which runs on all the

logical database partition servers on a computer. If you want to run **db2trc** on all logical database partition servers on all computers, use **rah**.

Note: The **db2_a11** command does not support commands that require interactive user input.

rah and db2_all commands overview

You can run the commands sequentially at one database partition server after another, or you can run the commands in parallel.

On Linux and UNIX operating systems, if you run the commands in parallel, you can either choose to have the output sent to a buffer and collected for display (the default behavior) or the output can be displayed at the computer where the command is issued. On Windows, if you run the commands in parallel, the output is displayed at the computer where the command is issued.

To use the **rah** command, type:

rah command

To use the **db2_a11** command, type:

db2_a11 command

To obtain help about **rah** syntax, type:

rah "?"

The command can be almost anything that you can type at an interactive prompt, including, for example, multiple commands to be run in sequence. On Linux and UNIX operating systems, you separate multiple commands using a semicolon (;). On Windows, you separate multiple commands using an ampersand (&). Do not use the separator character following the last command.

The following example shows how to use the **db2_a11** command to change the database configuration on all database partitions that are specified in the database partition configuration file. Because the ; character is placed inside double quotation marks, the request runs concurrently.

```
db2_a11 ";"DB2 UPDATE DB CFG FOR sample USING LOGFILSIZ 100"
```

Note: The **db2_a11** command does not support commands that require interactive user input.

rah and db2_all commands

This topic includes descriptions of the **rah** and **db2_a11** commands.

Command

Description

rah Runs the command on all computers.

db2_all

Runs a non-interactive command on all database partition servers that you specify. **db2_a11** does not support commands that require interactive user input.

db2_kill

Abruptly stops all processes being run on multiple database partition servers and cleans up all resources on all database partition servers. This

command renders your databases inconsistent. Do *not* issue this command except under direction from IBM Software Support or as directed to recover from a sustained trap.

db2_call_stack

On Linux and UNIX operating systems, causes all processes running on all database partition servers to write call traceback to the syslog.

On Linux and UNIX operating systems, these commands execute **rah** with certain implicit settings such as:

- Run in parallel at all computers
- Buffer command output in /tmp/\$USER/db2_kill, /tmp/\$USER/db2_call_stack respectively.

The command **db2_call_stack** is *not* available on Windows. Use the **db2pd -stack** command instead.

Specifying the rah and db2_all commands

You can specify **rah** command from the command line as the parameter, or in response to the prompt if you do not specify any parameter.

Use the prompt method if the command contains the following special characters:

| & ; < > () { } [] unsubstituted \$

If you specify the command as the parameter on the command line, you must enclose it in double quotation marks if it contains any of the special characters just listed.

Note: On Linux and UNIX operating systems, the command is added to your command history just as if you typed it at the prompt.

All special characters in the command can be entered normally (without being enclosed in quotation marks, except for \). If you require a \ in your command, you must type two backslashes (\\\).

Note: On Linux and UNIX operating systems, if you are not using a Korn shell, all special characters in the command can be entered normally (without being enclosed in quotation marks, except for ", \, unsubstituted \$, and the single quotation mark ('')). If you require one of these characters in your command, you must precede them by three backslashes (\\\). For example, if you require a \ in your command, you must type four backslashes (\\\\).

If you require a double quotation mark ("") in your command, you must precede it by three backslashes, for example, \\\".

Note:

1. On Linux and UNIX operating systems, you cannot include a single quotation mark ('') in your command unless your command shell provides some way of entering a single quotation mark inside a singly quoted string.
2. On Windows, you cannot include a single quotation mark ('') in your command unless your command window provides some way of entering a single quotation mark inside a singly quoted string.

When you run any korn-shell shell-script that contains logic to read from stdin in the background, explicitly redirect stdin to a source where the process can read without getting stopped on the terminal (SIGTTIN message). To redirect stdin, you can run a script with the following form:

```
shell_script </dev/null &
```

if there is no input to be supplied.

In a similar way, always specify </dev/null when running **db2_all** in the background. For example:

```
db2_all ";run_this_command" </dev/null &
```

By doing this you can redirect stdin and avoid getting stopped on the terminal.

An alternative to this method, when you are not concerned about output from the remote command, is to use the “daemonize” option in the **db2_all** prefix:

```
db2_all ";daemonize_this_command" &
```

Running commands in parallel (Linux, UNIX)

By default, the command is run sequentially at each computer, but you can specify to run the commands in parallel using background rshells by prefixing the command with certain prefix sequences. If the rshell is run in the background, then each command puts the output in a buffer file at its remote computer.

Note: The information in this section applies to Linux and UNIX operating systems only.

This process retrieves the output in two pieces:

1. After the remote command completes.
2. After the rshell terminates, which might be later if some processes are still running.

The name of the buffer file is `/tmp/$USER/rahout` by default, but it can be specified by the environment variables **\$RAHBUFDIR** or **\$RAHBUFNAME**.

When you specify that you want the commands to be run concurrently, by default, this script prefixes an additional command to the command sent to all hosts to check that **\$RAHBUFDIR** and **\$RAHBUFNAME** are usable for the buffer file. It creates **\$RAHBUFDIR**. To suppress this, export an environment variable `RAHCHECKBUF=no`. You can do this to save time if you know that the directory exists and is usable.

Before using **rah** to run a command concurrently at multiple computers:

- Ensure that a directory `/tmp/$USER` exists for your user ID at each computer. To create a directory if one does not exist, run:

```
rah ")mkdir /tmp/$USER"
```
- Add the following line to your `.kshrc` (for Korn shell syntax) or `.profile`, and also type it into your current session:

```
export RAHCHECKBUF=no
```
- Ensure that each computer ID at which you run the remote command has an entry in its `.rhosts` file for the ID which runs **rah**; and the ID which runs **rah** has an entry in its `.rhosts` file for each computer ID at which you run the remote command.

Monitoring rah processes (Linux, UNIX)

While any remote commands are still running or buffered output is still being accumulated, processes started by **rah** monitor activity to write messages to the terminal indicating which commands have not been run, and retrieve the buffered output.

About this task

Note: The information in this section applies to Linux and UNIX operating systems only.

The informative messages are written at an interval controlled by the environment variable **RAHWAITTIME**. Refer to the help information for details on how to specify this. All informative messages can be suppressed by exporting **RAHWAITTIME=0**.

The primary monitoring process is a command whose command name (as shown by the **ps** command) is **rahwaitfor**. The first informative message tells you the pid (process id) of this process. All other monitoring processes appear as **ksh** commands running the **rah** script (or the name of the symbolic link). If you want, you can stop all monitoring processes by the command:

```
kill pid
```

where *pid* is the process ID of the primary monitoring process. Do not specify a signal number. Leave the default of 15. This does not affect the remote commands at all, but prevents the automatic display of buffered output. Note that there might be two or more different sets of monitoring processes executing at different times during the life of a single execution of **rah**. However, if at any time you stop the current set, then no more are started.

If your regular login shell is not a Korn shell (for example `/bin/ksh`), you can use **rah**, but there are some slightly different rules on how to enter commands containing the following special characters:

```
" unsubstituted $ '
```

For more information, type **rah "?"**. Also, in a Linux or UNIX operating system, if the login shell at the ID which executes the remote commands is not a Korn shell, then the login shell at the ID which executes **rah** must also not be a Korn shell. (**rah** decides whether the shell of the remote ID is a Korn shell based on the local ID). The shell must not perform any substitution or special processing on a string enclosed in single quotation marks. It must leave it exactly as is.

Extension of the rah command to use tree logic (AIX and Solaris)

To enhance performance, **rah** has been extended to use `tree_logic` on large systems. That is, **rah** will check how many database partitions the list contains, and if that number exceeds a threshold value, it constructs a subset of the list and sends a recursive invocation of itself to those database partitions.

At those database partitions, the recursively invoked **rah** follows the same logic until the list is small enough to follow the standard logic (now the "leaf-of-tree" logic) of sending the command to all database partitions on the list. The threshold can be specified by the **RAHTREETHRESH** environment variable, or defaults to 15.

In the case of a multiple-logical-database partitions-per-physical-database partition system, **db2_a11** will favor sending the recursive invocation to distinct physical database partitions, which will then **rsh** to other logical database partitions on the same physical database partition, thus also reducing inter-physical-database

partition traffic. (This point applies only to **db2_all**, not rah, because rah always sends only to distinct physical database partitions.)

rah and db2_all command prefix sequences

A prefix sequence is one or more special characters.

Type one or more prefix sequences immediately preceding the characters of the command without any intervening blanks. If you want to specify more than one sequence, you can type them in any order, but characters within any multicharacter sequence must be typed in order. If you type any prefix sequences, you must enclose the entire command, including the prefix sequences in double quotation marks, as in the following examples:

- On Linux and UNIX operating systems:

```
rah "};ps -F pid,ppid,etime,args -u $USER"  
db2_all "};ps -F pid,ppid,etime,args -u $USER"
```

- On Windows operating systems:

```
rah "||db2 get db cfg for sample"  
db2_all "||db2 get db cfg for sample"
```

The prefix sequences are:

Sequence

Purpose

| Runs the commands in sequence in the background.

| & Runs the commands in sequence in the background and terminates the command after all remote commands have completed, even if some processes are still running. This might be later if, for example, child processes (on Linux and UNIX operating systems) or background processes (on Windows operating systems) are still running. In this case, the command starts a separate background process to retrieve any remote output generated after command termination and writes it back to the originating computer.

Note: On Linux and UNIX operating systems, specifying & degrades performance, because more **rsh** commands are required.

|| Runs the commands in parallel in the background.

|| & Runs the commands in parallel in the background and terminates the command after all remote commands have completed as described previously for the | & case.

Note: On Linux and UNIX operating systems, specifying & degrades performance, because more **rsh** commands are required.

;
Same as || &. This is an alternative shorter form.

Note: On Linux and UNIX operating systems, specifying ; degrades performance relative to ||, because more **rsh** commands are required.

] Prepends dot-execution of user's profile before executing command.

Note: Available on Linux and UNIX operating systems only.

} Prepends dot-execution of file named in **\$RAHENV** (probably .kshrc) before executing command.

Note: Available on Linux and UNIX operating systems only.

- }{ Prepends dot-execution of user's profile followed by execution of file named in **\$RAHENV** (probably .kshrc) before executing command.

Note: Available on Linux and UNIX operating systems only.
 -) Suppresses execution of user's profile and of file named in **\$RAHENV**.
- Note:** Available on Linux and UNIX operating systems only.
- ' Echoes the command invocation to the computer.
 - < Sends to all the computers except this one.
- <<-nnn<**
- Sends to all-but-database partition server *nnn* (all database partition servers in db2nodes.cfg except for database partition number *nnn*, see the first paragraph following the last prefix sequence in this table).
- nnn* is the corresponding 1-, 2-, or 3-digit database partition number to the *nodenum* value in the db2nodes.cfg file.
- <<-nnn<** is only applicable to **db2_a11**.
- <<+nnn<**
- Sends to only database partition server *nnn* (the database partition server in db2nodes.cfg whose database partition number is *nnn*, see the first paragraph following the last prefix sequence in this table).
- nnn* is the corresponding 1-, 2-, or 3-digit database partition number to the *nodenum* value in the db2nodes.cfg file.
- <<+nnn<** is only applicable to **db2_a11**.
- (blank character)**
- Runs the remote command in the background with stdin, stdout, and stderr all closed. This option is valid only when running the command in the background, that is, only in a prefix sequence which also includes \ or ;. It allows the command to complete much sooner (as soon as the remote command has been initiated). If you specify this prefix sequence on the **rah** command line, then either enclose the command in single quotation marks, or enclose the command in double quotation marks, and precede the prefix character by \ . For example,
- ```
rah ' ; mydaemon'
```
- or
- ```
rah ";\\" mydaemon"
```
- When run as a background process, the **rah** command never waits for any output to be returned.
- > Substitutes occurrences of > with the computer name.
 - " Substitutes occurrences of () by the computer index, and substitutes occurrences of ## by the database partition number.
 - The computer index is a number that associated with a computer in the database system. If you are not running multiple logical partitions, the computer index for a computer corresponds to the database partition number for that computer in the database partition configuration file. To obtain the computer index for a computer in a multiple logical partition database environment, do not count duplicate entries for those

computers that run multiple logical partitions. For example, if MACH1 is running two logical partitions and MACH2 is also running two logical partitions, the database partition number for MACH3 is 5 in the database partition configuration file. The computer index for MACH3, however, would be 3.

- On Windows operating systems, do not edit the database partition configuration file. To obtain the computer index, use the **db2nlist** command.
- When " is specified, duplicates are not eliminated from the list of computers.

Usage notes

- Prefix sequences are considered to be part of the command. If you specify a prefix sequence as part of a command, you must enclose the entire command, including the prefix sequences, in double quotation marks.

Controlling the rah command

This topic lists the environment variables to control the **rah** command.

Table 18. Environment variables that control the rah command

Name	Meaning	Default
\$RAHBUFDIR	Directory for buffer Note: Available on Linux and UNIX operating systems only.	/tmp/\$USER
\$RAHBUFNAME	File name for buffer Note: Available on Linux and UNIX operating systems only.	rahout
\$RAHOSTFILE (on Linux and UNIX operating systems); RAHOSTFILE (on Windows operating systems)	File containing list of hosts	db2nodes.cfg
\$RAHOSTLIST (on Linux and UNIX operating systems); RAHOSTLIST (on Windows operating systems)	List of hosts as a string	extracted from \$RAHOSTFILE
\$RAHCHECKBUF	If set to "no", bypass checks Note: Available on Linux and UNIX operating systems only.	not set

Table 18. Environment variables that control the rah command (continued)

Name	Meaning	Default
\$RAHSLEEPTIME (on Linux and UNIX operating systems); RAHSLEEPTIME (on Windows operating systems)	Time in seconds this script waits for initial output from commands run in parallel.	86400 seconds for db2_kill , 200 seconds for all others
\$RAHWAITTIME (on Linux and UNIX operating systems); RAHWAITTIME (on Windows operating systems)	<p>On Windows operating systems, interval in seconds between successive checks that remote jobs are still running.</p> <p>On Linux and UNIX operating systems, interval in seconds between successive checks that remote jobs are still running and rah: waiting for pid> ... messages.</p>	45 seconds
	<p>On all operating systems, specify any positive integer. Prefix value with a leading zero to suppress messages, for example, export RAHWAITTIME=045.</p> <p>It is not necessary to specify a low value as rah does not rely on these checks to detect job completion.</p>	
\$RAHENV Note: Available on Linux and UNIX operating systems only.	Specifies file name to be executed if \$RAHDFILES=E or K or PE or B	\$ENV
\$RAHUSER (on Linux and UNIX operating systems); RAHUSER (on Windows operating systems)	<p>On Linux and UNIX operating systems, user ID under which the remote command is to be run.</p> <p>On Windows operating systems, the logon account associated with the DB2 Remote Command Service</p>	\$USER

Note: On Linux and UNIX operating systems, the value of **\$RAHENV** where **rah** is run is used, not the value (if any) set by the remote shell.

Specifying which . files run with rah (Linux and UNIX)

This topics lists the . files that are run if no prefix sequence is specified.

Note: The information in this section applies to Linux and UNIX operating systems only.

P .profile

E File named in **\$RAHENV** (probably .kshrc)

K Same as E

PE .profile followed by file named in **\$RAHENV** (probably .kshrc)

B Same as PE

N None (or Neither)

Note: If your login shell is not a Korn shell, any dot files you specify to be executed are executed in a Korn shell process, and so must conform to Korn shell syntax. So, for example, if your login shell is a C shell, to have your .cshrc environment set up for commands executed by **rah**, you should either create a Korn shell *INSTHOME/.profile* equivalent to your .cshrc and specify in your *INSTHOME/.cshrc*:

```
setenv RAHDOTFILES P
```

or you should create a Korn shell *INSTHOME/.kshrc* equivalent to your .cshrc and specify in your *INSTHOME/.cshrc*:

```
setenv RAHDOTFILES E  
setenv RAHENV INSTHOME/.kshrc
```

Also, it is your .cshrc must not write to stdout if there is no tty (as when invoked by **rsh**). You can ensure this by enclosing any lines which write to stdout by, for example,

```
if { tty -s } then echo "executed .cshrc";  
endif
```

Setting the default environment profile for rah on Windows

To set the default environment profile for the **rah** command, use a file called db2rah.env, which should be created in the instance directory.

About this task

Note: The information in this section applies to Windows only.

The file should have the following format:

```
; This is a comment line  
DB2INSTANCE=instancename  
DB2DBDFT=database  
; End of file
```

You can specify all the environment variables that you need to initialize the environment for **rah**.

Determining problems with rah (Linux, UNIX)

This topic gives suggestions on how to handle some problems that you might encounter when you are running **rah**.

Note: The information in this section applies to Linux and UNIX operating systems only.

1. **rah** hangs (or takes a very long time)

This problem might be caused because:

- **rah** has determined that it needs to buffer output, and you did not export RAHCHECKBUF=no. Therefore, before running your command, **rah** sends a command to all computers to check the existence of the buffer directory, and to create it if it does not exist.
- One or more of the computers where you are sending your command is not responding. The **rsh** command will eventually time out but the time-out interval is quite long, usually about 60 seconds.

2. You have received messages such as:

- Login incorrect
- Permission denied

Either one of the computers does not have the ID running **rah** correctly defined in its /etc/hosts file, or the ID running **rah** does not have one of the computers correctly defined in its .rhosts file. If the **DB2RSHCMD** registry variable has been configured to use ssh, then the ssh clients and servers on each computer might not be configured correctly.

Note: You might need to have greater security regarding the transmission of passwords in clear text between database partitions. This will depend on the remote shell program you are using. **rah** uses the remote shell program specified by the **DB2RSHCMD** registry variable. You can select between the two remote shell programs: ssh (for additional security), or rsh (or remsh for HP-UX). If this registry variable is not set, rsh (or remsh for HP-UX) is used.

3. When running commands in parallel using background remote shells, although the commands run and complete within the expected elapsed time at the computers, **rah** takes a long time to detect this and put up the shell prompt. The ID running **rah** does not have one of the computers correctly defined in its .rhosts file, or if the **DB2RSHCMD** registry variable has been configured to use ssh, then the ssh clients and servers on each computer might not be configured correctly.
4. Although **rah** runs fine when run from the shell command line, if you run **rah** remotely using rsh, for example,

```
rsh somewhere -l $USER db2_kill
```

rah never completes.

This is normal. **rah** starts background monitoring processes, which continue to run after it has exited. Those processes normally persist until all processes associated with the command you ran have themselves terminated. In the case of **db2_kill**, this means termination of all database managers. You can terminate the monitoring processes by finding the process whose command is **rahwaitfor** and kill *process_id*. Do not specify a signal number. Instead, use the default (15).

5. The output from **rah** is not displayed correctly, or **rah** incorrectly reports that **\$RAHBUFSIZE** does not exist, when multiple commands of **rah** were issued under the same **\$RAHUSER**.

This is because multiple concurrent executions of **rah** are trying to use the same buffer file (for example, **\$RAHBUFDIR** or **\$RAHBUFSIZE**) for buffering the outputs. To prevent this problem, use a different **\$RAHBUFSIZE** for each concurrent **rah** command, for example in the following ksh:

```
export RAHBUFSIZE=rahout
rah ";$command_1" &
export RAHBUFSIZE=rah2out
rah ";$command_2" &
```

or use a method that makes the shell choose a unique name automatically such as:

```
RAHBUFSIZE=rahout.$$ db2_all "....."
```

Whatever method you use, you must ensure that you clean up the buffer files at some point if disk space is limited. **rah** does not erase a buffer file at the end of execution, although it will erase and then re-use an existing file the next time you specify the same buffer file.

6. You entered

```
rah '"print from ()'
```

and received the message:

```
ksh: syntax error at line 1 : (' unexpected
```

Prerequisites for the substitution of () and ## are:

- Use **db2_all**, not **rah**.
- Ensure a **RAHOSTFILE** is used either by exporting **RAHOSTFILE** or by defaulting to your /sqllib/db2nodes.cfg file. Without these prerequisites, **rah** leaves the () and ## as is. You receive an error because the command **print from ()** is not valid.

For a performance tip when running commands in parallel, use | rather than |&, and use || rather than ||& or ; unless you truly need the function provided by &. Specifying & requires more remote shell commands and therefore degrades performance.

Dropping database partitions

You can drop a database partition that is not being used by any database and free the computer for other uses.

Before you begin

Verify that the database partition is not in use by issuing the **DROP DBPARTITIONNUM VERIFY** command or the sqledrpn API.

- If you receive message SQL6034W (Database partition not used in any database), you can drop the database partition.
- If you receive message SQL6035W (Database partition in use by database), use the **REDISTRIBUTE DATABASE PARTITION GROUP** command to redistribute the data from the database partition that you are dropping to other database partitions from the database alias.

Also ensure that all transactions for which this database partition was the coordinator have all committed or rolled back successfully. This might require doing crash recovery on other servers. For example, if you drop the coordinator partition, and another database partition participating in a transaction crashed before the coordinator partition was dropped, the crashed database partition will not be able to query the coordinator partition for the outcome of any indoubt transactions.

Procedure

To drop a database partition using the command line:

Issue the **STOP DBM** command with the **DROP DBPARTITIONNUM** parameter to drop the database partition.

After the command completes successfully, the system is stopped. Then start the database manager with the **START DBM** command.

Dropping a database partition from an instance (Windows)

On Windows, use the **db2ndrop** command to drop a database partition server from an instance that has no databases. If you drop a database partition server, its database partition number can be reused for a new database partition server.

About this task

Exercise caution when you drop database partition servers from an instance. If you drop the instance-owning database partition server zero (0) from the instance, the instance becomes unusable. If you want to drop the instance, use the **db2idrop** command.

Note: Do not use the **db2ndrop** command if the instance contains databases. Instead, use the **STOP DBM DROP DBPARTITIONNUM** command. This ensures that the database is correctly removed from the database partition. **DO NOT EDIT** the **db2nodes.cfg** file, since changing the file might cause inconsistencies in the partitioned database environment.

If you want to drop a database partition that is assigned the logical port 0 from a computer that is running multiple logical database partitions, you must drop all the other database partitions assigned to the other logical ports before you can drop the database partition assigned to logical port 0. Each database partition server must have a database partition assigned to logical port 0.

The command has the following parameters:

db2ndrop /n:dbpartitionnum /i:instance_name

/n:dbpartitionnum

The unique database partition number (*dbpartitionnum*) to identify the database partition server. This is a required parameter. The number can be from zero (0) to 999 in ascending sequence. Recall that database partition zero (0) represents the instance-owning computer.

/i:instance_name

The instance name (*instance_name*). This is an optional parameter. If not given, the default is the current instance (set by the **DB2INSTANCE** registry variable).

Tables in partitioned database environments

There are performance advantages to creating a table across several database partitions in a partitioned database environment. The work associated with the retrieval of data can be divided among the database partitions.

Before you begin

Before creating a table that will be physically divided or distributed, you need to consider the following:

- Table spaces can span more than one database partition. The number of database partitions they span depends on the number of database partitions in a database partition group.
- Tables can be collocated by being placed in the same table space or by being placed in another table space that, together with the first table space, is associated with the same database partition group.

About this task

Creating a table that will be a part of several database partitions is specified when you are creating the table. There is an additional option when creating a table in a partitioned database environment: the *distribution key*. A distribution key is a key that is part of the definition of a table. It determines the database partition on which each row of data is stored.

If you do not specify the distribution key explicitly, the following defaults are used. *Ensure that the default distribution key is appropriate.*

- If a primary key is specified in the CREATE TABLE statement, the first column of the primary key is used as the distribution key.
- For a multiple partition database partition group, if there is no primary key, the first column that is not a long field is used.
- If no columns satisfy the requirements for a default distribution key, the table is created without one (this is allowed only in single-partition database partition groups).

You must be careful to select an appropriate distribution key because *it cannot be changed later*. Furthermore, any unique indexes (and therefore unique or primary keys) must be defined as a superset of the distribution key. That is, if a distribution key is defined, unique keys and primary keys must include all of the same columns as the distribution key (they might have more columns).

The size of a database partition of a table is the smaller amount of a specific limit associated with the type of table space and page size used, and the amount of disk space available. For example, assuming a large DMS table space with a 4 KB page size, the size of a table is the smaller amount of 8 TB multiplied by the number of database partitions and the amount of available disk space. See the related links for the complete list of database manager page size limits.

To create a table in a partitioned database environment using the command line, enter:

```
CREATE TABLE name>
  (<column_name> <data_type> <null_attribute>)
  IN <table_space_name>
  INDEX IN <index_space_name>
  LONG IN <long_space_name>
  DISTRIBUTE BY HASH (<column_name>)
```

Following is an example:

```
CREATE TABLE MIXREC (MIX_CNTL INTEGER NOT NULL,
                     MIX_DESC CHAR(20) NOT NULL,
                     MIX_CHR CHAR(9) NOT NULL,
                     MIX_INT INTEGER NOT NULL,
                     MIX_INTS SMALLINT NOT NULL,
                     MIX_DEC DECIMAL NOT NULL,
                     MIXFLT FLOAT NOT NULL,
                     MIX_DATE DATE NOT NULL,
                     MIX_TIME TIME NOT NULL,
                     MIX_TMSTMP TIMESTAMP NOT NULL)
                     IN MIXTS12
                     DISTRIBUTE BY HASH (MIX_INT)
```

In the preceding example, the table space is MIXTS12 and the distribution key is MIX_INT. If the distribution key is not specified explicitly, it is MIX_CNTL. (If no primary key is specified and no distribution key is defined, the distribution key is the first non-long column in the list.)

A row of a table, and all information about that row, always resides on the same database partition.

Redistributing data across database partitions

Redistributing data is a task that you might perform in a partitioned database environment after adding or removing database partitions or when an undesirable proportion of data is appearing on a particular partition so as to rebalance or reconfigure the distribution.

Data redistribution

Data redistribution is a database administration operation that can be performed to primarily move data within a partitioned database environment when partitions are added or removed. The goal of this operation is typically to balance the usage of storage space, improve database system performance, or satisfy other system requirements.

Data redistribution can be performed by using one of the following interfaces:

- **REDISTRIBUTE DATABASE PARTITION GROUP** command
- ADMIN_CMD built-in procedure
- STEPWISE_REDISTRIBUTE_DBPG built-in procedure
- sqludrdr API

Data redistribution within a partitioned database is done for one of the following reasons:

- To rebalance data whenever a new database partition is added to the database environment or an existing database partition is removed.
- To introduce user-specific data distribution across partitions.
- To secure sensitive data by isolating it within a particular partition.

Data redistribution is performed by connecting to a database at the catalog database partition and beginning a data redistribution operation for a specific partition group by using one of the supported interfaces. Data redistribution relies on the existence of distribution key definitions for the tables within the partition group. The distribution key value for a row of data within the table is used to determine on which partition the row of data will be stored. A distribution key is generated automatically when a table is created in a multi-partition database partition group. A distribution key can also be explicitly defined by using the CREATE TABLE or ALTER TABLE statements. By default during data redistribution, for each table within a specified database partition group, table data is divided and redistributed evenly among the database partitions. Other distributions, such as a skewed distribution, can be achieved by specifying an input distribution map which defines how the data is to be distributed. Distribution maps can be generated during a data redistribution operation for future use or can be created manually.

Comparison of logged, recoverable redistribution and minimally logged, not roll-forward recoverable redistribution

When performing data redistribution by using either the **REDISTRIBUTE DATABASE PARTITION GROUP** command or the ADMIN_CMD built-in procedure, you can choose between two methods of data redistribution: logged, recoverable redistribution and minimally logged, not roll-forward recoverable redistribution. The latter method is specified by using the **NOT ROLLFORWARD RECOVERABLE** command parameter.

Data redistribution in capacity growth scenarios, during load balancing, or during performance tuning can require precious maintenance window time, a considerable

amount of planning time, as well as log space and extra container space that can be expensive. Your choice of redistribution methods depends on whether you prioritize recoverability or speed:

- When the logged, recoverable redistribution method is used, extensive logging of all row movement is performed such that the database can be recovered in the event of any interruptions, errors, or other business need.
- The not roll-forward recoverable redistribution method offers better performance because data is moved in bulk and log records are no longer required for insert and delete operations.

The latter method is particularly beneficial if, in the past, large active log space and storage requirements forced you to break a single data redistribution operation into multiple smaller redistribution tasks, which might have resulted in even more time required to complete the end-to-end data redistribution operation.

The not roll-forward recoverable redistribution method is the best practice in most situations because the data redistribution takes less time, is less error prone, and consumes fewer system resources. As a result, the total cost of performing data redistribution is reduced, which frees up time and resources for other business operations.

Minimally logged, not roll-forward recoverable redistribution

When the **REDISTRIBUTE DATABASE PARTITION GROUP** command is issued and the **NOT ROLLFORWARD RECOVERABLE** parameter is specified, a minimal logging strategy is used that minimizes the writing of log records for each moved row. This type of logging is important for the usability of the redistribute operation since an approach that fully logs all data movement could, for large systems, require an impractical amount of active and permanent log space and would generally have poorer performance characteristics.

There are also features and optional parameters that are only available when you choose the not roll-forward recoverable redistribution method. For example, by default this method of redistribution quiesces the database and performs a precheck to ensure that prerequisites are met. You can also optionally specify to rebuild indexes and collect table statistics as part of the redistribution operation. The combination and automation of these otherwise manual tasks makes them less error prone, faster, and more efficient, while providing you with more control over the operations.

The not roll-forward recoverable redistribution method automatically reorganizes the tables, which can free up disk space. This table reorganization comes at no additional performance cost to the redistribute operation. For tables with clustering indexes, the reorganization does not attempt to maintain clustering. If perfect clustering is desired, it will be necessary to perform a **REORG TABLE** command on tables with a clustering index after data redistribution completes. For multi-dimensional-clustered (MDC) tables, the reorganization maintains the clustering of the table and frees unused blocks for reuse; however the total size of the table after redistribution appears unchanged.

Note: It is critical that you back up each affected table space or the entire database when the redistribute operation is complete because rolling forward through this type of redistribute operation results in all tables that were redistributed being marked invalid. Such tables can only be dropped, which means there is no way to recover the data in these tables. This is why, for recoverable databases, the

REDISTRIBUTE DATABASE PARTITION GROUP utility when issued with the **NOT ROLLFORWARD RECOVERABLE** option puts all table spaces it touches into the BACKUP PENDING state. This state forces you to back up all redistributed table spaces at the end of a successful redistribute operation. With a backup taken after the redistribution operation, you should not have a need to roll-forward through the redistribute operation itself.

There is one important consequence of the lack of roll-forward recoverability: If you choose to allow updates to be made against tables in the database (even tables outside the database partition group being redistributed) while the redistribute operation is running, including the period at the end of redistribute where the table spaces touched by redistribute are being backed up, such updates can be lost in the event of a serious failure, for example, a database container is destroyed. The reason that such updates can be lost is that the redistribute operation is not roll-forward recoverable. If it is necessary to restore the database from a backup taken before the redistribution operation, then it will not be possible to roll-forward through the logs in order to replay the updates that were made during the redistribution operation without also rolling forward through the redistribution which, as was described above, leaves the redistributed tables in the UNAVAILABLE state. Thus, the only thing that can be done in this situation is to restore the database from the backup taken before the redistribution without rolling forward. Then the redistribute operation can be performed again. Unfortunately, all the updates that occurred during the original redistribute operation are lost.

The importance of this point cannot be overemphasized. In order to be certain that there will be no lost updates during a redistribution operation, one of the following must be true:

- You must avoid making updates during the operation of the **REDISTRIBUTE DATABASE PARTITION GROUP** command, including the period after the command finishes where the affected table spaces are being backed up.
- The redistribution operation is performed with the **QUIESCE DATABASE** command parameter set to YES. You must still ensure that any applications or users that are allowed to access the quiesced database are not making updates.
- Updates that are applied during the redistribute operation come from a repeatable source, meaning that they can be applied again at any time. For example, if the source of updates is data that is stored in a file and the updates are applied during batch processing, then clearly even in the event of a failure requiring a database restore, the updates would not be lost since they could simply be applied again at any time.

With respect to allowing updates to the database during the redistribution operation, you must decide whether such updates are appropriate or not based on whether the updates can be repeated after a database restore, if necessary.

Note: Not every failure during operation of the **REDISTRIBUTE DATABASE PARTITION GROUP** command results in this problem. In fact, most do not. The **REDISTRIBUTE DATABASE PARTITION GROUP** command is fully restartable, meaning that if the utility fails in the middle of its work, it can be easily continued or aborted with the **CONTINUE** or **ABORT** options. The failures mentioned above are failures that require the user to restore from the backup taken before the redistribute operation.

Logged, recoverable redistribution

The original and default version of the **REDISTRIBUTE DATABASE PARTITION GROUP** command, this method redistributes data by using standard SQL inserts and deletes. Extensive logging of all row movement is performed such that the database is recoverable by restoring it using the **RESTORE DATABASE** command then rolling forward through all changes using the **ROLLFORWARD DATABASE** command.

After the data redistribution, the source table contains empty spaces because rows were deleted and sent to new database partitions. If you want to free the empty spaces, you must reorganize the tables. To reorganize the tables, you must use a separate operation, after the redistribution is complete. To improve performance of this method, drop the indexes and re-create them after the redistribution is complete.

Determining if data redistribution is needed

Determining the current data distribution for a database partition group or table can be helpful in determining if data redistribution is required. Details about the current data distribution can also be used to create a custom distribution map that specifies how to distribute data.

About this task

If a new database partition is added to a database partition group, or an existing database partition is dropped from a database partition group, perform data redistribution to balance data among all the database partitions.

If no database partitions have been added or dropped from a database partition group, then data redistribution is usually only indicated when there is an unequal distribution of data among the database partitions of the database partition group. Note that in some cases an unequal distribution of data can be desirable. For example, if some database partitions reside on a powerful machine, then it might be beneficial for those database partitions to contain larger volumes of data than other partitions.

Procedure

To determine if data redistribution is needed:

1. Get information about the current distribution of data among database partitions in the database partition group.

Run the following query on the largest table (alternatively, a representative table) in the database partition group:

```
SELECT DBPARTITIONNUM(column_name), COUNT(*) FROM table_name
    GROUP BY DBPARTITIONNUM(column_name)
    ORDER BY DBPARTITIONNUM(column_name) DESC
```

Here, *column_name* is the name of the distribution key for table *table_name*.

The output of this query shows how many records from *table_name* reside on each database partition. If the distribution of data among database partitions is not as desired, then proceed to the next step.

2. Get information about the distribution of data across hash partitions.

Run the following query with the same *column_name* and *table_name* that were used in the previous step:

```
SELECT HASHEDVALUE(column_name), COUNT(*) FROM table_name  
    GROUP BY HASHEDVALUE(column_name)  
    ORDER BY HASHEDVALUE(column_name) DESC
```

The output of this query can easily be used to construct the distribution file needed when the **USING DISTFILE** parameter in the **REDISTRIBUTE DATABASE PARTITION GROUP** command is specified. Refer to the **REDISTRIBUTE DATABASE PARTITION GROUP** command reference for a description of the format of the distribution file.

3. Optional: If the data requires redistribution, you can plan to do this operation during a system maintenance opportunity.

When the **USING DISTFILE** parameter is specified, the **REDISTRIBUTE DATABASE PARTITION GROUP** command uses the information in the file to generate a new partition map for the database partition group. This operation results in a uniform distribution of data among database partitions.

If a uniform distribution is not desired, you can construct your own target partition map for the redistribution operation. The target partition map can be specified by using the **USING TARGETMAP** parameter in the **REDISTRIBUTE DATABASE PARTITION GROUP** command.

Results

After doing this investigation, you will know if your data is uniformly distributed or not or if data redistribution is required.

Prerequisites for data redistribution

Before data redistribution can be performed successfully for a set of tables within a database partition group, certain prerequisites must be met.

The following is a list of mandatory prerequisites:

- Authorization to perform data redistribution from the supported data redistribution interface of choice.
- A significant amount of time during a period of low system activity in which to perform the redistribution operation.
- All tables containing data to be redistributed as part of a data redistribution operation must be in a NORMAL state. For example, tables cannot be in LOAD PENDING state or other inaccessible load table states. To check the states of tables, establish a connection to each partition in the database partition group and issue the **LOAD QUERY** command. The output of this command contains information about the state of the table. The documentation of the **LOAD QUERY** command explains the meaning of each of the table states and how to move tables from one state to another.
- All tables within the database partition being redistributed must have been defined with a distribution key. If a new database partition is added to a single-partition system, data redistribution cannot be performed until all of the tables within the partitions have a distribution key. For tables that were created using the CREATE TABLE statement and have definitions that do not contain a distribution key, you must alter the table by using the ALTER TABLE statement to add a distribution key before redistributing the data.
- Replicated materialized query tables contained in a database partition group must be dropped before you redistribute the data. Store a copy of the materialized query table definitions so that they can be recreated after data redistribution completes.

- If a non-uniform redistribution is desired a distribution map must be created as a target distribution map to be used a parameter to the redistribute interface.
- A backup of the database must be created by using the **BACKUP DATABASE** command. This backup is not a mandatory prerequisite however it is strongly recommended that it be done.
- A connection must be established to the database from the catalog database partition.
- Adequate space must be available to rebuild all indexes either during or after the data redistribution. The **INDEXING MODE** command parameter affects when the indexes are rebuilt.
- When the **NOT ROLLFORWARD RECOVERABLE** command parameter is specified, adequate space should be available for writing status information to control files used by IBM Service for problem determination. The control files are generated in the following paths and should be manually deleted when the data redistribution operation is complete:
 - On Linux and UNIX operating systems: `diagpath/redis/db_name/db_partitiongroup_name/timestamp/`
 - On Windows operating systems: `diagpath\redis\db_name\db_partitiongroup_name\timestamp\`

You can calculate the space requirements in bytes for the control files by using the following formula:

$$(number \text{ of } pages \text{ for all tables in the database partition group}) * 64 \text{ bytes} \\ + number \text{ of LOB values in the database partition group}) * 600 \text{ bytes}$$

To estimate *number of LOB values in the database partition group*, add the number of LOB columns in your tables and multiply it by the number of rows in the largest table.

- When the **NOT ROLLFORWARD RECOVERABLE** command parameter is **not** specified, adequate log file space must be available to contain the log entries associated with the INSERT and DELETE operations performed during data redistribution otherwise data redistribution will be interrupted or fail.

The **util_heap_sz** database configuration parameter is critical to the processing of data movement between database partitions - allocate as much memory as possible to **util_heap_sz** for the duration of the redistribution operation. Sufficient **sortheap** is also required if indexes are being rebuilt as part of the redistribution operation. Increase the value of **util_heap_sz** and **sortheap** database configuration parameter, as necessary, to improve redistribution performance.

Log space requirements for data redistribution

To successfully perform a data redistribution operation, adequate log file space must be allocated to ensure that data redistribution is not interrupted. Log space requirements are less of a concern when you specify the **NOT ROLLFORWARD RECOVERABLE** command parameter, since there is minimal logging during that type of data redistribution.

The quantity of log file space required depends on multiple factors including which options of the **REDISTRIBUTE DATABASE PARTITION GROUP** command are used.

When the redistribution is performed from any supported interface where the data redistribution is roll-forward recoverable:

- The log must be large enough to accommodate the INSERT and DELETE operations at each database partition where data is being redistributed. The

heaviest logging requirements will be either on the database partition that will lose the most data, or on the database partition that will gain the most data.

- If you are moving to a larger number of database partitions, use the ratio of current database partitions to the new number of database partitions to estimate the number of INSERT and DELETE operations. For example, consider redistributing data that is uniformly distributed before redistribution. If you are moving from four to five database partitions, approximately twenty percent of the four original database partitions will move to the new database partition. This means that twenty percent of the DELETE operations will occur on each of the four original database partitions, and all of the INSERT operations will occur on the new database partition.
- Consider a nonuniform distribution of the data, such as the case in which the distribution key contains many NULL values. In this case, all rows that contain a NULL value in the distribution key move from one database partition under the old distribution scheme and to a different database partition under the new distribution scheme. As a result, the amount of log space required on those two database partitions increases, perhaps well beyond the amount calculated by assuming uniform distribution.
- The redistribution of each table is a single transaction. For this reason, when you estimate log space, you multiply the percentage of change, such as twenty percent, by the size of the largest table. Consider, however, that the largest table might be uniformly distributed but the second largest table, for example, might have one or more inflated database partitions. In such a case, consider using the non-uniformly distributed table instead of the largest one.

Note: After you estimate the maximum amount of data to be inserted and deleted at a database partition, double that estimate to determine the peak size of the active log. If this estimate is greater than the active log limit of 1024 GB, then the data redistribution must be done in steps. For example, use the STEPWISE_REDISTRIBUTE_DBPG procedure with a number of steps proportional to how much the estimate is greater than active log limit. You might also set the **logsecond** database configuration parameter to -1 to avoid most log space problems.

When the redistribution is performed from any supported interface where the data redistribution is not roll-forward recoverable:

- Log records are not created when rows are moved as part of data redistribution. This behavior significantly reduces log file space requirements; however, when this option is used with database roll-forward recovery, the redistribute operation log record cannot be rolled forward, and any tables processed as part of the roll-forward operation remain in UNAVAILABLE state.
- If the database partition group undergoing data redistribution contains tables with long-field (LF) or large-object (LOB) data in the tables, the number of log records generated during data redistribution will be higher, because a log record is created for each row of data. In this case, expect the log space requirement per database partition to be roughly one third of the amount of data moving on that partition (that is, data being sent, received, or both).

Restrictions on data redistribution

Restrictions on data redistribution are important to note before proceeding with data redistribution or when troubleshooting problems related to data redistribution.

The following restrictions apply to data redistribution:

- Data redistribution on partitions where tables do not have partitioning key definitions is restricted.
- When data redistribution is in progress:
 - Starting another redistribution operation on the same database partition group is restricted.
 - Dropping the database partition group is restricted.
 - Altering the database partition group is restricted.
 - Executing an ALTER TABLE statement on any table in the database partition group is restricted.
 - Creating new indexes in the table undergoing data redistribution is restricted.
 - Dropping indexes defined on the table undergoing data redistribution is restricted.
 - Querying data in the table undergoing data redistribution is restricted.
 - Updating the table undergoing data redistribution is restricted.
- Updating tables in a database undergoing a data redistribution that was started using the **REDISTRIBUTE DATABASE PARTITION GROUP** command where the **NOT ROLLFORWARD RECOVERABLE** command parameter was specified is restricted. Although the updates can be made, if data redistribution is interrupted the changes made to the data might be lost and so this practice is strongly discouraged.
- When the **REDISTRIBUTE DATABASE PARTITION GROUP** command is issued and the **NOT ROLLFORWARD RECOVERABLE** command parameter is specified:
 - Data distribution changes that occur during the redistribution are not roll-forward recoverable.
 - If the database is recoverable, the table space is put into the BACKUP PENDING state after accessing the first table within the table space. To remove the table from this state, you must take a backup of the table space changes when the redistribution operation completes.
 - During data redistribution, the data in the tables in the database partition group being redistributed cannot be updated - the data is read-only. Tables that are actively being redistributed are inaccessible.
- For typed (hierarchy) tables, if the **REDISTRIBUTE DATABASE PARTITION GROUP** command is used and the **TABLE** parameter is specified with the value **ONLY**, then the table name is restricted to being the name of the root table only. Sub-table names cannot be specified.
- Data redistribution is supported for the movement of data between database partitions. For partitioned tables, however, movement of data between ranges of a data partitioned table is restricted unless both of the following are true:
 - The partitioned table has an access mode of FULL ACCESS in the SYSTABLES.ACCESS_MODE catalog table.
 - The partitioned table does not have any partitions currently being attached or detached.
- For replicated materialized query tables, if the data in a database partition group contains replicated materialized query tables, you must drop these tables before you redistribute the data. After data is redistributed, you can recreate the materialized query tables.
- For database partitions that contain multi-dimensional-clustered tables (MDCs) use of the **REDISTRIBUTE DATABASE PARTITION GROUP** command is restricted and will not proceed successfully if there are any multi-dimensional-clustered tables

in the database partition group that contain rolled out blocks that are pending cleanup. These MDC tables must be cleaned up before data redistribution can be resumed or restarted.

- Dropping tables that are currently marked in the DB2 catalog views as being in the state "Redistribute in Progress" is restricted. To drop a table in this state, first run the **REDISTRIBUTE DATABASE PARTITION GROUP** command with the **ABORT** or **CONTINUE** parameters and an appropriate table list so that redistribution of the table is either completed or aborted.

Best practices for data redistribution

Data redistribution can be optimally performed when best practices for data redistribution are followed.

Consider the following best practices when planning your data redistribution:

- Ensure that all documented data redistribution prerequisites have been met.
By default, redistribution operations that are not roll-forward recoverable perform a *precheck* and proceed only if the verification completes successfully. To verify the prerequisites without launching the redistribution operation, specify the **PRECHECK ONLY** command parameter.
- Gather information and metrics about your database environment.
If performance changes after the redistribution, you can use the information and metrics to identify the reason for the change.
- Back up the database before you perform the data redistribution.
This is especially important if the redistribution operation is not roll-forward recoverable; if a catastrophic failure occurs during the redistribution and the database is lost or a table is corrupted, you can restore the database from this backup.
- Perform the redistribution during a planned outage, if possible. The instance does not need to be stopped; quiescing the database is sufficient.
By default, redistribution operations that are not roll-forward recoverable force all users off the database and put the database into a quiesced mode. You must still ensure that any applications or users that are allowed to access the quiesced database are not making updates, for the following reasons:
 - If a disaster occurs, recovering data changes that occurred during the redistribution is complex and, in some cases, not possible.
 - Redistribution typically uses a lot of resources on the servers. Parallel querying of data might cause both the redistribution and the queries to slow down significantly. Redistributing the data online can also cause lock timeouts or deadlocks.
- Perform a redistribution operation that is not roll-forward recoverable.
Data is moved in bulk instead of by internal insert and delete operations. This reduces the number of times that a table must be scanned and accessed, which results in better performance.
Log records are not required for each of the insert and delete operations. This means that you do not need to manage large amounts of active log space and log archiving space in your system when performing data redistribution.
- A uniform distribution of data might not always result in the best database performance. If a uniform distribution is not desired, then you can construct your own target partition map for the redistribution operation.
In general if you redistribute data in a frequently accessed table such that infrequently accessed data is on few database partitions in the database partition

group, and the frequently accessed data is distributed over a larger number of database partitions, you can improve data access performance and throughput for the most frequently run applications that access this data.

Data redistribution mechanism

Data redistribution can be performed by using different methods in different interfaces however internally the mechanism by which the data is moved is the same. It can be helpful to understand this mechanism so that you are aware of automatic changes being made within the DB2 database environment.

Data redistribution involves the use of the available source distribution map and target distribution map to identify hash database partitions that have been assigned to a new location. The new location is identified by a new database partition number. All rows that correspond to a database partition that have a new location are moved from the database partition specified in the source distribution map to the database partition specified in the target distribution map.

Data redistribution internally invokes a utility that performs the following ordered actions:

1. Obtains a new distribution map ID for the target distribution map, and inserts it into the SYSCAT.PARTITIONMAPS catalog view.
2. Updates the REDISTRIBUTE_PMAP_ID column in the SYSCAT.DBPARTITIONGROUPS catalog view for the database partition group with the new distribution map ID.
3. Adds any new database partitions to the SYSCAT.DBPARTITIONGROUPDEF catalog view.
4. Sets the IN_USE column in the SYSCAT.DBPARTITIONGROUPDEF catalog view to 'D' for any database partition that is to be dropped, if the **DROP DBPARTITIONNUM** command parameter was specified.
5. Commits all catalog updates.
6. Creates database files for all new database partitions if the **ADD DBPARTITIONNUM** command parameter is specified; also might create table spaces in the new database partitions.
7. Redistributions the data on a table-by-table basis for every table in the database partition group, in the following steps:
 - a. Puts the table spaces into the BACKUP PENDING state, if the utility did not put them into that state already.
 - b. Locks the row for the table in the SYSTABLES catalog table.
 - c. Invalidates all packages that involve this table. The distribution map ID associated with the table changes because the table rows are redistributed. Because the packages are invalidated, the compiler must obtain the new database partitioning information for the table and generate packages accordingly.
 - d. Locks the table in super exclusive mode (with a z-lock).
 - e. Redistributions data by using bulk data movement operations.
 - f. If the redistribution operation succeeds, the distribution map ID for the table is updated in SYSCAT.TABLES. The utility issues a COMMIT for the table and continues with the next table in the database partition group. If the operation fails before the table is fully redistributed, the utility fails. Any partially redistributed tables are left in the REDIST_IN_PGRS state and the table is inaccessible until the redistribute operation is either continued or aborted.

- Deletes database files and deletes entries in the SYSCAT.DBPARTITIONGROUPDEF catalog view for database partitions that were previously marked to be dropped.
8. Updates the database partition group record in the SYSCAT.DBPARTITIONGROUPS catalog view to set PMAP_ID to the value of REDISTRIBUTE_PMAP_ID and REDISTRIBUTE_PMAP_ID to NULL.
 9. Deletes the old distribution map from the SYSCAT.PARTITIONMAPS catalog view.
 10. Does a COMMIT for all changes.

When these steps are done data redistribution is complete. For more information about the success or failure status of the data redistribution and each of the individual data redistributions, review the redistribution log file.

Redistributing data across database partitions by using the REDISTRIBUTE DATABASE PARTITION GROUP command

The **REDISTRIBUTE DATABASE PARTITION GROUP** command is the recommended interface for performing data redistribution.

Procedure

To redistribute data across database partitions in a database partition group:

1. Optional: Perform a backup of the database. See the **BACKUP DATABASE** command.
It is strongly recommended that you create a backup copy of the database before you perform a data redistribution that is not roll-forward recoverable.
2. Connect to the database partition that contains the system catalog tables. See the **CONNECT** statement.
3. Issue the **REDISTRIBUTE DATABASE PARTITION GROUP** command.

Note: In previous versions of the DB2 database product, this command used the **NODEGROUP** keyword instead of the **DATABASE PARTITION GROUP** keywords.

Specify the following arguments:

database partition group name

You must specify the database partition group within which data is to be redistributed.

UNIFORM

OPTIONAL: Specifies that data is to be evenly distributed. **UNIFORM** is the default when no distribution-type is specified, so if no other distribution type has been specified, it is valid to omit this option.

USING DISTFILE *distfile-name*

OPTIONAL: Specifies that a customized distribution is desired and the file path name of a distribution file that contains data that defines the desired data skew. The contents of this file is used to generate a target distribution map.

USING TARGETMAP *targetmap-name*

OPTIONAL: Specifies that a target data redistribution map is to be used and the name of file that contains the target redistribution map.

For details, see the **REDISTRIBUTE DATABASE PARTITION GROUP** command-line utility information.

4. Allow the command to run uninterrupted. When the command completes, perform the following actions if the data redistribution proceeded successfully:
 - Take a backup of all table spaces in the database partition group that are in the BACKUP PENDING state. Alternatively, a full database backup can be performed.

Note: Table spaces are only put into the BACKUP PENDING state if the database is recoverable and the **NOT ROLLFORWARD RECOVERABLE** command parameter is used in the **REDISTRIBUTE DATABASE PARTITION GROUP** command.

- Recreate any replicated materialized query tables dropped before redistribution.
- Execute the **RUNSTATS** command if the following conditions are met:
 - The **STATISTICS NONE** command parameter was specified in the **REDISTRIBUTE DATABASE PARTITION GROUP** command, or the **NOT ROLLFORWARD RECOVERABLE** command parameter was omitted. Both of these conditions mean that the statistics were not collected during data redistribution.
 - There are tables in the database partition group possessing a statistics profile.

The **RUNSTATS** command collects data distribution statistics for the SQL compiler and optimizer to use when choosing data access plans for queries.

- If the **NOT ROLLFORWARD RECOVERABLE** command parameter was specified, delete the control files located in the following paths :
 - On Linux and UNIX operating systems: **diagpath/redist/db_name/db_partitiongroup_name/timestamp/**
 - On Windows operating systems: **diagpath\redist\db_name\db_partitiongroup_name\timestamp**

Results

Data redistribution is complete and information about the data redistribution process is available in the redistribution log file. Information about the distribution map that was used can be found in the DB2 explain tables.

Redistributing database partition groups using the STEPWISE_REDISTRIBUTE_DBPG procedure

Data redistribution can be performed using built-in procedures.

Procedure

To redistribute a database partition group using the STEPWISE_REDISTRIBUTE_DBPG procedure:

1. Analyze the database partition group regarding log space availability and data skew using the **ANALYZE_LOG_SPACE** procedure.
The **ANALYZE_LOG_SPACE** procedure returns a result set (an open cursor) of the log space analysis results, containing fields for each of the database partitions of the given database partition group.
2. Create a data distribution file for a given table using the **GENERATE_DISTFILE** procedure.
The **GENERATE_DISTFILE** procedure generates a data distribution file for the given table and saves it using the provided file name.

3. Create and report the content of a stepwise redistribution plan for the database partition group using the STEPWISE_REDISTRIBUTE_DBPG procedure.
4. Create a data distribution file for a given table using the GET_SWRD_SETTINGS and SET_SWRD_SETTINGS procedures.

The GET_SWRD_SETTINGS procedure reads the existing redistribute registry records for the given database partition group.

The SET_SWRD_SETTINGS procedure creates or makes changes to the redistribute registry. If the registry does not exist, it creates it and add records into it. If the registry already exists, it uses *overwriteSpec* to identify which of the field values need to be overwritten. The *overwriteSpec* field enables this function to take NULL inputs for the fields that do not need to be updated.

5. Redistribute the database partition group according to the plan using the STEPWISE_REDISTRIBUTE_DBPG procedure.

The STEPWISE_REDISTRIBUTE_DBPG procedure redistributes part of the database partition group according to the input and the setting file.

Example

The following is an example of a CLP script on AIX:

```
# -----
# Set the database you wish to connect to
# -----
dbName="SAMPLE"

# -----
# Set the target database partition group name
# -----
dbpgName="IBMDEFAULTGROUP"

# -----
# Specify the table name and schema
# -----
tbSchema="$USER"
tbName="STAFF"

# -----
# Specify the name of the data distribution file
# -----
distFile="$HOME/sql1lib/function/$dbName.IBMDEFAULTGROUP_swrdData.dst"

export DB2INSTANCE=$USER
export DB2COMM=TCPIP

# -----
# Invoke call statements in clp
# -----
db2start
db2 -v "connect to $dbName"

# -----
# Analysing the effect of adding a database partition without applying the changes - a 'what if'
# hypothetical analysis
#
# - In the following case, the hypothesis is adding database partition 40, 50 and 60 to the
#   database partition group, and for database partitions 10,20,30,40,50,60, using a respective
#   target ratio of 1:2:1:2:1:2.
#
# NOTE: in this example only partitions 10, 20 and 30 actually exist in the database
#       partition group
# -----
db2 -v "call sysproc.analyze_log_space('$dbpgName', '$tbSchema', '$tbName', 2, ' ',
'A', '40,50,60', '10,20,30,40,50,60', '1,2,1,2,1,2')"
```

```

# -----
# Analysing the effect of dropping a database partition without applying the changes
#
# - In the following case, the hypothesis is dropping database partition 30 from the database
#   partition group, and redistributing the data in database partitions 10 and 20 using a
#   respective target ratio of 1 : 1
#
# NOTE: In this example all database partitions 10, 20 and 30 should exist in the database
#       partition group
# -----
db2 -v "call sysproc.analyze_log_space('$dbpgName', '$tbSchema', '$tbName', 2, ' ',
'D', '30', '10,20','1,1')"

# -----
# Generate a data distribution file to be used by the redistribute process
# -----
db2 -v "call sysproc.generate_distfile('$tbSchema', '$tbName', '$distFile')"

# -----
# Write a step wise redistribution plan into a registry
#
# Setting the 10th parameter to 1, may cause a currently running step wise redistribute
# stored procedure to complete the current step and stop, until this parameter is reset
# to 0, and the redistribute stored procedure is called again.
# -----
db2 -v "call sysproc.set_swrd_settings('$dbpgName', 255, 0, ' ', '$distFile', 1000,
12, 2, 1, 0, '10,20,30', '50,50,50')"

# -----
# Report the content of the step wise redistribution plan for the given database
# partition group.
# -----
db2 -v "call sysproc.get_swrd_settings('$dbpgName', 255, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)"

# -----
# Redistribute the database partition group "$dbpgName" according to the redistribution
# plan stored in the registry by set_swrd_settings. It starting with step 3 and
# redistributes the data until 2 steps in the redistribution plan are completed.
# -----
db2 -v "call sysproc.stepwise_redistribute_dbpg('$dbpgName', 3, 2)"

```

Monitoring a data redistribution operation

You can use the **LIST UTILITIES** command to monitor the progress of data redistribution operations on a database.

Procedure

Issue the **LIST UTILITIES** command and specify the **SHOW DETAIL** parameter:

```
list utilities show detail
```

Results

The following is an example of the output from this command:

ID	= 1
Type	= REDISTRIBUTE
Database Name	= RDST819
Partition Number	= 11
Description	= RDST_V10_015 UNIFORM ADD NODES COMPACT ON SPACE REUSE RECORD LEVEL INDEXING MODE INCREMENTAL
Start Time	= 02-20-2007 23:21:33.785819
State	= Executing

```

Invocation Type          = User
Progress Monitoring:
Estimated Percentage Complete = 8
Summary:
  Total Work           = 1965600
  Completed Work       = 155221
  Total Number Of Tables = 15
  Tables Completed     = 0
  Tables In Progress   = 3

Current Table 1:
  Description          = "NEWTON  ."."RDST_V10_015A"
  Total Work            = 655200 bytes
  Completed Work        = 55001 bytes

Current Table 2:
  Description          = "NEWTON  ."."RDST_V10_015B"
  Total Work            = 450200 bytes
  Completed Work        = 54220 bytes

Current Table 3:
  Description          = "NEWTON  ."."RDST_V10_015C"
  Total Work            = 978901 bytes
  Completed Work        = 46000 bytes

```

Redistribution event log files

During data redistribution event logging is performed. Event information is logged to event log files which can later be used to perform error recovery.

When data redistribution is performed, information about each table that is processed is logged in a pair of event log files. The event log files are named *database-name.database-partition-group-name.timestamp.log* and *database-name.database-partition-group-name.timestamp*.

The log files are located as follows:

- The *homeinst/sqllib/redist* directory on Linux and UNIX operating systems
- The *db2instprof\instance\redist* directory on Windows operating systems, where *db2instprof* is the value of the **DB2INSTPROF** registry variable

The following is an example of the event log file names:

```
SAMPLE.IBMDEFAULTGROUP.2012012620240204
SAMPLE.IBMDEFAULTGROUP.2012012620240204.log
```

These files are for a redistribution operation on a database named SAMPLE with a database partition group named IBMDEFAULTGROUP. The files were created on January 26, 2012 at 8:24 PM local time.

The three main uses of the event log files are as follows:

- To provide general information about the redistribute operation, such as the old and new distribution maps.
- Provide users with information that helps them determine which tables have been redistributed so far by the utility.
- To provide information about each table that has been redistributed, including the indexing mode being used for the table, an indication of whether the table was successfully redistributed or not, and the starting and ending times for the redistribution operation on the table.

Recovery from errors related to data redistribution

Recovery from failures and errors that occur during data redistribution generally requires that you consult the redistribution event log file. The log file contains useful information about data redistribution processing, including information about which table or tables, if any, failed to be processed successfully.

Possible reasons that a redistribution might fail include:

- A documented prerequisite for successful data redistribution was not met.
- A documented restriction on data redistribution was encountered that resulted in the interruption of the data redistribution.
- During data redistribution a table to be processed was encountered in a restricted access state, such as LOAD PENDING.
- Any other problem documented in the event log.

When you have identified and resolved the cause of the failure, perform the recovery by using the same redistribution interfaces that were used when the operation failed. For example, if the **REDISTRIBUTE DATABASE PARTITION GROUP** command was used, once the problem has been addressed you can begin data redistribution again by reissuing the command with one of the following options:

CONTINUE

This option indicates that the redistribution operation should continue until all tables specified in the original **REDISTRIBUTE DATABASE PARTITION GROUP** command are redistributed.

ABORT This option indicates that the redistribution operation should be aborted and that all tables that have been redistributed, or partially redistributed, so far should be returned to the state they were in before the **REDISTRIBUTE DATABASE PARTITION GROUP** command was first run on the database partition group.

These options cannot be specified unless a previous data redistribution operation failed or completed without redistributing all tables in the database partition group. The latter case can occur if the **TABLE** command parameter is used and only a subset of tables is specified. In these cases, the value of the **REDISTRIBUTE_PMAP_ID** column in the **SYSCAT.DBPARTITIONGROUPS** table will have a value different from -1.

If an interface other than the **REDISTRIBUTE DATABASE PARTITION GROUP** command was used, continuation or abortion of data redistribution is possible by redistributing data again using the appropriate parameter value for the specified interface. See the reference information for the interface for the correct parameter values.

Examples of redistribute event log file entries

Examples of common redistribute event log files entries and descriptions of them provide a useful reference that can be consulted when troubleshooting errors or interruptions that occur during data redistribution.

There are two event log files created each time you redistribute a database partition group. The file named *database-name.database-partition-group-name.timestamp* provides a brief summary of the event. For example, *SAMPLE.IBMDEFAULTGROUP.2012012622083174* contains:

Data Redistribution Utility :

```

The following options have been specified :
Database partition group name : IBMDEFAULTGROUP
Data Redistribution option : U
Redistribute database partition group : uniformly
No. of partitions to be added : 0
List of partitions to be added :
No. of partitions to be dropped : 1
List of partitions to be dropped :
2
The execution of the Data Redistribution operation on :

    Begun at   Ended at   Table (poolID;objectId)

22.08.32      22.08.33   "DB2INST1"."CL_SCHED" (2;4)
22.08.33      22.08.35   "DB2INST1"."DEPARTMENT" (2;5)
22.08.35      22.08.36   "DB2INST1"."EMPLOYEE" (2;6)
...
22.09.13      22.09.15   "DB2INST1"."PRODUCTSUPPLIER" (4;10)
                                         "DB2INST1"."PRODUCTSUPPLIER" (4;10)

--All tables have been successfully redistributed.--
```

The file named *database-name.database-partition-group-name.timestamp.log* provides more detailed log entries. The following examples illustrate some common log file entries. Although each field value in the redistribute log file entries is not defined, the examples illustrate the main fields and most common or most important field values.

Example 1: First event log entry dumped during a redistribute operation

```

2012-01-26-22.08.32.607340-300 I1850E893          LEVEL: Event
PID     : 27700           TID : 46912984049984  PROC : db2sysc 0
INSTANCE: db2inst1        NODE : 000           DB   : SAMPLE
APPHDL  : 0-74            APPID: *N0.db2inst1...
AUTHID  : DB2INST1        HOSTNAME: ...
EDUID   : 65              EDUNAME: db2agent (SAMPLE) 0
FUNCTION: DB2 UDB, relation data serv, sqlrdrin, probe:3852
CHANGE   : DB PART MAP ID : IBMDEFAULTGROUP : FROM "1" : TO "3" : success
IMPACT   : None
DATA #1 : String, 24 bytes
HexDump Old Map Entries:
DATA #2 : Dumped object of size 65536 bytes at offset 0, 61 bytes
/home/db2inst1/sqlllib/redist/27700.65.000.dump.bin
DATA #3 : String, 24 bytes
HexDump New Map Entries:
DATA #4 : Dumped object of size 65536 bytes at offset 65672, 61 bytes
/home/db2inst1/sqlllib/redist/27700.65.000.dump.bin
```

The first event log entry dumped during a redistribute operation provides information about the distribution map files with which the utility will be working. In this case, the old distribution map for partition group IBMDEFAULTGROUP has an id of 1 and the new distribution map has an id of 3. A hexdump of each distribution map file is made to the redist directory in the instance path, and the names of the dump files are included in the entry. In this example, the file named 27700.65.000.dump.bin contains both distribution maps.

Example 2: Event log associated with the start of the redistribute operation

```
2012-01-26-22.08.32.625956-300 I2744E774          LEVEL: Event
PID    : 27700           TID : 46912984049984  PROC : db2sysc 0
INSTANCE: db2inst1       NODE : 000          DB   : SAMPLE
APPHDL : 0-74           APPID: *N0.db2inst1...
AUTHID : DB2INST1       HOSTNAME: ...
EDUID  : 65             EDUNAME: db2agent (SAMPLE) 0
FUNCTION: DB2 UDB, relation data serv, sqlrdrInitLogFile, probe:1802
START   : REDIST DB PART GROUP : IBMDEFAULTGROUP : success
IMPACT  : None
DATA #1 : String, 28 bytes
Partitioning Option: UNIFORM
DATA #2 : String, 23 bytes
Statistics: USE PROFILE
DATA #3 : String, 22 bytes
Indexing Mode: REBUILD
DATA #4 : String, 17 bytes
PRECHECK MODE: Y
DATA #5 : String, 16 bytes
QUIESCE MODE: Y
```

This entry indicates that the start of the redistribute operation has completed successfully and that redistribution of tables is about to begin. The partitioning option, statistics option, indexing mode, precheck mode, and quiesce mode to be used for the redistribution operation are also shown.

In this example, the partitioning option is UNIFORM, which indicates that data will be redistributed uniformly. Other possible values for this option include TARGETMAP, CONTINUE and ABORT.

The statistics option is USE PROFILE, which is the default statistics collection mode and means that if the table has a statistics profile, then statistics will be collected according to that profile. Otherwise, statistics will not be collected. If the value of this option is NONE, this indicates that the **STATISTICS NONE** option was specified, which means that no statistics are to be collected for the table, regardless of whether the table has a statistics profile defined or not.

In this example, the indexing mode is REBUILD, which is the default indexing mode and means that indexes on each table will be rebuilt during data redistribution. If the value of this option is DEFERRED, it means that the user specified the **INDEXING MODE DEFERRED** option, which results in indexes being marked invalid during redistribution. Such indexes must be rebuilt after data redistribution is complete.

In this example, the precheck mode is Y, which is the default precheck mode for data redistributions that are **NOT ROLLFORWARD RECOVERABLE**. This mode indicates that the redistribution operation begins only if the verification completes successfully.

In this example, the quiesce mode is Y, which is the default quiesce mode for data redistributions that are **NOT ROLLFORWARD RECOVERABLE**. This mode indicates that the redistribution operation forces all users off the database and puts it into a quiesced mode.

Example 3: Event log associated with start of redistributing a table

```
2012-01-26-22.08.32.843840-300 I4072E541          LEVEL: Event
PID    : 27700           TID : 46912874998080  PROC : db2sysc 0
INSTANCE: db2inst1       NODE : 000          DB   : SAMPLE
APPHDL : 0-113          APPID: *N0.DB2....
```

```

AUTHID : DB2INST1           HOSTNAME: ...
EDUID  : 181                 EDUNAME: db2agent (SAMPLE) 0
FUNCTION: DB2 UDB, database utilities - Redistribute, sqlurRedistributeTableByID
, probe:8743
START   : REDIST TABLE : "DB2INST1"."CL_SCHED" : success
IMPACT  : None

```

This entry indicates that the start of redistributing table "DB2INST1"."CL_SCHED" was successful.

Example 4: Event log associated with successful completion of redistribution for a table

```

2012-01-26-22.08.33.659785-300 I4614E541           LEVEL: Event
PID     : 27700              TID : 4691287499800  PROC : db2sysc 0
INSTANCE: db2inst1          NODE : 000          DB    : SAMPLE
APPHDL  : 0-113             APPID: *N0.DB2.....
AUTHID  : DB2INST1          HOSTNAME: ...
EDUID   : 181                EDUNAME: db2agent (SAMPLE) 0
FUNCTION: DB2 UDB, database utilities - Redistribute, sqlurRedistributeTableByID
, probe:9350
STOP    : REDIST TABLE : "DB2INST1"."CL_SCHED" : success
IMPACT  : None

```

This entry indicates that table "DB2INST1"."CL_SCHED" has been successfully redistributed. If an error had occurred during processing of this table, this entry would indicate failure.

Example 5: Event log associated with successful completion of redistribution of a database partition group

```

2012-01-26-22.09.16.746325-300 I28994E515           LEVEL: Event
PID     : 27700              TID : 46912984049984  PROC : db2sysc 0
INSTANCE: db2inst1          NODE : 000          DB    : SAMPLE
APPHDL  : 0-74               APPID: *N0.db2inst1....
AUTHID  : DB2INST1          HOSTNAME: ...
EDUID   : 65                 EDUNAME: db2agent (SAMPLE) 0
FUNCTION: DB2 UDB, relation data serv, sqlrdrdt, probe:1308
STOP    : REDIST DB PART GROUP : IBMDEFAULTGROUP : success
IMPACT  : None

```

This entry indicates that redistribution of database partition group IBMDEFAULTGROUP completed successfully. If the operation had not completed successfully, this entry would indicate failure.

These example entries can be a helpful reference when you consult your log files to resolve errors that occur during data redistribution or to verify that data redistribution complete successfully.

Scenario: Redistributing data in new database partitions

This scenario shows how to add new database partitions to a database and redistribute data between the database partitions. The **REDISTRIBUTE DATABASE PARTITION GROUP** command is demonstrated as part of showing how to redistribute data on different table sets within a database partition group.

About this task

Scenario:

A database DBPG1 has two database partitions, specified as (0, 1) and a database partition group definition (0, 1).

The following table spaces are defined on database partition group DBPG_1:

- Table space TS1 - this table space has two tables, T1 and T2
- Table space TS2 - this table space has three tables defined, T3, T4, and T5

Starting in Version 9.7, you can add database partitions while the database is running and while applications are connected to it. However, the operation can be performed offline in this scenario by changing the default value of the **DB2_FORCE_OFFLINE_ADD_PARTITION** registry variable to TRUE.

Procedure

To redistribute data between the database partitions in DBPG1:

1. Identify objects that must be disabled or removed before the redistribution.
 - a. Replicate MQTs: This type of MQT is not supported as part of the redistribution operation. They must be dropped before running the redistribution and recreated afterward.

```
SELECT tabschema, tablename
  FROM syscat.tables
 WHERE partition_mode = 'R'
```
 - b. Write-to-table event monitors: Disable any automatically activated write-to-table event monitors that have a table that resides in the database partition group to be redistributed.

```
SELECT distinct evmonname
  FROM syscat.eventtables E
  JOIN syscat.tables T on T.tablename = E.tablename
    AND T.tabschema = E.tabschema
  JOIN syscat.tablespaces S on S.tbspace = T.tbspace
    AND S.ngname = 'DBPG_1'
```
 - c. Explain tables: It is recommended to create the explain tables in a single partition database partition group. If they are defined in a database partition group that requires redistribution, however, and the data generated to date does not need to be maintained, consider dropping them. The explain tables can be redefined once the redistribution is complete.

```
CREATE EXPLAIN TABLES IN 'DBPG_1'
```
 - d. Table access mode and state: Ensure that all tables in the database partition groups to be redistributed are in full access mode and normal table states.

```
SELECT DISTINCT TRIM(T.OWNER) || '.' || TRIM(T.TABNAME)
  AS NAME, T.ACCESS_MODE, A.LOAD_STATUS
  FROM SYSCAT.TABLES T, SYSCAT.DBPARTITIONGROUPS
  N, SYSIBMADM.ADMINTABINFO A
 WHERE T.PMAP_ID = N.PMAP_ID
   AND A.TABSCHEMA = T.OWNER
   AND A.TABNAME = T.TABNAME
   AND N.DBPGNAME = 'DBPG_1'
   AND (T.ACCESS_MODE <> 'F' OR A.LOAD_STATUS IS NOT NULL)
```
 - e. Statistics profiles: If a statistics profile is defined for the table, table statistics can be updated as part of the redistribution process. Having the redistribution utility update the statistics for the table reduces I/O, as all the data is scanned for the redistribution and no additional scan of the data is needed for **RUNSTATS**.

```
RUNSTATS on table schema.table
  USE PROFILE runstats_profile
  SET PROFILE ONLY
```

- Review the database configuration. The **util_heap_sz** is critical to the data movement processing between database partitions - allocate as much memory as possible to **util_heap_sz** for the duration of the redistribution. Sufficient **sortheap** is required, if index rebuild is done as part of the redistribution. Increase **util_heap_sz** and **sortheap** as necessary to improve redistribution performance.
- Retrieve the database configuration settings to be used for the new database partitions. When adding database partitions, a default database configuration is used. As a result, it is important to update the database configuration on the new database partitions before the **REDISTRIBUTE DATABASE PARTITION GROUP** command is issued. This sequence of events ensures that the configuration is balanced.

```

SELECT name,
CASE WHEN deferred_value_flags = 'AUTOMATIC'
      THEN deferred_value_flags
      ELSE substr(deferred_value,1,20)
      END
AS deferred_value
FROM sysibmadm.dbcfg
WHERE dbpartitionnum = existing-node
  AND deferred_value != ''
  AND name NOT IN ('hadr_local_host','hadr_local_svc','hadr_peer_window',
    'hadr_remote_host','hadr_remote_inst','hadr_remote_svc',
    'hadr_syncmode','hadr_timeout','backup_pending','codepage',
    'codeset','collate_info','country','database_consistent',
    'database_level','hadr_db_role','log_retain_status',
    'loghead','logpath','multipage_alloc','numsegs','pagesize',
    'release','restore_pending','restrict_access',
    'rollfwd_pending','territory','user_exit_status',
    'number_compat','varchar2_compat','database_memory')
  )

```

- Back up the database (or the table spaces in the pertinent database partition group), before starting the redistribution process. This action ensures a recent recovery point.
- Add three new database partitions to the database. Issue the following commands:

```

START DBM DBPARTITIONNUM 3 ADD DBPARTITIONNUM HOSTNAME HOSTNAME3
PORT PORT3 WITHOUT TABLESPACES;
START DBM DBPARTITIONNUM 4 ADD DBPARTITIONNUM HOSTNAME HOSTNAME4
PORT PORT4 WITHOUT TABLESPACES;
START DBM DBPARTITIONNUM 5 ADD DBPARTITIONNUM HOSTNAME HOSTNAME5
PORT PORT5 WITHOUT TABLESPACES;

```

If the **DB2_FORCE_OFFLINE_ADD_PARTITION** is set to TRUE, new database partitions are not visible to the instance until it has been shut down and restarted. For example:

```

STOP DBM;
START DBM;

```

- Define system temporary table space containers on the newly defined database partitions.
 - Add the new database partitions to the database partition groups. The following command changes the DBPG_1 definition from (0, 1) to (0, 1, 3, 4, 5):
- ```

ALTER DATABASE PARTITION GROUP DBPG_1
ADD dbpartitionnums (3 to 5)
WITHOUT TABLESPACES

```

- Define permanent data table space containers on the newly defined database partitions.

```
ALTER TABLESPACE tablespace_name
 ADD container_information
 ON dbpartitionnums (3 to 5)
```

- Apply the database configuration settings to the new database partitions (or issue a single **UPDATE DB CFG** command against all database partitions).
- Capture the definition of and then drop any replicated MQTs existing in the database partition groups to be redistributed.

```
db2look -d DBPG1 -e -z
 schema -t replicated_MQT_table_names
 -o repMQTs.clp
```

- Disable any write-to-table event monitors that exist in the database partition groups to be redistributed.

```
SET EVENT MONITOR monitor_name STATE 0
```

- Run the redistribution utility to redistribute uniformly across all database partitions.

```
REDISTRIBUTE DATABASE PARTITION GROUP DBPG_1 NOT ROLLFORWARD RECOVERABLE
 UNIFORM STOP AT 2006-03-10-07.00.00.000000;
```

Let us presume that the command ran successfully for tables T1, T2 and T3, and then stopped due to the specification of the **STOP AT** option.

To abort the data redistribution for the database partition group and to revert the changes made to tables T1, T2, and T3, issue the following command:

```
REDISTRIBUTE DATABASE PARTITION GROUP DBPG_1
 NOT ROLLFORWARD RECOVERABLE ABORT;
```

You might abort the data redistribution when an error or an interruption occurs and you do not want to continue the redistribute operation. For this scenario, presume that this command was run successfully and that tables T1 and T2 were reverted to their original state.

To redistribute T5 and T4 only with 5000 4K pages as **DATA BUFFER**:

```
REDISTRIBUTE DATABASE PARTITION GROUP DBPG_1 NOT ROLLFORWARD RECOVERABLE
 UNIFORM TABLE (T5, T4) ONLY DATA BUFFER 5000;
```

If the command ran successfully, the data in tables T4 and T5 have been redistributed successfully.

To complete the redistribution of data on table T1, T2, and T3 in a specified order, issue:

```
REDISTRIBUTE DATABASE PARTITION GROUP DBPG_1 NOT ROLLFORWARD RECOVERABLE
 CONTINUE TABLE (T1) FIRST;
```

Specifying **TABLE (T1) FIRST** forces the database manager to process table T1 first so that it can return to being online (read-only) before other tables. All other tables are processed in an order determined by the database manager.

#### Note:

- The **ADD DBPARTITIONNUM** parameter can be specified in the **REDISTRIBUTE DATABASE PARTITION GROUP** command as an alternative to performing the **ALTER DATABASE PARTITION GROUP** and **ALTER TABLESPACE** statements in steps 7 on page 182 and 8. When a database partition is added by using this command parameter, containers for table spaces are based on the containers of the corresponding table space on the lowest numbered existing partition in the database partition group.
- The **REDISTRIBUTE DATABASE PARTITION GROUP** command in this example is not roll-forward recoverable.

- After the **REDISTRIBUTE DATABASE PARTITION GROUP** command finishes, all the table spaces it accessed will be left in the BACKUP PENDING state. Such table spaces must be backed up before the tables they contain are accessible for write operations.

For more information, refer to the “**REDISTRIBUTE DATABASE PARTITION GROUP** command”.

Consider also specifying a table list as input to the **REDISTRIBUTE DATABASE PARTITION GROUP** command to enforce the order that the tables are processed. The redistribution utility will move the data (compressed and compacted). Optionally, indexes will be rebuilt and statistics updated if statistics profiles are defined. Therefore instead of previous command, the following script can be run:

```
REDISTRIBUTE DATABASE PARTITION GROUP DBPG_1
 NOT ROLLFORWARD RECOVERABLE uniform
 TABLE (t1, t2,...) FIRST;
```

---

## Chapter 8. Multidimensional clustering tables

Multidimensional clustering (MDC) provides an elegant method for clustering data in tables along multiple dimensions in a flexible, continuous, and automatic way. MDC can significantly improve query performance.

In addition, MDC can significantly reduce the overhead of data maintenance, such as reorganization and index maintenance operations during insert, update, and delete operations. MDC is primarily intended for data warehousing and large database environments, but it can also be used in online transaction processing (OLTP) environments.

---

### New insert time clustering tables

Insert time clustering (ITC) tables provide an effective way of maintaining data clustering and easier management of space utilization.

ITC tables have similar characteristics to MDC tables. For example, these table types use block based allocation and block indexes. ITC and MDC tables differ in how data is clustered. ITC tables cluster data by using a virtual column which clusters rows, that are inserted at a similar time, together. Clustering dimensions on MDC tables are specified by the creator.

ITC tables are created with the CREATE TABLE command by specifying the ORGANIZE BY INSERT TIME clause.

A convenient, online way to convert existing tables to ITC tables is the ADMIN\_MOVE\_TABLE procedure. Another method to convert existing tables to ITC tables is export/import or a load from table. Existing tables cannot be altered to become ITC tables.

---

### Comparison of regular and MDC tables

Regular tables have indexes that are record-based. Any clustering of the indexes is restricted to a single dimension. Prior to Version 8, the database manager supported only single-dimensional clustering of data, through clustering indexes. Using a clustering index, the database manager attempts to maintain the physical order of data on pages in the key order of the index when records are inserted and updated in the table.

Clustering indexes greatly improve the performance of range queries that have predicates containing the key (or keys) of the clustering index. Performance is improved with a good clustering index because only a portion of the table needs to be accessed, and more efficient prefetching can be performed.

Data clustering using a clustering index has some drawbacks, however. First, because space is filled up on data pages over time, clustering is not guaranteed. An insert operation will attempt to add a record to a page nearby to those having the same or similar clustering key values, but if no space can be found in the ideal location, it will be inserted elsewhere in the table. Therefore, periodic table reorganizations may be necessary to re-cluster the table and to set up pages with additional free space to accommodate future clustered insert requests.

Second, only one index can be designated as the “clustering” index, and all other indexes will be unclustered, because the data can only be physically clustered along one dimension. This limitation is related to the fact that the clustering index is record-based, as all indexes have been prior to Version 8.1.

Third, because record-based indexes contain a pointer for every single record in the table, they can be very large in size.

### Clustering index

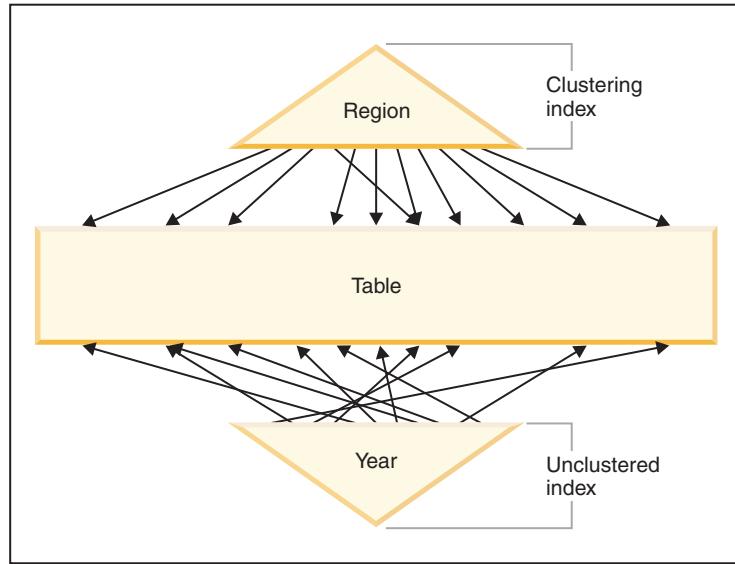


Figure 29. A regular table with a clustering index

The table in Figure 29 has two record-based indexes defined on it:

- A clustering index on “Region”
- Another index on “Year”

The “Region” index is a clustering index which means that as keys are scanned in the index, the corresponding records should be found for the most part on the same or neighboring pages in the table. In contrast, the “Year” index is unclustered which means that as keys are scanned in that index, the corresponding records will likely be found on random pages throughout the table. Scans on the clustering index will exhibit better I/O performance and will benefit more from sequential prefetching, the more clustered the data is to that index.

MDC introduces indexes that are block-based. “Block indexes” point to blocks or groups of records instead of to individual records. By physically organizing data in an MDC table into blocks according to clustering values, and then accessing these blocks using block indexes, MDC is able not only to address all of the drawbacks of clustering indexes, but to provide significant additional performance benefits.

First, MDC enables a table to be physically clustered on more than one key, or dimension, simultaneously. With MDC, the benefits of single-dimensional clustering are therefore extended to multiple dimensions, or clustering keys. Query performance is improved where there is clustering of one or more specified dimensions of a table. Not only will these queries access only those pages having records with the correct dimension values, these qualifying pages will be grouped into blocks, or extents.

Second, although a table with a clustering index can become unclustered over time, in most cases an MDC table is able to maintain and guarantee its clustering over all dimensions automatically and continuously. This eliminates the need to frequently reorganize MDC tables to restore the physical order of the data. While record order within blocks is always maintained, the physical ordering of blocks (that is, from one block to another, in a block index scan) is not maintained on inserts (or even on the initial load, in some cases).

Third, in MDC the clustering indexes are block-based. These indexes are drastically smaller than regular record-based indexes, so take up much less disk space and are faster to scan.

## Block indexes for MDC tables

This topic shows how records are organized in MDC tables using block indexes.

The MDC table shown in Figure 30 is physically organized such that records having the same “Region” and “Year” values are grouped together into separate blocks, or extents. An extent is a set of contiguous pages on disk, so these groups of records are clustered on physically contiguous data pages. Each table page belongs to exactly one block, and all blocks are of equal size (that is, an equal number of pages). The size of a block is equal to the extent size of the table space, so that block boundaries line up with extent boundaries. In this case, two block indexes are created, one for the “Region” dimension, and another for the “Year” dimension. These block indexes contain pointers only to the blocks in the table. A scan of the “Region” block index for all records having “Region” equal to “East” will find two blocks that qualify. All records, and only those records, having “Region” equal to “East” will be found in these two blocks, and will be clustered on those two sets of contiguous pages or extents. At the same time, and completely independently, a scan of the “Year” index for records between 1999 and 2000 will find three blocks that qualify. A data scan of each of these three blocks will return all records and only those records that are between 1999 and 2000, and will find these records clustered on the sequential pages within each of the blocks.

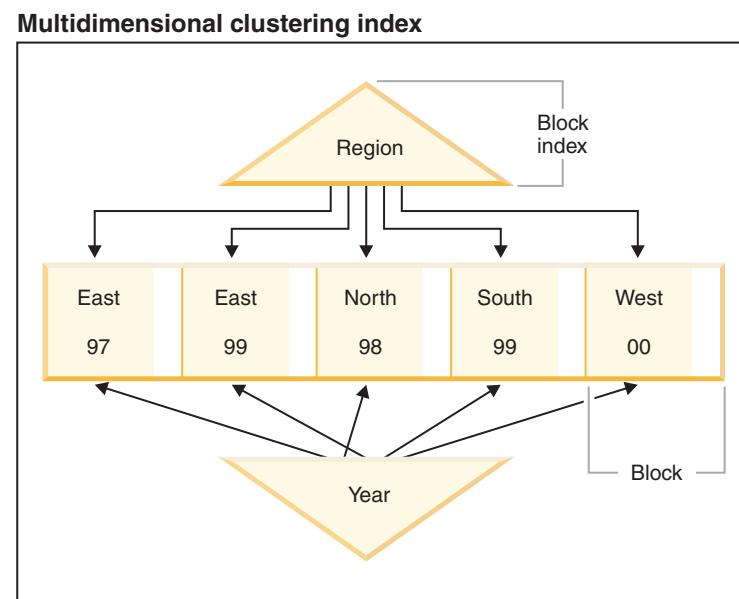


Figure 30. A multidimensional clustering table

In addition to these clustering improvements, MDC tables provide the following benefits:

- Probes and scans of block indexes are much faster due to their incredibly small size in relation to record-based indexes
- Block indexes and the corresponding organization of data allows for fine-grained “database partition elimination”, or selective table access
- Queries that utilize the block indexes benefit from the reduced index size, optimized prefetching of blocks, and guaranteed clustering of the corresponding data
- Reduced locking and predicate evaluation is possible for some queries
- Block indexes have much less overhead associated with them for logging and maintenance because they only need to be updated when adding the first record to a block, or removing the last record from a block
- Data rolled in can reuse the contiguous space left by data previously rolled out.

**Note:** An MDC table defined with even just a single dimension can benefit from these MDC attributes, and can be a viable alternative to a regular table with a clustering index. This decision should be based on many factors, including the queries that make up the workload, and the nature and distribution of the data in the table. Refer to “Choosing MDC table dimensions” on page 206 and “Creating MDC or ITC tables” on page 198.

When you create a table, you can specify one or more keys as dimensions along which to cluster the data. Each of these MDC dimensions can consist of one or more columns similar to regular index keys. A *dimension block index* will be automatically created for each of the dimensions specified, and it will be used by the optimizer to quickly and efficiently access data along each dimension. A *composite block index* will also automatically be created, containing all columns across all dimensions, and will be used to maintain the clustering of data over insert and update activity. A composite block index will only be created if a single dimension does not already contain all the dimension key columns. The composite block index may also be selected by the optimizer to efficiently access data that satisfies values from a subset, or from all, of the column dimensions.

**Note:** The usefulness of this index during query processing depends on the order of its key parts. The key part order is determined by the order of the columns encountered by the parser when parsing the dimensions specified in the ORGANIZE BY DIMENSIONS clause of the CREATE TABLE statement. Refer to “Block indexes for MDC and ITC tables” on page 205 for more information.

Block indexes are structurally the same as regular indexes, except that they point to blocks instead of records. Block indexes are smaller than regular indexes by a factor of the block size multiplied by the average number of records on a page. The number of pages in a block is equal to the extent size of the table space, which can range from 2 to 256 pages. The page size can be 4 KB, 8 KB, 16 KB, or 32 KB.

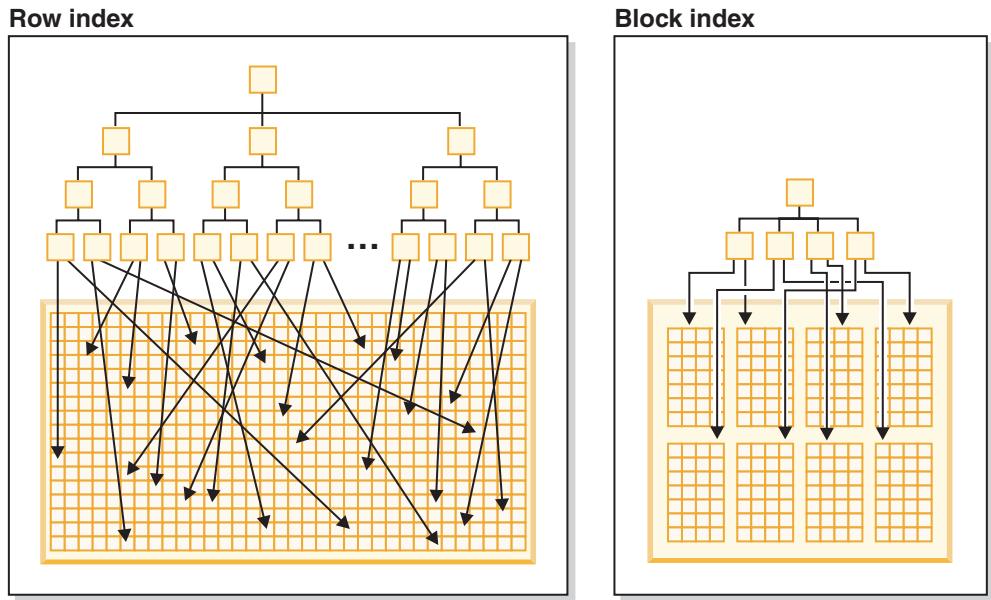


Figure 31. How row indexes differ from block indexes

As seen in Figure 31, in a block index there is a single index entry for each block compared to a single entry for each row. As a result, a block index provides a significant reduction in disk usage and significantly faster data access.

In an MDC table, every unique combination of dimension values form a logical *cell*, which may be physically made up of one or more blocks of pages. The logical cell will only have enough blocks associated with it to store the records having the dimension values of that logical cell. If there are no records in the table having the dimension values of a particular logical cell, no blocks will be allocated for that logical cell. The set of blocks that contain data having a particular dimension key value is called a *slice*.

An MDC table can be partitioned. The block index on a partitioned MDC table can be either nonpartitioned or partitioned:

- For a partitioned MDC table created with DB2 Version 9.7 Fix Pack 1 or later releases, the block indexes on the table are partitioned.
- For a partitioned MDC table created with DB2 V9.7 or earlier releases, the block indexes on the table are nonpartitioned.

Nonpartitioned block index are supported after upgrading the database to DB2 V9.7 Fix Pack 1 or later releases.

## Block indexes and query performance for MDC tables

Scans on any of the block indexes of an MDC table provide clustered data access, because each block identifier (BID) corresponds to a set of sequential pages in the table that is guaranteed to contain data having the specified dimension value. Moreover, dimensions or slices can be accessed independently from each other through their block indexes without compromising the cluster factor of any other dimension or slice. This provides the multidimensionality of multidimensional clustering.

Queries that take advantage of block index access can benefit from a number of factors that improve performance.

- Because block indexes are so much smaller than regular indexes, a block index scan is very efficient.
- Prefetching of data pages does not rely on sequential detection when block indexes are used. The DB2 database manager looks ahead in the index, prefetching blocks of data into memory using big-block I/O, and ensuring that the scan does not incur I/O costs when data pages are accessed in the table.
- The data in the table is clustered on sequential pages, optimizing I/O and localizing the result set to a selected portion of the table.
- If a block-based buffer pool is used, and the block size is equal to the extent size, MDC blocks are prefetched from sequential pages on disk into sequential pages in memory, further increasing the positive effect of clustering on performance.
- The records from each block are retrieved using a mini-relational scan of its data pages, which is often faster than scanning data through RID-based retrieval.

Queries can use block indexes to narrow down a portion of the table having a particular dimension value or range of values. This provides a fine-grained form of “database partition elimination”, that is, block elimination. This can translate into better concurrency for the table, because other queries, loads, inserts, updates and deletes may access other blocks in the table without interacting with this query's data set.

If the Sales table is clustered on three dimensions, the individual dimension block indexes can also be used to find the set of blocks containing records which satisfy a query on a subset of all of the dimensions of the table. If the table has dimensions of “YearAndMonth”, “Region” and “Product”, this can be thought of as a logical cube, as illustrated in Figure 32 on page 191.

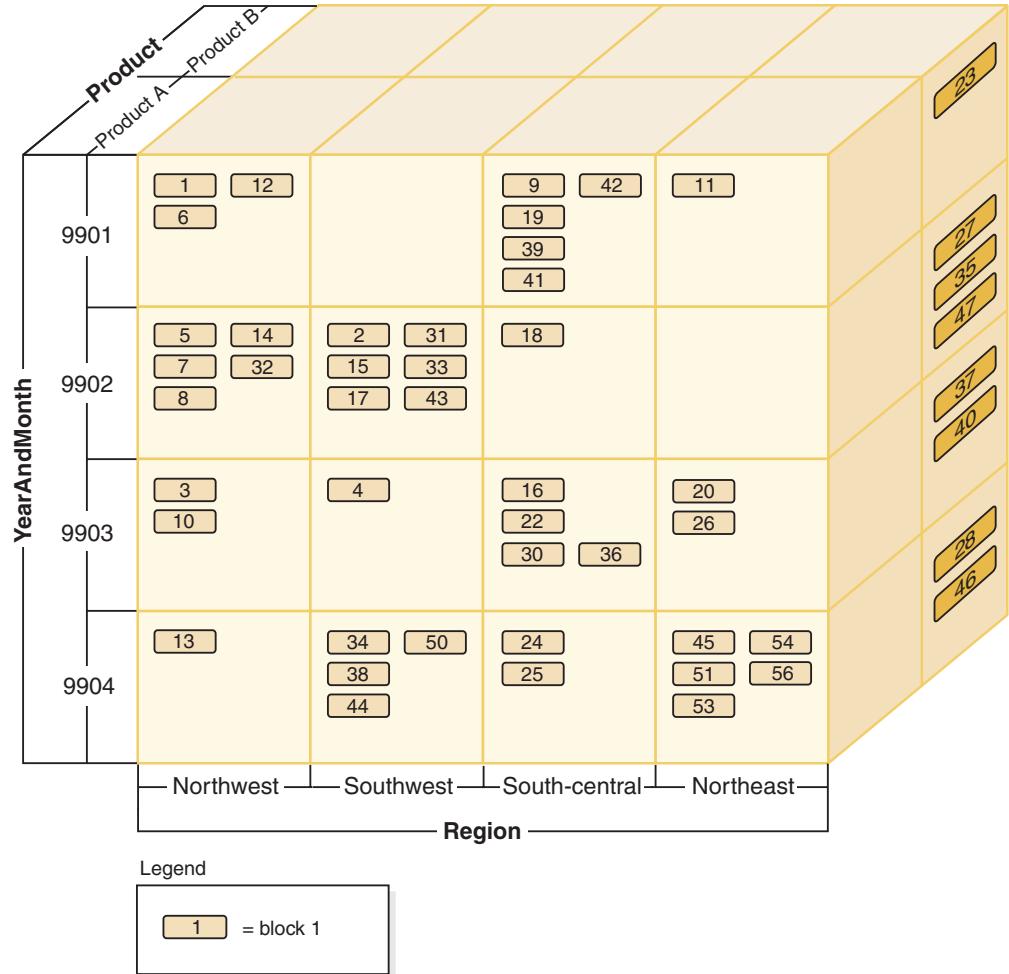


Figure 32. Multidimensional table with dimensions of 'Region', 'YearAndMonth', and 'Product'

Four block indexes will be created for the MDC table shown in Figure 32: one for each of the individual dimensions, "YearAndMonth", "Region", and "Product"; and another with all of these dimension columns as its key. To retrieve all records having a "Product" equal to "ProductA" and "Region" equal to "Northeast", the database manager would first search for the ProductA key from the "Product" dimension block index. (See Figure 33.) The database manager then determines the blocks containing all records having "Region" equal to "Northeast", by looking up the "Northeast" key in the "Region" dimension block index. (See Figure 34.)

|           |   |   |   |     |    |     |    |    |    |    |    |    |     |    |
|-----------|---|---|---|-----|----|-----|----|----|----|----|----|----|-----|----|
| Product A | 1 | 2 | 3 | ... | 11 | ... | 20 | 22 | 24 | 25 | 26 | 30 | ... | 56 |
|-----------|---|---|---|-----|----|-----|----|----|----|----|----|----|-----|----|

Figure 33. Key from dimension block index on 'Product'

|           |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Northeast | 11 | 20 | 23 | 26 | 27 | 28 | 35 | 37 | 40 | 45 | 46 | 47 | 51 | 53 | 54 | 56 |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|

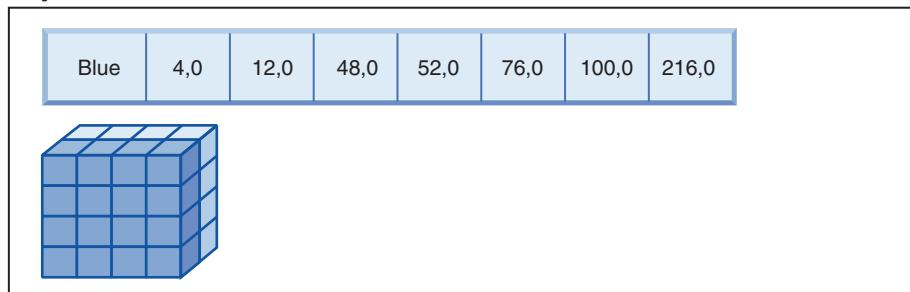
Figure 34. Key from dimension block index on 'Region'

Block index scans can be combined through the use of the logical AND and logical OR operators and the resulting list of blocks to scan also provides clustered data access.

Using the previous example, in order to find the set of blocks containing all records having both dimension values, you have to find the intersection of the two slices. This is done by using the logical AND operation on the BID lists from the two block index keys. The common BID values are 11, 20, 26, 45, 54, 51, 53, and 56.

The following example illustrates how to use the logical OR operation with block indexes to satisfy a query having predicates that involve two dimensions. Figure 35 assumes an MDC table where the two dimensions are “Colour” and “Nation”. The goal is to retrieve all those records in the MDC table that meet the conditions of having “Colour” of “blue” or having a “Nation” name “USA”.

#### Key from the dimension block index on Colour



**+** (OR)

#### Key from the dimension block index on Nation



**=**

#### Resulting block ID (BID) list of blocks to scan

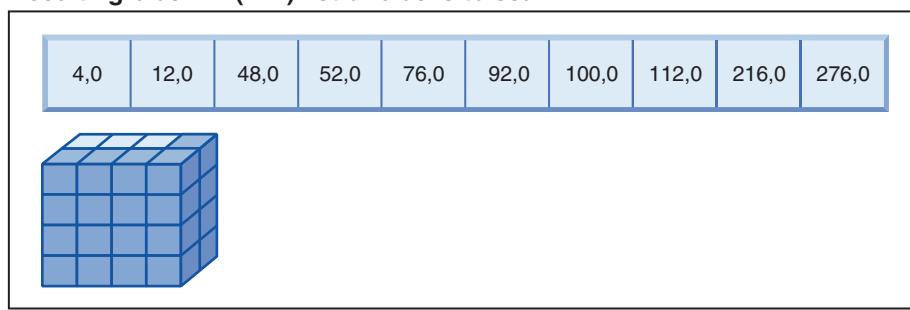


Figure 35. How the logical OR operation can be used with block indexes

This diagram shows how the result of two separate block index scans are combined to determine the range of values that meet the predicate restrictions. (The numbers indicate record identifiers (RIDs), slot fields.)

Based on the predicates from the SELECT statement, two separate dimension block index scans are done; one for the blue slice, and another for the USA slice. A logical OR operation is done in memory in order to find the union of the two slices, and determine the combined set of blocks found in both slices (including the removal of duplicate blocks).

Once the database manager has list of blocks to scan, the database manager can do a mini-relational scan of each block. Prefetching of the blocks can be done, and will involve just one I/O per block, as each block is stored as an extent on disk and can be read into the buffer pool as a unit. If predicates need to be applied to the data, dimension predicates need only be applied to one record in the block, because all records in the block are guaranteed to have the same dimension key values. If other predicates are present, the database manager only needs to check these on the remaining records in the block.

MDC tables also support regular RID-based indexes. Both RID and block indexes can be combined using a logical AND operation, or a logical OR operation, with the index. Block indexes provide the optimizer with additional access plans to choose from, and do not prevent the use of traditional access plans (RID scans, joins, table scans, and others). Block index plans will be costed by the optimizer along with all other possible access plans for a particular query, and the most inexpensive plan will be chosen.

The DB2 Design Advisor can help to recommend RID-based indexes on MDC tables, or to recommend MDC dimensions for a table.

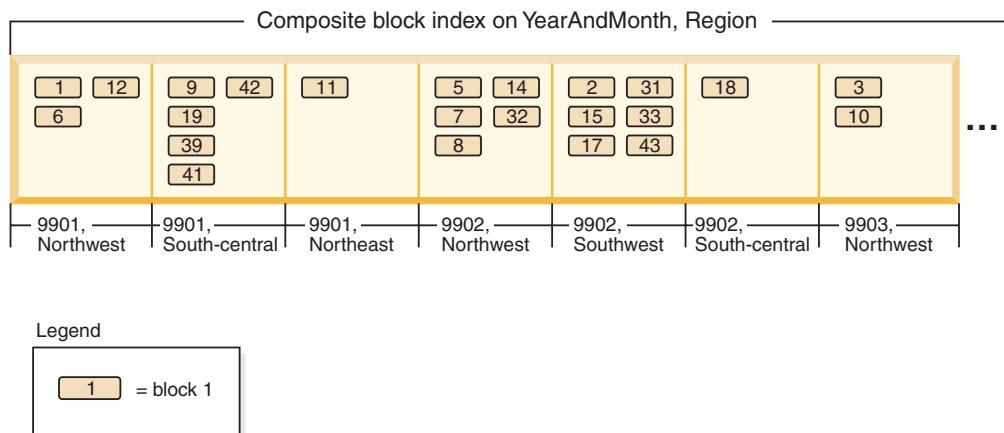
## **Maintaining clustering automatically during INSERT operations**

Automatic maintenance of data clustering in an MDC table is ensured using the composite block index. It is used to dynamically manage and maintain the physical clustering of data along the dimensions of the table over the course of INSERT operations.

A key is found in this composite block index only for each of those logical cells of the table that contain records. This block index is therefore used during an INSERT to quickly and efficiently determine if a logical cell exists in the table, and only if so, determine exactly which blocks contain records having that cell's particular set of dimension values.

When an insert occurs:

- The composite block index is probed for the logical cell corresponding to the dimension values of the record to be inserted.
- If the key of the logical cell is found in the index, its list of block ID (BIDs) gives the complete list of blocks in the table having the dimension values of the logical cell. (See Figure 36 on page 194.) This limits the numbers of extents of the table to search for space to insert the record.
- If the key of the logical cell is not found in the index; or, if the extents containing these values are full, a new block is assigned to the logical cell. If possible, the reuse of an empty block in the table occurs first before extending the table by another new extent of pages (a new block).



*Figure 36. Composite block index on 'YearAndMonth', 'Region'*

Data records having particular dimension values are guaranteed to be found in a set of blocks that contain only and all the records having those values. Blocks are made up of consecutive pages on disk. As a result, access to these records is sequential, providing clustering. This clustering is automatically maintained over time by ensuring that records are only inserted into blocks from cells with the record's dimension values. When existing blocks in a logical cell are full, an empty block is reused or a new block is allocated and added to the set of blocks for that logical cell. When a block is emptied of data records, the block ID (BID) is removed from the block indexes. This disassociates the block from any logical cell values so that it can be reused by another logical cell in the future. Thus, cells and their associated block index entries are dynamically added and removed from the table as needed to accommodate only the data that exists in the table. The composite block index is used to manage this, because it maps logical cell values to the blocks containing records having those values.

Because clustering is automatically maintained in this way, reorganization of an MDC table is never needed to re-cluster data. However, reorganization can still be used to reclaim space. For example, if cells have many sparse blocks where data could fit on fewer blocks, or if the table has many pointer-overflow pairs, a reorganization of the table would compact records belonging to each logical cell into the minimum number of blocks needed, as well as remove pointer-overflow pairs.

The following example illustrates how the composite block index can be used for query processing. If you want to find all records in the table in Figure 36 having "Region" of 'Northwest' and "YearAndMonth" of '9903', the database manager would look up the key value 9903, Northwest in the composite block index, as shown in Figure 37 on page 195. The key is made up a key value, namely '9903, Northwest', and a list of BIDs. You can see that the only BIDs listed are 3 and 10, and indeed there are only two blocks in the Sales table containing records having these two particular values.

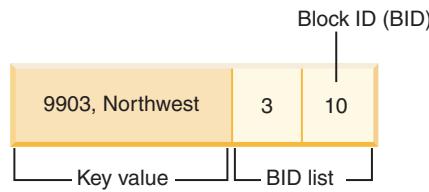


Figure 37. Key from composite block index on 'YearAndMonth', 'Region'

To illustrate the use of the composite block index during insert, take the example of inserting another record with dimension values 9903 and Northwest. The database manager would look up this key value in the composite block index and find BIDs for blocks 3 and 10. These blocks contain all records and the only records having these dimension key values. If there is space available, the database manager inserts the new record into one of these blocks. If there is no space on any pages in these blocks, the database manager allocates a new block for the table, or uses a previously emptied block in the table. Note that, in this example, block 48 is currently not in use by the table. The database manager inserts the record into the block and associates this block to the current logical cell by adding the BID of the block to the composite block index and to each dimension block index. See Figure 38 for an illustration of the keys of the dimension block indexes after the addition of Block 48.

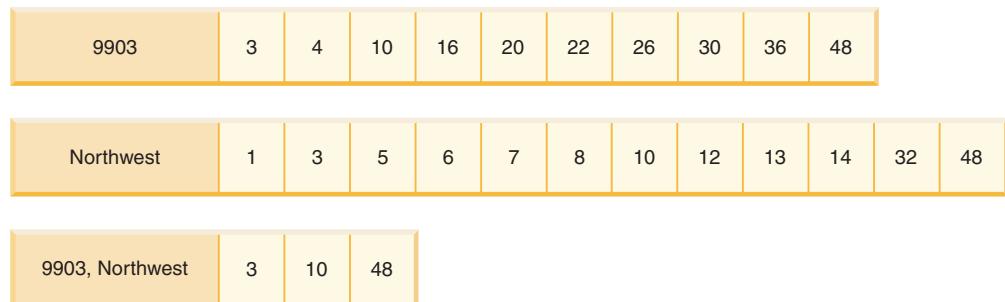


Figure 38. Keys from the dimension block indexes after addition of Block 48

## Block maps for MDC and ITC tables

For MDC tables, when a block is emptied, it is disassociated from its current logical cell values by removing its BID from the block indexes. The block can then be reused by another logical cell. For ITC tables, all blocks are associated with a single cell. Freeing a block within a cell means it can be reused by a subsequent insert. This reuse reduces the need to extend the table with new blocks.

When a new block is needed, previously emptied blocks need to be found quickly without having to search the table for them.

The block map is a structure used to facilitate locating empty blocks in the MDC or ITC table. The block map is stored as a separate object:

- In SMS, as a separate .BKM file
- In DMS, as a new object descriptor in the object table.

The block map is an array containing an entry for each block of the table. Each entry comprises a set of status bits for a block.

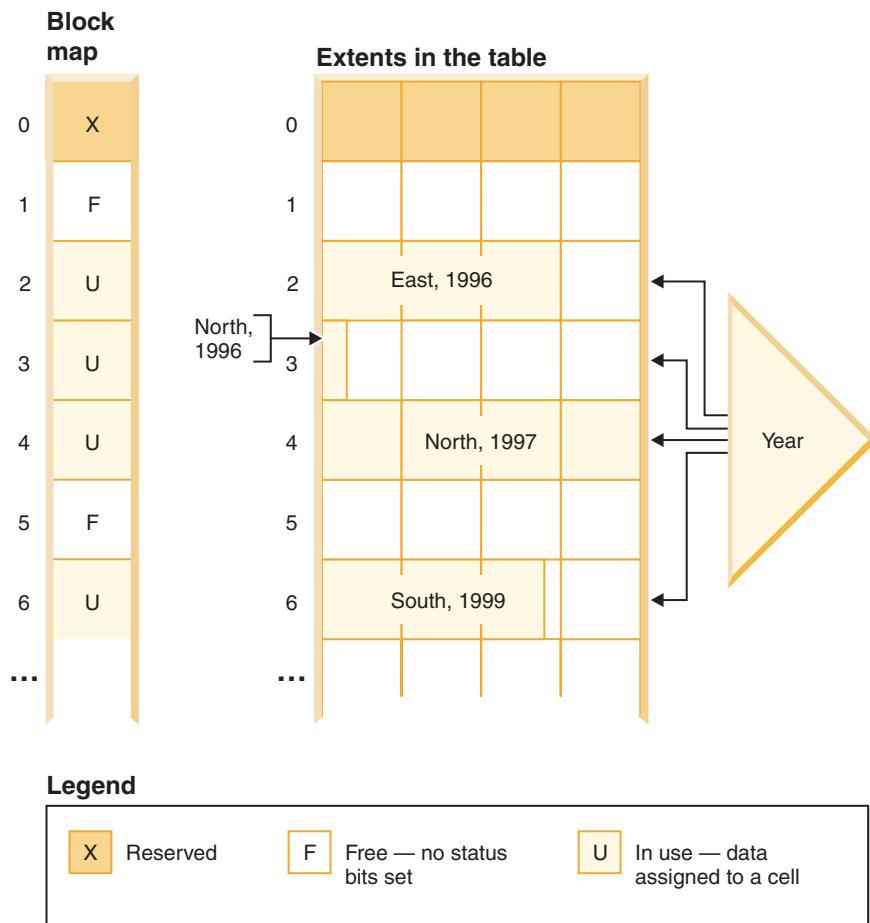


Figure 39. How a block map works

In Figure 39, the left side shows the block map array with different entries for each block in the table. The right side shows how each extent of the table is being used: some are free, most are in use, and records are only found in blocks marked in use in the block map. For simplicity, only one of the two dimension block indexes is shown in the diagram.

**Note:**

1. There are pointers in the block index only to blocks which are marked IN USE in the block map.
2. The first block is reserved. This block contains system records for the table.

Free blocks are found easily for use in a cell, by scanning the block map for FREE blocks, that is, blocks without any bits set.

Table scans also use the block map to access only extents currently containing data. Any extents not in use do not need to be included in the table scan at all. To illustrate, a table scan in this example (Figure 39) would start from the third extent (extent 2) in the table, skipping the first reserved extent and the subsequent empty extent, scan blocks 2, 3 and 4 in the table, skip the next extent (not touching the data pages of that extent), and then continue scanning from there.

---

## Updates to MDC and ITC tables

In an MDC table, updates of non-dimension values are done in place just as they are done with regular tables. If the update of a record in an MDC or ITC table causes the record to grow in length and it no longer fits on the page, another page with sufficient space is found.

The search for this new page begins within the same block. If there is no space in that block, the algorithm to insert a new record is used to find a page in the logical cell with enough space. There is no need to update the block indexes, unless no space is found in the cell and a new block needs to be added to the cell.

For an ITC table, if there is insufficient room in the block to place the updated row, the row is moved to a new block. This move causes the row to no longer be clustered with rows that were inserted at a similar time.

### Considerations for MDC tables only

Updates of dimension values are treated as a delete of the current record followed by an insert of the changed record, because the record is changing the logical cell to which it belongs. If the deletion of the current record causes a block to be emptied, the block index needs to be updated. Similarly, if the insert of the new record requires it to be inserted into a new block, the block index needs to be updated.

MDC tables are treated like any existing table; that is, triggers, referential integrity, views, and materialized query tables can all be defined upon them.

### Considerations for MDC and ITC tables

Block indexes need be only updated when inserting the first record into a block or when deleting the last record from a block. Index resources and requirements associated with block indexes for maintenance and logging is therefore much less than regular indexes. For every block index that would have otherwise been a regular index, the maintenance and logging resources and requirement is greatly reduced.

When you are reusing blocks that were recently made empty, a conditional Z lock on the block must be used to ensure that it is not currently being scanned by a UR scanner.

---

## Deleting from MDC and ITC tables

When a record is deleted in an MDC or ITC table, if it is not the last record in the block, the database manager merely deletes the record and removes its RID from any record-based indexes defined on the table.

When a delete removes the last record in a block, the database manager frees the block. The block is freed by changing the IN\_USE status bit and removing the BID of the block from all block indexes. If there are record-based indexes as well, the RID is removed from them.

**Note:** Therefore, block index entries are removed once per entire block and only if the block is emptied, instead of once per deleted row in a record-based index.

---

## Multidimensional and insert time clustering extent management

Freeing data extents from within the multidimensional (MDC) or insert time clustering (ITC) table is done through the reorganization of the table.

Within an MDC and ITC table, a block map tracks all the data extents belonging to a table and indicates which blocks and extents have data on them and which do not. Blocks with data are marked as being “in use”. Whenever deletions on MDC or ITC tables, or rollouts on MDC tables happen, block entries with the block map are no longer marked “in use” but rather are freed for reuse by the table.

However, these blocks and extents cannot be used by other objects within the table space. You can release these free data extents from the table through the reorganization of the table. You can use the **REORG TABLE** command with the **RECLAIM EXTENTS** parameter so the table is available and online to your users while space is reclaimed. The freeing of extents from the MDC or ITC table is only supported for tables in DMS table spaces.

The **REORG TABLE** command uses the **RECLAIM EXTENTS** parameter to free extents from exclusive use by the MDC or ITC table and makes the space available for use by other database objects within the table space.

The option also allows for your control of concurrent access to the MDC or ITC table while the extents are being freed. Write access is the default, read access and no access are also choices to control concurrent access.

If the MDC or ITC table is also range or database partitioned, by default the freeing of extents occurs on all data or database partitions. You can run the command to free extents only on a specific partition by specifying a partition name (for data partitions) or a partition number (for database partitions).

Both the **REORG TABLE** command and the db2Reorg API can be used to free extents.

Automatic support is available to make the freeing of extents part of your automatic maintenance activities for the database. To enable a reorganization to free extents in an MDC or ITC table, the **auto\_maint**, **auto\_tbl\_maint**, and **auto\_reorg** database configuration parameters must all have a value of ON. The configuring of these database configuration parameters can be carried out using the command line. On a DB2 instance where the database partitioning feature is enabled, the configuring of the parameters must be issued on the catalog partition.

A maintenance policy controls when an automatic reorganization of an MDC or ITC table takes place to free unused extents. The DB2 system stored procedures **AUTOMAINT\_SET\_POLICY** and **AUTOMAINT\_SET\_POLICYFILE** are used to set this maintenance policy. XML is used to store the automated maintenance policy.

---

## Creating MDC or ITC tables

There are many factors to consider when creating MDC or ITC tables. Decisions on how to create, place, and use your MDC or ITC tables can be influenced by your current database environment (for example, whether you have a partitioned database or not), and by your choice of dimensions.

## Moving data from existing tables to MDC tables

To improve query performance and reduce the requirements of data maintenance operations in a data warehouse or large database environment, you can move data from regular tables into multidimensional clustering (MDC) tables. To move data from an existing table to an MDC table:

1. export your data,
2. drop the original table (optional),
3. create a multidimensional clustering (MDC) table (using the CREATE TABLE statement with the ORGANIZE BY DIMENSIONS clause),
4. load the MDC table with your data.

An ALTER TABLE procedure called SYSPROC.ALTOBJ can be used to carry out the translation of data from an existing table to an MDC table. The procedure is called from the DB2 Design Advisor. The time required to translate the data between the tables can be significant and depends on the size of the table and the amount of data that needs to be translated.

The ALTOBJ procedure runs the following steps when altering a table:

1. drop all dependent objects of the table,
2. rename the table,
3. create the table with the new definition,
4. recreate all dependent objects of the table,
5. transform existing data in the table into the data required in the new table.  
That is, the selecting of data from the old table and loading that data into the new one where column functions can be used to transform from an old data type to a new data type.

## Moving data from existing tables to ITC tables

To reduce the requirements of data maintenance operations, you can move data from regular tables into insert time clustering (ITC) tables. To move data from an existing table to an ITC table use the online table move stored procedure.

The ExampleBank scenario shows how data from an existing table is moved into an ITC table. The scenario also shows how convenient reclaiming space is when using ITC tables. For more information, see the Related concepts links.

## MDC Advisor feature on the DB2 Design Advisor

The DB2 Design Advisor (**db2advis**) has an MDC feature. This feature recommends clustering dimensions for use in an MDC table, including coarsifications on base columns in order to improve workload performance. The term *coarsification* refers to a mathematical expression to reduce the cardinality (the number of distinct values) of a clustering dimension. A common example is coarsification by date, week of the date, month of the date, or quarter of the year.

A requirement to use the MDC feature of the DB2 Design Advisor is the existence of at least several extents of data within the database. The DB2 Design Advisor uses the data to model data density and cardinality.

If the database does not have data in the tables, the DB2 Design Advisor does not recommend MDC, even if the database contains empty tables but has a mocked up set of statistics to imply a populated database.

The recommendation includes identifying potential generated columns that define coarsification of dimensions. The recommendation does not include possible block sizes. The extent size of the table space is used when making recommendations for MDC tables. The assumption is that the recommended MDC table is created in the same table space as the existing table, and therefore has the same extent size. The recommendations for MDC dimensions change depending on the extent size of the table space, because the extent size affects the number of records that can fit into a block or cell. The extent size directly affects the density of the cells.

Only single-column dimensions, and not composite-column dimensions, are considered, although single or multiple dimensions might be recommended for the table. The MDC feature recommends coarsifications for most supported data types with the goal of reducing the cardinality of cells in the resulting MDC solution. The data type exceptions include: CHAR, VARCHAR, GRAPHIC, and VARGRAPHIC data types. All supported data types are cast to INTEGER and are coarsified through a generated expression.

The goal of the MDC feature of the DB2 Design Advisor is to select MDC solutions that result in improved performance. A secondary goal is to keep the storage expansion of the database constrained to a modest level. A statistical method is used to determine the maximum storage expansion on each table.

The analysis operation within the advisor includes not only the benefits of block index access but also the effect of MDC on insert, update, and delete operations against dimensions of the table. These actions on the table have the potential to cause records to be moved between cells. The analysis operation also models the potential performance effect of any table expansion resulting from the organization of data along particular MDC dimensions.

The MDC feature is run by using the `-m <advise type>` flag on the `db2advis` utility. The "C" advise type is used to indicate multidimensional clustering tables. The advise types are: "I" for index, "M" for materialized query tables, "C" for MDC, and "P" for partitioned database environment. The advise types can be used in combination with each other.

**Note:** The DB2 Design Advisor does not explore tables that are less than 12 extents in size.

The advisor analyzes both MQTs and regular base tables when coming up with recommendations.

The output from the MDC feature includes:

- Generated column expressions for each table for coarsified dimensions that appear in the MDC solution.
- An ORGANIZE BY DIMENSIONS clause recommended for each table.

The recommendations are reported both to `stdout` and to the ADVISE tables that are part of the explain facility.

## MDC tables and partitioned database environments

Multidimensional clustering can be used in a partitioned database environment. In fact, MDC can complement a partitioned database environment. A partitioned database environment is used to distribute data from a table across multiple physical or logical database partitions to:

- take advantage of multiple machines to increase processing requests in parallel,
- increase the physical size of the table beyond the limits of a single database partition,
- improve the scalability of the database.

The reason for distributing a table is independent of whether the table is an MDC table or a regular table. For example, the rules for the selection of columns to make up the distribution key are the same. The distribution key for an MDC table can involve any column, whether those columns make up part of a dimension of the table or not.

If the distribution key is identical to a dimension from the table, then each database partition contains a different portion of the table. For instance, if our example MDC table is distributed by color across two database partitions, then the Color column is used to divide the data. As a result, the Red and Blue slices might be found on one database partition and the Yellow slice on the other. If the distribution key is not identical to the dimensions from the table, then each database partition has a subset of data from each slice. When choosing dimensions and estimating cell occupancy, note that on average the total amount of data per cell is determined by taking all of the data and dividing by the number of database partitions.

## **MDC tables with multiple dimensions**

If you know that certain predicates are heavily used in queries, you can cluster the table on the columns involved. You can do this by using the ORGANIZE BY DIMENSIONS clause.

Example 1:

```
CREATE TABLE T1 (c1 DATE, c2 INT, c3 INT, c4 DOUBLE)
 ORGANIZE BY DIMENSIONS (c1, c3, c4)
```

The table in Example 1 is clustered on the values within three columns forming a logical cube (that is, having three dimensions). The table can now be logically sliced up during query processing on one or more of these dimensions such that only the blocks in the appropriate slices or cells are processed by the relational operators involved. The size of a block (the number of pages) is the extent size of the table.

## **MDC tables with dimensions based on more than one column**

Each dimension can be made up of one or more columns. As an example, you can create a table that is clustered on a dimension containing two columns.

Example 2:

```
CREATE TABLE T1 (c1 DATE, c2 INT, c3 INT, c4 DOUBLE)
 ORGANIZE BY DIMENSIONS (c1, (c3, c4))
```

In Example 2, the table is clustered on two dimensions, c1 and (c3,c4). Thus, in query processing, the table can be logically sliced up on either the c1 dimension, or on the composite (c3, c4) dimension. The table has the same number of blocks as the table in Example 1, but one less dimension block index. In Example 1, there are three dimension block indexes, one for each of the columns c1, c3, and c4. In Example 2, there are two dimension block indexes, one on the column c1 and the other on the columns c3 and c4. The main difference between the two approaches

is that, in Example 1, queries involving c4 can use the dimension block index on c4 to quickly and directly access blocks of relevant data. In Example 2, c4 is a second key part in a dimension block index, so queries involving c4 involve more processing. However, in Example 2 there is one less block index to maintain and store.

The DB2 Design Advisor does not make recommendations for dimensions containing more than one column.

## MDC tables with column expressions as dimensions

Column expressions can also be used for clustering dimensions. The ability to cluster on column expressions is useful for rolling up dimensions to a coarser granularity, such as rolling up an address to a geographic location or region, or rolling up a date to a week, month, or year. To implement the rolling up of dimensions in this way, you can use generated columns. This type of column definition allows the creation of columns using expressions that can represent dimensions. In Example 3, the statement creates a table clustered on one base column and two column expressions.

Example 3:

```
CREATE TABLE T1(c1 DATE, c2 INT, c3 INT, c4 DOUBLE,
 c5 DOUBLE GENERATED ALWAYS AS (c3 + c4),
 c6 INT GENERATED ALWAYS AS (MONTH(C1)))
 ORGANIZE BY DIMENSIONS (c2, c5, c6)
```

In Example 3, column c5 is an expression based on columns c3 and c4, and column c6 rolls up column c1 to a coarser granularity in time. The statement clusters the table based on the values in columns c2, c5, and c6.

## Range queries on generated column dimensions

Range queries on a generated column dimension require monotonic column functions. Expressions must be monotonic to derive range predicates for dimensions on generated columns. If you create a dimension on a generated column, queries on the base column are able to take advantage of the block index on the generated column to improve performance, with one exception. For range queries on the base column (date, for example) to use a range scan on the dimension block index, the expression used to generate the column in the CREATE TABLE statement must be monotonic. Although a column expression can include any valid expression (including user-defined functions (UDFs)), if the expression is non-monotonic, only equality or IN predicates are able to use the block index to satisfy the query when these predicates are on the base column.

As an example, assume that you create an MDC table with dimensions on the generated column month, where  $\text{month} = \text{INTEGER}(\text{date})/100$ . For queries on the dimension (month), block index scans can be done. For queries on the base column (date), block index scans can also be done to narrow down which blocks to scan, and then apply the predicates on date to the rows in those blocks only.

The compiler generates additional predicates to be used in the block index scan. For example, with the query:

```
SELECT * FROM MDCTABLE WHERE DATE > "1999-03-03" AND DATE < "2000-01-15"
```

the compiler generates the additional predicates: “month >= 199903” and “month <= 200001” which can be used as predicates for a dimension block index scan. When scanning the resulting blocks, the original predicates are applied to the rows in the blocks.

A non-monotonic expression allows equality predicates to be applied to that dimension. A good example of a non-monotonic function is MONTH( ) as seen in the definition of column c6 in Example 3. If the c1 column is a date, timestamp, or valid string representation of a date or timestamp, then the function returns an integer value in the range of 1 to 12. Even though the output of the function is deterministic, it actually produces output similar to a step function (that is, a cyclic pattern):

```
MONTH(date('01/05/1999')) = 1
MONTH(date('02/08/1999')) = 2
MONTH(date('03/24/1999')) = 3
MONTH(date('04/30/1999')) = 4
...
MONTH(date('12/09/1999')) = 12
MONTH(date('01/18/2000')) = 1
MONTH(date('02/24/2000')) = 2
...
```

Although date in this example is continually increasing, MONTH(date) is not. More specifically, it is not guaranteed that whenever date1 is larger than date2, MONTH(date1) is greater than or equal to MONTH(date2). It is this condition that is required for monotonicity. This non-monotonicity is allowed, but it limits the dimension in that a range predicate on the base column cannot generate a range predicate on the dimension. However, a range predicate on the expression is fine, for example, where month(c1) between 4 and 6. This can use the index on the dimension in the typical way, with a starting key of 4 and a stop key of 6.

To make this function monotonic, include the year as the high-order part of the month. There is an extension to the INTEGER built-in function to help in defining a monotonic expression on date. INTEGER(date) returns an integer representation of the date, which then can be divided to find an integer representation of the year and month. For example, INTEGER(date('2000/05/24')) returns 20000524, and therefore INTEGER(date('2000/05/24'))/100 = 200005. The function INTEGER(date)/100 is monotonic.

Similarly, the built-in functions DECIMAL and BIGINT also have extensions so that you can derive monotonic functions. DECIMAL(timestamp) returns a decimal representation of a timestamp, and this can be used in monotonic expressions to derive increasing values for month, day, hour, minute, and so on. BIGINT(date) returns a big integer representation of the date, similar to INTEGER(date).

The database manager determines the monotonicity of an expression, where possible, when creating the generated column for the table, or when creating a dimension from an expression in the dimensions clause. Certain functions can be recognized as monotonicity-preserving, such as DAYS( ) or YEAR( ). Also, various mathematical expressions such as division, multiplication, or addition of a column and a constant are monotonicity-preserving. Where DB2 determines that an expression is not monotonicity-preserving, or if it cannot determine this, the dimension supports only the use of equality predicates on its base column.

---

## Load for MDC and ITC tables

If you roll data in to your data warehouse on a regular basis, you can use multidimensional clustering (MDC) tables to your advantage. In MDC tables, load first reuses previously emptied blocks in the table before extending the table and adding new blocks for the remaining data.

After you delete a set of data, for example, all the data for a month, you can use the load utility to roll in the next month of data and it can reuse the blocks that were emptied after the (committed) deletion. You can also choose to use the MDC rollout feature with deferred cleanup. After the rollout, which is also a deletion, is committed, the blocks are not free and cannot yet be reused. A background process is invoked to maintain the record ID (RID) based indexes. When the maintenance is complete, the blocks are freed and can be reused. For insert time clustering (ITC) tables, blocks that are not in use are reused where possible before the table is extended. This includes blocks that were reclaimed. Rollout is not supported on ITC tables.

When loading data into MDC tables, the input data can be either sorted or unsorted. If unsorted, and the table has more than one dimension, consider doing the following:

- Increase the **util\_heap\_sz** configuration parameter.

To improve the performance of the load utility when loading MDC tables, increase the **util\_heap\_sz** database configuration parameter value. The mdc-load algorithm performs better when more memory is available to the utility. This reduces disk I/O during the clustering of data that is performed during the load phase. If the **LOAD** command is being used to load several MDC tables concurrently, **util\_heap\_sz** must be increased accordingly.

- Increase the value given with the **DATA BUFFER** clause of the **LOAD** command.

Increasing this value affects a single load request. The utility heap size must be large enough to accommodate the possibility of multiple concurrent load requests. Beginning in version 9.5, the value of the **DATA BUFFER** parameter of the **LOAD** command can temporarily exceed **util\_heap\_sz** if more memory is available in the system.

- Ensure the page size used for the buffer pool is the same as the largest page size for the temporary table space.

During the load phase, extra logging for the maintenance of the block map is performed. There are approximately two extra log records per extent allocated. To ensure good performance, the **logbufsz** database configuration parameter must be set to a value that takes this into account.

The following restrictions apply when loading data into MDC or ITC tables:

- The **SAVECOUNT** parameter in the **LOAD** command is not supported.
- The **totalfreespace** file type modifier is not supported since these tables manage their own free space.
- The **anyorder** file type modifier is required for MDC or ITC tables. If a load is executed into an MDC or ITC table without the **anyorder** modifier, it is explicitly enabled by the utility.

When using the **LOAD** command with an MDC or ITC table, violations of unique constraints are handled as follows:

- If the table included a unique key before the load operation and duplicate records are loaded into the table, the original record remains and the new records are deleted during the delete phase.
- If the table did not include a unique key prior to the load operation and both a unique key and duplicate records are loaded into the table, only one of the records with the unique key is loaded and the others are deleted during the delete phase.

**Note:** There is no explicit technique for determining which record is loaded and which is deleted.

Load begins at a block boundary, so it is best used for data belonging to new cells, for the initial populating of a table, and for loading additional data into ITC tables.

MDC and ITC load operations always have a build phase since all MDC and ITC tables have block indexes.

## Logging considerations for MDC and ITC tables

Index maintenance and logging is reduced when dimensions and therefore block indexes are used, as compared to cases where RID indexes are used.

The database manager removes the BID from the block indexes only when the last record in an entire block is deleted. This index operation is also logged at this time. Similarly, the database manager inserts a BID into the block index only when a record is inserted into a new block. That record must be the first record of a logical cell or an insert to a logical cell of blocks that are currently full. This index operation is also logged at this time.

Because blocks can be 2 - 256 pages of records, this block index maintenance and logging is relatively small. Inserts and deletes to the table and to RID indexes are still logged. For roll out deletions, the deleted records are not logged. Instead, the pages that contain the records are made to look empty by reformatting parts of the pages. The changes to the reformatted parts are logged, but the records themselves are not logged.

## Block indexes for MDC and ITC tables

Dimension block indexes are created when you define dimensions for a multidimensional clustering (MDC) table. A composite block index is created when you define multiple dimensions. If you define only one dimension for your MDC table, or if your table is an insert time clustering (ITC) table, the database manager creates only one block index, which serves as both the dimension block index and the composite block index. If your MDC or ITC table is partitioned, the block index is also partitioned.

If you create an MDC table that has dimensions on column A and on (column A, column B), the database manager creates a dimension block index on column A and a dimension block index on (column A, column B). Because a composite block index is a block index of all the dimensions in the table, the dimension block index on (column A, column B) also serves as the composite block index.

The composite block index for an MDC table is used in query processing to access data with specific dimension values. The order of key parts in the composite block index has no effect on insert processing, but might affect its use or applicability for query processing. The order of key parts is determined by the order of columns in the ORGANIZE BY DIMENSIONS clause when the MDC table is created.

Multicolumn dimensions in the ORGANIZE BY DIMENSION clause take precedence when there is a duplicate. For example, if a table is created by using the following statement, the composite block index is created on columns (c4, c3, c1, c2).

```
CREATE TABLE t1 (c1 int, c2 int, c3 int, c4 int)
 ORGANIZE BY DIMENSIONS (c1, c4, (c3, c1), c2)
```

Although c1 is specified twice in the ORGANIZE BY DIMENSIONS clause, it is used only once as a key part for the composite block index; (c3, c1) replaces (c1). The following example shows you how to create a table whose composite block index has a column order of (c1, c2, c3, c4):

```
CREATE TABLE t1 (c1 int, c2 int, c3 int, c4 int)
 ORGANIZE BY DIMENSIONS (c2, c1, (c2, c3), c4)
```

## Choosing MDC table dimensions

After you have decided to work with multidimensional clustering tables, the dimensions that you choose will depend not only on the type of queries that will use the tables and benefit from block-level clustering, but even more importantly on the amount and distribution of your actual data.

### Queries that will benefit from MDC

The first consideration when choosing clustering dimensions for your table is the determination of which queries will benefit from clustering at a block level. Typically, there will be several candidates when choosing dimensions based on the queries that make up the work to be done on the data. The ranking of these candidates is important. Columns that are involved in equality or range predicate queries, and especially columns with low cardinalities, show the greatest benefit from clustering dimensions. Consider creating dimensions for foreign keys in an MDC fact table involved in star joins with dimension tables. Keep in mind the performance benefits of automatic and continuous clustering on more than one dimension, and of clustering at the extent or block level.

There are many queries that can take advantage of multidimensional clustering. Examples of such queries follow. In some of these examples, assume that there is an MDC table t1 with dimensions c1, c2, and c3. In the other examples, assume that there is an MDC table mdctable with dimensions color and nation.

#### Example 1:

```
SELECT FROM t1 WHERE c3 < 5000
```

This query involves a range predicate on a single dimension, so it can be internally rewritten to access the table using the dimension block index on c3. The index is scanned for block identifiers (BIDs) of keys having values less than 5000, and a mini-relational scan is applied to the resulting set of blocks to retrieve the actual records.

#### Example 2:

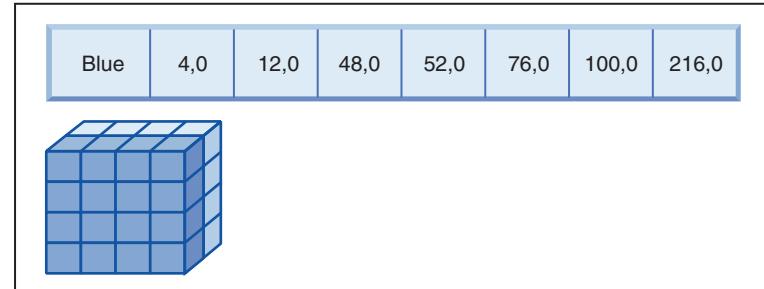
```
SELECT FROM t1 WHERE c2 IN (1,2037)
```

This query involves an IN predicate on a single dimension, and can trigger block index based scans. This query can be internally rewritten to access the table using the dimension block index on c2. The index is scanned for BIDs of keys having values of 1 and 2037, and a mini-relational scan is applied to the resulting set of blocks to retrieve the actual records.

Example 3:

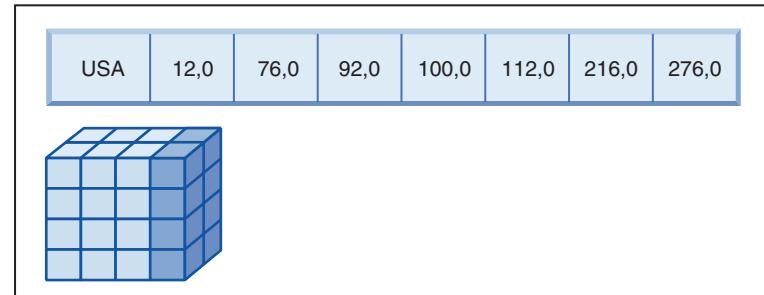
```
SELECT * FROM MDCTABLE WHERE COLOR='BLUE' AND NATION='USA'
```

**Key from the dimension block index on Colour**



**+ (AND)**

**Key from the dimension block index on Nation**



=

**Resulting block ID (BID) list of blocks to scan**

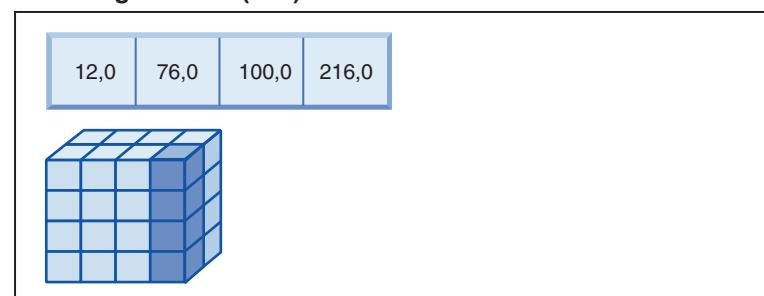


Figure 40. A query request that uses a logical AND operation with two block indexes

To carry out this query request, the following is done (and is shown in Figure 40):

- A dimension block index lookup is done: one for the Blue slice and another for the USA slice.
- A block logical AND operation is carried out to determine the intersection of the two slices. That is, the logical AND operation determines only those blocks that are found in both slices.
- A mini-relation scan of the resulting blocks in the table is carried out.

Example 4:

```
SELECT ... FROM t1
WHERE c2 > 100 AND c1 = '16/03/1999' AND c3 > 1000 AND c3 < 5000
```

This query involves range predicates on c2 and c3 and an equality predicate on c1, along with a logical AND operation. This can be internally rewritten to access the table on each of the dimension block indexes:

- A scan of the c2 block index is done to find BIDs of keys having values greater than 100
- A scan of the c3 block index is done to find BIDs of keys having values between 1000 and 5000
- A scan of the c1 block index is done to find BIDs of keys having the value '16/03/1999'.

A logical AND operation is then done on the resulting BIDs from each block scan, to find their intersection, and a mini-relational scan is applied to the resulting set of blocks to find the actual records.

**Example 5:**

```
SELECT * FROM MDCTABLE WHERE COLOR='BLUE' OR NATION='USA'
```

To carry out this query request, the following is done:

- A dimension block index lookup is done: one for each slice.
- A logical OR operation is done to find the union of the two slices.
- A mini-relation scan of the resulting blocks in the table is carried out.

**Example 6:**

```
SELECT FROM t1 WHERE c1 < 5000 OR c2 IN (1,2,3)
```

This query involves a range predicate on the c1 dimension, an IN predicate on the c2 dimension, and a logical OR operation. This can be internally rewritten to access the table on the dimension block indexes c1 and c2. A scan of the c1 dimension block index is done to find values less than 5000 and another scan of the c2 dimension block index is done to find values 1, 2, and 3. A logical OR operation is done on the resulting BIDs from each block index scan, then a mini-relational scan is applied to the resulting set of blocks to find the actual records.

**Example 7:**

```
SELECT FROM t1 WHERE c1 = 15 AND c4 < 12
```

This query involves an equality predicate on the c1 dimension and another range predicate on a column that is not a dimension, along with a logical AND operation. This can be internally rewritten to access the dimension block index on c1, to get the list of blocks from the slice of the table having value 15 for c1. If there is a RID index on c4, an index scan can be done to retrieve the RIDs of records having c4 less than 12, and then the resulting list of blocks undergoes a logical AND operation with this list of records. This intersection eliminates RIDs not found in the blocks having c1 of 15, and only those listed RIDs found in the blocks that qualify are retrieved from the table.

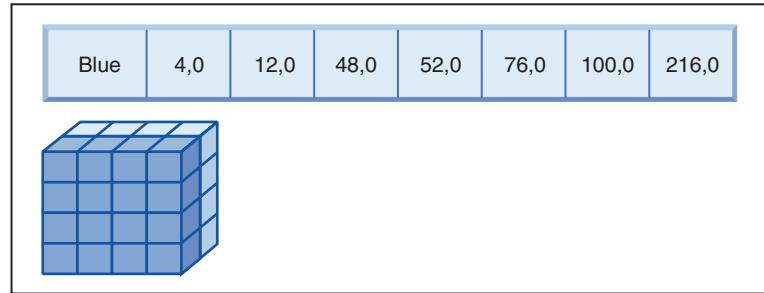
If there is no RID index on c4, then the block index can be scanned for the list of qualifying blocks, and during the mini-relational scan of each block, the predicate  $c4 < 12$  can be applied to each record found.

**Example 8:**

Given a scenario where there are dimensions for color, year, nation and a row ID (RID) index on the part number, the following query is possible.

```
SELECT * FROM MDCTABLE WHERE COLOR='BLUE' AND PARTNO < 1000
```

#### Key from the dimension block index on Colour



**+** (AND)

#### Row IDs (RID) from RID index on Partno



=

#### Resulting row IDs to fetch



Figure 41. A query request that uses a logical AND operation on a block index and a row ID (RID) index

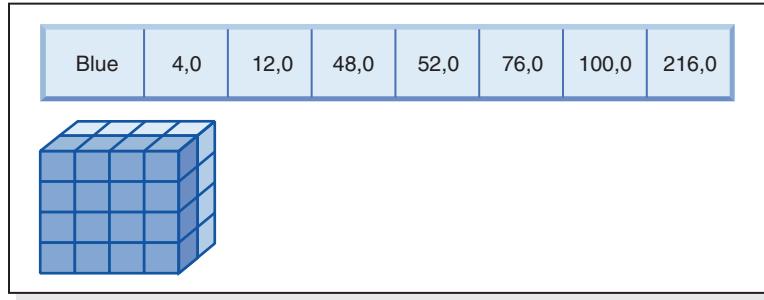
To carry out this query request, the following is done (and is shown in Figure 41):

- A dimension block index lookup and a RID index lookup are done.
- A logical AND operation is used with the blocks and RIDs to determine the intersection of the slice and those rows meeting the predicate condition.
- The result is only those RIDs that also belong to the qualifying blocks.

Example 9:

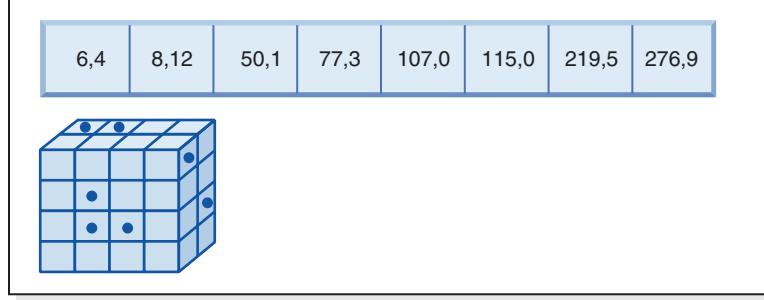
```
SELECT * FROM MDCTABLE WHERE COLOR='BLUE' OR PARTNO < 1000
```

### Key from the dimension block index on Colour



**+ (OR)**

### Row IDs (RID) from RID index on Partno



**=**

### Resulting blocks and RIDs to fetch

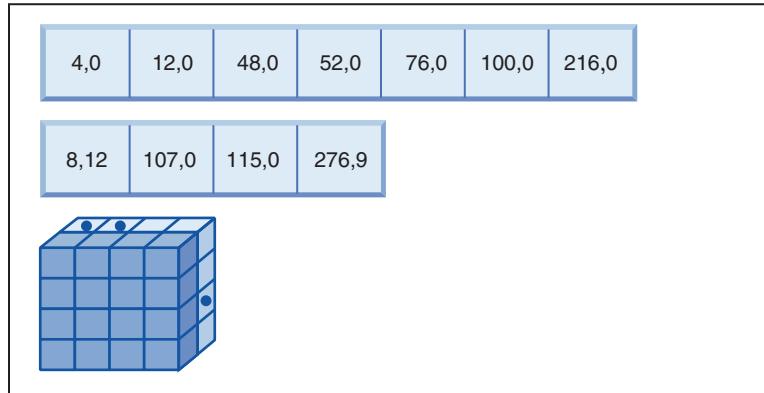


Figure 42. How block index and row ID using a logical OR operation works

To carry out this query request, the following is done (and is shown in Figure 42):

- A dimension block index lookup and a RID index lookup are done.
- A logical OR operation is used with the blocks and RIDs to determine the union of the slice and those rows meeting the predicate condition.
- The result is all of the rows in the qualifying blocks, plus additional RIDs that fall outside the qualifying blocks that meet the predicate condition. A mini-relational scan of each of the blocks is performed to retrieve their records, and the additional records outside these blocks are retrieved individually.

Example 10:

```
SELECT ... FROM t1 WHERE c1 < 5 OR c4 = 100
```

This query involves a range predicate on dimension c1, an equality predicate on a non-dimension column c4, and a logical OR operation. If there is a RID index on

the c4 column, this might be internally rewritten to do a logical OR operation using the dimension block index on c1 and the RID index on c4. If there is no index on c4, a table scan might be chosen instead, because all records must be checked. The logical OR operation uses a block index scan on c1 for values less than 4, and a RID index scan on c4 for values of 100. A mini-relational scan is performed on each block that qualifies, because all records within those blocks will qualify, and any additional RIDs for records outside of those blocks are retrieved as well.

Example 11:

```
SELECT FROM t1,d1,d2,d3
 WHERE t1.c1 = d1.c1 and d1.region = 'NY'
 AND t2.c2 = d2.c3 and d2.year='1994'
 AND t3.c3 = d3.c3 and d3.product='basketball'
```

This query involves a star join. In this example, t1 is the fact table and it has foreign keys c1, c2, and c3, corresponding to the primary keys of d1, d2, and d3, the dimension tables. The dimension tables do not need to be MDC tables. Region, year, and product are columns of the dimension tables that can be indexed using regular or block indexes (if the dimension tables are MDC tables). When accessing the fact table on c1, c2, and c3 values, block index scans of the dimension block indexes on these columns can be done, followed by a logical AND operation using the resulting BIDs. When there is a list of blocks, a mini-relational scan can be done on each block to get the records.

## Density of cells

The choices made for the appropriate dimensions and for the extent size are of **critical** importance to MDC design. These factors determine the table's expected cell density. They are important because an extent is allocated for every existing cell, regardless of the number of records in the cell. The right choices will take advantage of block-based indexing and multidimensional clustering, resulting in performance gains. The goal is to have densely-filled blocks to get the most benefit from multidimensional clustering, and to get optimal space utilization.

Thus, a very important consideration when designing a multidimensional table is the expected density of cells in the table, based on present and anticipated data. You can choose a set of dimensions, based on query performance, that cause the potential number of cells in the table to be very large, based on the number of possible values for each of the dimensions. The number of possible cells in the table is equal to the Cartesian product of the cardinalities of each of the dimensions. For example, if you cluster the table on dimensions Day, Region and Product and the data covers 5 years, you might have 1821 days \* 12 regions \* 5 products = 109 260 different possible cells in the table. Any cell that contains only a few records still requires an entire block of pages to store its records. If the block size is large, this table might end up being much larger than it needs to be.

There are several design factors that can contribute to optimal cell density:

- Varying the number of dimensions.
- Varying the granularity of one or more dimensions.
- Varying the block (extent) size and page size of the table space.

Carry out the following steps to achieve the best design possible:

1. Identify candidate dimensions.

Determine which queries will benefit from block-level clustering. Examine the potential workload for columns which have some or all of the following characteristics:

- Range and equality of any IN-list predicates
  - Roll-in or roll-out of data
  - Group-by and order-by clauses
  - Join clauses (especially in star schema environments).
2. Estimate the number of cells.
- Identify how many potential cells are possible in a table organized along a set of candidate dimensions. Determine the number of unique combinations of the dimension values that occur in the data. If the table exists, an exact number can be determined for the current data by selecting the number of distinct values in each of the columns that will be dimensions for the table. Alternatively, an approximation can be determined if you only have the statistics for a table, by multiplying the column cardinalities for the dimension candidates.
- Note:** If your table is in a partitioned database environment, and the distribution key is not related to any of the dimensions considered, determine an average amount of data per cell by taking all of the data and dividing by the number of database partitions.
3. Estimate the space occupancy or density.
- On average, consider that each cell has one partially-filled block where only a few rows are stored. There will be more partially-filled blocks as the number of rows per cell becomes smaller. Also, note that on average (assuming little or no data skew), the number of records per cell can be found by dividing the number of records in the table by the number of cells. However, if your table is in a partitioned database environment, consider how many records there are per cell on each database partition, because blocks are allocated for data on a database partition basis. When estimating the space occupancy and density in a partitioned database environment, consider the average number of records per cell on each database partition, not across the entire table.

There are several ways to improve the density:

- Reduce the block size so that partially-filled blocks take up less space.
- Reduce the size of each block by making the extent size appropriately small. Each cell that has a partially-filled block, or that contains only one block with few records on it, wastes less space. The trade-off, however, is that for those cells having many records, more blocks are needed to contain them. This increases the number of block identifiers (BIDs) for these cells in the block indexes, making these indexes larger and potentially resulting in more inserts and deletes to these indexes as blocks are more quickly emptied and filled. It also results in more small groupings of clustered data in the table for these more populated cell values, versus a smaller number of larger groupings of clustered data.

- Reduce the number of cells by reducing the number of dimensions, or by increasing the granularity of the cells with a generated column.

You can roll up one or more dimensions to a coarser granularity to give it a lower cardinality. For example, you can continue to cluster the data in the previous example on Region and Product, but replace the dimension of Day with a dimension of YearAndMonth. This gives cardinalities of 60 (12 months times 5 years), 12, and 5 for YearAndMonth, Region, and Product, with a possible number of cells of 3600. Each cell then holds a greater range of values and is less likely to contain only a few records.

Take into account predicates commonly used on the columns involved, such as whether many are on Month of Date, or Quarter, or Day. This affects the desirability of changing the granularity of the dimension. If, for example, most predicates are on particular days and you have clustered the table on Month, DB2 for Linux, UNIX, and Windows can use the block index on YearAndMonth to quickly narrow down which months contain the required days and access only those associated blocks. When scanning the blocks, however, the Day predicate must be applied to determine which days qualify. However, if you cluster on Day (and Day has high cardinality), the block index on Day can be used to determine which blocks to scan, and the Day predicate only has to be reapplied to the first record of each cell that qualifies. In this case, it might be better to consider rolling up one of the other dimensions to increase the density of cells, as in rolling up the Region column, which contains 12 different values, to Regions West, North, South and East, using a user-defined function.

---

## Scenario: MDC tables

As a scenario of how to work with an MDC table, we will imagine an MDC table called "Sales" that records sales data for a national retailer. The table is clustered along the dimensions "YearAndMonth" and "Region". Records in the table are stored in blocks, which contain enough consecutive pages on disk to fill an extent.

In Figure 43 on page 214, a block is represented by a rectangle, and is numbered according to the logical order of allocated extents in the table. The grid in the diagram represents the logical database partitioning of these blocks, and each square represents a logical cell. A column or row in the grid represents a slice for a particular dimension. For example, all records containing the value 'South-central' in the "Region" column are found in the blocks contained in the slice defined by the 'South-central' column in the grid. In fact, each block in this slice also only contains records having 'South-central' in the "Region" field. Thus, a block is contained in this slice or column of the grid if and only if it contains records having 'South-central' in the "Region" field.

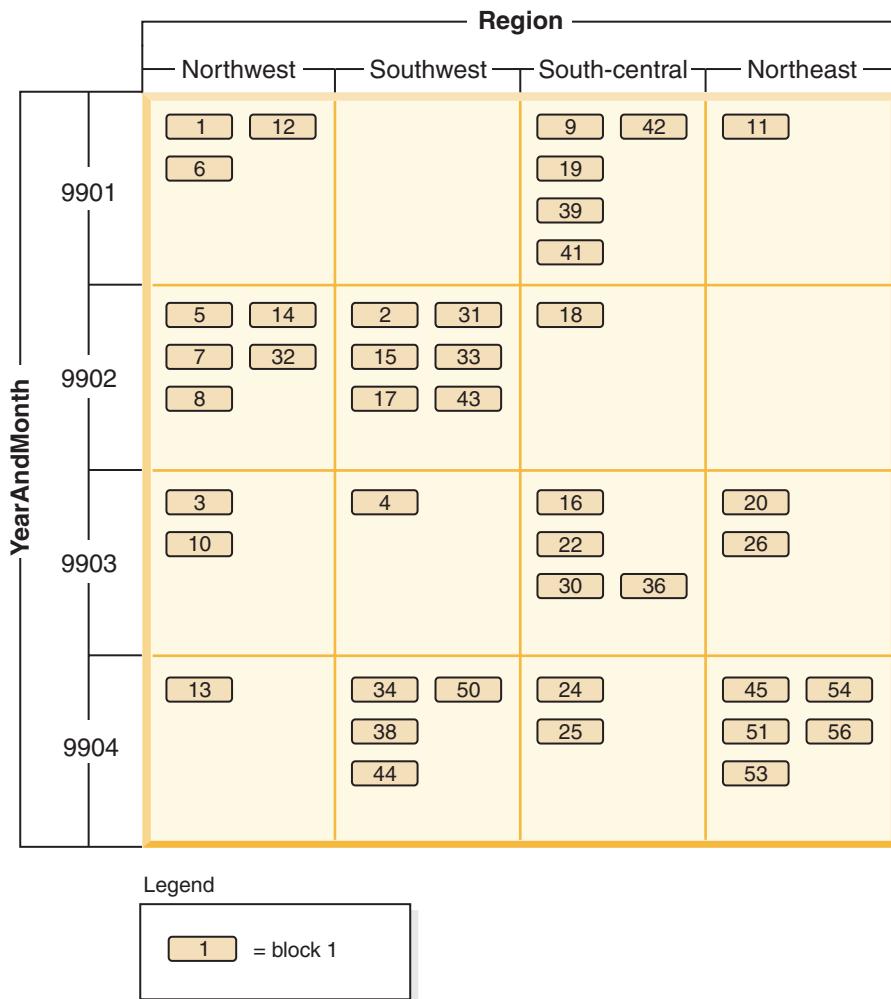


Figure 43. Multidimensional table with dimensions of 'Region' and 'YearAndMonth' that is called Sales

To determine which blocks comprise a slice, or equivalently, which blocks contain all records having a particular dimension key value, a dimension block index is automatically created for each dimension when the table is created.

In Figure 44 on page 215, a dimension block index is created on the "YearAndMonth" dimension, and another on the "Region" dimension. Each dimension block index is structured in the same manner as a traditional RID index, except that at the leaf level the keys point to a block identifier (BID) instead of a record identifier (RID). A RID identifies the location of a record in the table by a physical page number and a slot number - the slot on the page where the record is found. A BID represents a block by the physical page number of the first page of that extent, and a dummy slot (0). Because all pages in the block are physically consecutive starting from that one, and we know the size of the block, all records in the block can be found using this BID.

A slice, or the set of blocks containing pages with all records having a particular key value in a dimension, will be represented in the associated dimension block index by a BID list for that key value.

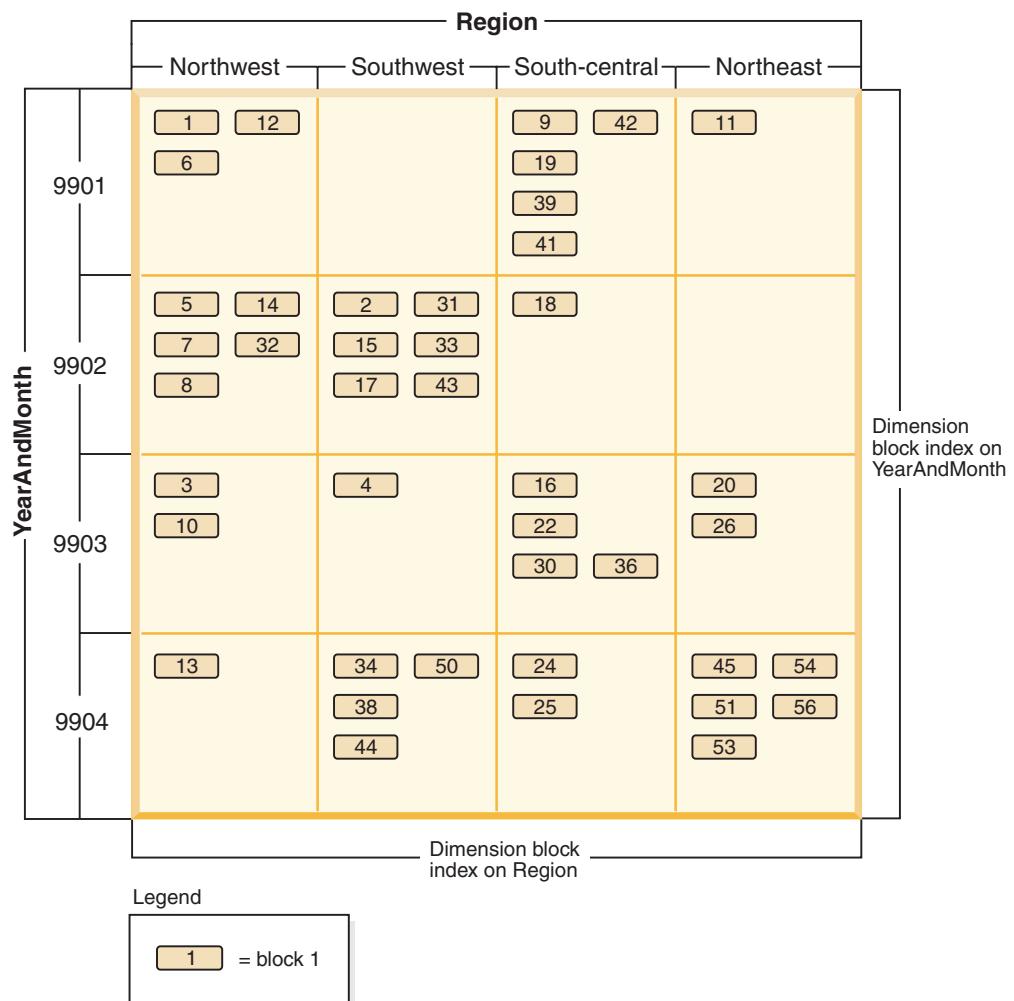


Figure 44. Sales table with dimensions of 'Region' and 'YearAndMonth' showing dimension block indexes

Figure 45 shows how a key from the dimension block index on "Region" would appear. The key is made up of a key value, namely 'South-central', and a list of BIDs. Each BID contains a block location. In Figure 45, the block numbers listed are the same that are found in the 'South-central' slice found in the grid for the Sales table (see Figure 43 on page 214).



Figure 45. Key from the dimension block index on 'Region'

Similarly, to find the list of blocks containing all records having '9902' for the "YearAndMonth" dimension, look up this value in the "YearAndMonth" dimension block index, shown in Figure 46 on page 216.

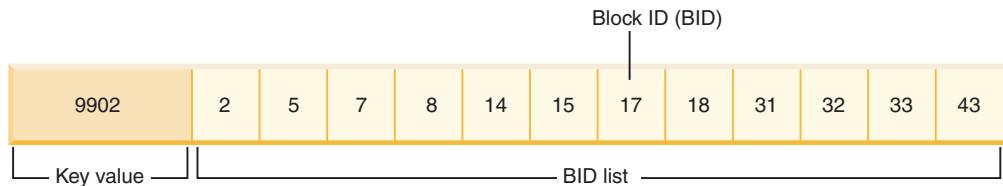


Figure 46. Key from the dimension block index on 'YearAndMonth'

## Scenario: Creating an ITC table

Insert time clustering (ITC) tables can help Olivia, and ExampleBANK, manage database size more effectively without manual intervention or database downtime.

Olivia creates an insert time clustering table as a test (the ORGANIZE BY INSERT TIME clause ensures that the table is created as an ITC table):

```
DB2 CREATE TABLE T1(c1 int, c2 char(100), ...) IN TABLESPACE1
 ORGANIZE BY INSERT TIME;
DB2 CREATE INDEX INX1 ON T1(C1);
```

## Scenario: Converting an existing table to an ITC table

Olivia sees the benefit of using insert time clustering tables. Olivia now wants to use this solution on existing tables in the production database. This change is accomplished by using the online table move utility.

Olivia has a table that exists on a system with the following schema. In this scenario, assume that the table actually has a column which is useful for placing data in approximate insert time order (C4).

```
CREATE TABLE EXMP.T1 (C1 INT, C2 CHAR(50), C3 BIGINT, C4 DATE) IN TABLESPACE1
CREATE INDEX INX1 ON EXMP.T1(C4)
```

Olivia now creates the target table for the conversion:

```
DB2 CREATE TABLE EXMP.NEWT1(C1 INT, C2 CHAR(50), C3 BIGINT, C4 DATE)
 IN TABLESPACE1 ORGANIZE BY INSERT TIME
```

The schema is identical to the original table but by using the ORGANIZE BY INSERT TIME keywords, Olivia ensures that this table is clustered by time.

Olivia uses the online table move stored procedure to perform the conversion.

Since a clustering index exists on column C4, it gives Olivia a good approximation of insert time ordering. For tables that do not have such a column, the space reclamation benefits of moving to an insert time clustering table is not apparent for some time. This benefit is not immediately apparent because newer data is grouped together with older data.

```
DB2 CALL SYSPROC.ADMIN_MOVE_TABLE('EXMP', 'T1', 'NEWT1', NULL, 'MOVE')
```

EXMP.T1 is now in a time clustering table format. It is ready to have extents reclaimed after subsequent batch deletions.

---

## Chapter 9. Partitioned tables

Partitioned tables use a data organization scheme in which table data is divided across multiple storage objects, called *data partitions* or *ranges*, according to values in one or more table partitioning key columns of the table.

A data partition or range is part of a table, containing a subset of rows of a table, and stored separately from other sets of rows. Data from a given table is partitioned into multiple data partitions or ranges based on the specifications provided in the PARTITION BY clause of the CREATE TABLE statement. These data partitions or ranges can be in different table spaces, in the same table space, or a combination of both. If a table is created using the PARTITION BY clause, the table is partitioned.

All of the table spaces specified must have the same page size, extent size, storage mechanism (DMS or SMS), and type (REGULAR or LARGE), and all of the table spaces must be in the same database partition group.

A partitioned table simplifies the rolling in and rolling out of table data and a partitioned table can contain vastly more data than an ordinary table. You can create a partitioned table with a maximum of 32,767 data partitions. Data partitions can be added to, attached to, and detached from a partitioned table, and you can store multiple data partition ranges from a table in one table space.

Indexes on a partitioned table can be partitioned or nonpartitioned. Both nonpartitioned and partitioned indexes can exist together on a single partitioned table.

### Restrictions

Partitioned hierarchical or temporary tables, range-clustered tables, and partitioned views are not supported for use in partitioned tables.

---

## Table partitioning keys

A table partitioning key is an ordered set of one or more columns in a table. The values in the table partitioning key columns are used to determine in which data partition each table row belongs.

To define the table partitioning key on a table use the CREATE TABLE statement with the PARTITION BY clause.

Choosing an effective table partitioning key column is essential to taking full advantage of the benefits of table partitioning. The following guidelines can help you to choose the most effective table partitioning key columns for your partitioned table.

- Define range granularity to match data roll-out. It is most common to use week, month, or quarter.
- Define ranges to match the data roll-in size. It is most common to partition data on a date or time column.
- Partition on a column that provides advantages in partition elimination.

## Supported data types

Table 19 shows the data types (including synonyms) that are supported for use as a table partitioning key column:

*Table 19. Supported data types*

| Data type column 1        | Data type column 2             |
|---------------------------|--------------------------------|
| SMALLINT                  | INTEGER                        |
| INT                       | BIGINT                         |
| FLOAT                     | REAL                           |
| DOUBLE                    | DECIMAL                        |
| DEC                       | DECFLOAT                       |
| NUMERIC                   | NUM                            |
| CHARACTER                 | CHAR                           |
| VARCHAR                   | DATE                           |
| TIME                      | GRAPHIC                        |
| VARGRAPHIC                | CHARACTER VARYING              |
| TIMESTAMP                 | CHAR VARYING                   |
| CHARACTER FOR BIT DATA    | CHAR FOR BIT DATA              |
| VARCHAR FOR BIT DATA      | CHARACTER VARYING FOR BIT DATA |
| CHAR VARYING FOR BIT DATA | User defined types (distinct)  |

## Unsupported data types

The following data types can occur in a partitioned table, but are not supported for use as a table partitioning key column:

- User defined types (structured)
- LONG VARCHAR
- LONG VARCHAR FOR BIT DATA
- BLOB
- BINARY LARGE OBJECT
- CLOB
- CHARACTER LARGE OBJECT
- DBCLOB
- LONG VARGRAPHIC
- REF
- Varying length string for C
- Varying length string for Pascal
- XML

If you choose to automatically generate data partitions using the EVERY clause of the CREATE TABLE statement, only one column can be used as the table partitioning key. If you choose to manually generate data partitions by specifying each range in the PARTITION BY clause of the CREATE TABLE statement, multiple columns can be used as the table partitioning key, as shown in the following example:

```

CREATE TABLE sales (year INT, month INT)
 PARTITION BY RANGE(year, month)
 (STARTING FROM (2001, 1) ENDING (2001,3) IN tbsp1,
 ENDING (2001,6) IN tbsp2, ENDING (2001,9)
 IN tbsp3, ENDING (2001,12) IN tbsp4,
 ENDING (2002,3) IN tbsp5, ENDING (2002,6)
 IN tbsp6, ENDING (2002,9) IN tbsp7,
 ENDING (2002,12) IN tbsp8)

```

This results in eight data partitions, one for each quarter in year 2001 and 2002.

**Note:**

1. When multiple columns are used as the table partitioning key, they are treated as a composite key (which are similar to composite keys in an index), in the sense that trailing columns are dependent on the leading columns. Each starting or ending value (all of the columns, together) must be specified in 512 characters or less. This limit corresponds to the size of the LOWVALUE and HIGHVALUE columns of the SYSCAT.DATAPARTITIONS catalog view. A starting or ending value specified with more than 512 characters will result in error SQL0636N, reason code 9.
2. Table partitioning is multicolumn not multidimension. In table partitioning, all columns used are part of a single dimension.

## Generated columns

Generated columns can be used as table partitioning keys. This example creates a table with twelve data partitions, one for each month. All rows for January of any year will be placed in the first data partition, rows for February in the second, and so on.

*Example 1*

```

CREATE TABLE monthly_sales (sales_date date,
sales_month int GENERATED ALWAYS AS (month(sales_date)))
 PARTITION BY RANGE (sales_month)
 (STARTING FROM 1 ENDING AT 12 EVERY 1);

```

**Note:**

1. You cannot alter or drop the expression of a generated column that is used in the table partitioning key. Adding a generated column expression on a column that is used in the table partitioning key is not permitted. Attempting to add, drop or alter a generated column expression for a column used in the table partitioning key results in error (SQL0270N rc=52).
2. Data partition elimination will not be used for range predicates if the generated column is not monotonic, or the optimizer can not detect that it is monotonic. In the presence of non-monotonic expressions, data partition elimination can only take place for equality or IN predicates. For a detailed discussion and examples of monotonicity see “Creating MDC or ITC tables” on page 198.

## Table partitioning and multidimensional clustering tables

In a table that is both multidimensional clustered and data partitioned, columns can be used both in the table partitioning range-partition-spec and in the multidimensional clustering (MDC) key. A table that is both multidimensional clustered and partitioned can achieve a finer granularity of data partition and block elimination than could be achieved by either functionality alone.

There are also many applications where it is useful to specify different columns for the MDC key than those on which the table is partitioned. It should be noted that table partitioning is multicolumn, while MDC is multi-dimension.

## Characteristics of a mainstream DB2 data warehouse

The following recommendations were focused on typical, mainstream warehouses that were new for DB2 V9.1. The following characteristics are assumed:

- The database runs on multiple machines or multiple AIX logical partitions.
- Partitioned database environments are used (tables are created using the `DISTRIBUTE BY HASH` clause).
- There are four to fifty data partitions.
- The table for which MDC and table partitioning is being considered is a major fact table.
- The table has 100,000,000 to 100,000,000,000 rows.
- New data is loaded at various time frames: nightly, weekly, monthly.
- Daily ingest volume is 10 thousand to 10 million records.
- Data volumes vary: The biggest month is 5X the size of the smallest month. Likewise, the biggest dimensions (product line, region) have a 5X size range.
- 1 to 5 years of detailed data is retained.
- Expired data is rolled out monthly or quarterly.
- Tables use a wide range of query types. However, the workload is mostly analytical queries with the following characteristics, relative to OLTP workloads:
  - larger results sets with up to 2 million rows
  - most or all queries are hitting views, not base tables
- SQL clauses selecting data by ranges (`BETWEEN` clause), items in lists, and so on.

## Characteristics of a mainstream DB2 V9.1 data warehouse fact table

A typical warehouse fact table, might use the following design:

- Create data partitions on the Month column.
- Define a data partition for each period you roll-out, for example, 1 month, 3 months.
- Create MDC dimensions on Day and on 1 to 4 additional dimensions. Typical dimensions are: product line and region.
- All data partitions and MDC clusters are spread across all database partitions.

MDC and table partitioning provide overlapping sets of benefits. The following table lists potential needs in your organization and identifies a recommended organization scheme based on the characteristics identified previously.

*Table 20. Using table partitioning with MDC tables*

| Issue                             | Recommended scheme | Recommendation                                                                                          |
|-----------------------------------|--------------------|---------------------------------------------------------------------------------------------------------|
| Data availability during roll-out | Table partitioning | Use the <code>DETACH PARTITION</code> clause to roll out large amounts of data with minimal disruption. |

*Table 20. Using table partitioning with MDC tables (continued)*

| Issue                                                                                                                                      | Recommended scheme         | Recommendation                                                                                                                                                                                                                                                                        |
|--------------------------------------------------------------------------------------------------------------------------------------------|----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Query performance                                                                                                                          | Table partitioning and MDC | MDC is best for querying multiple dimensions. Table partitioning helps through data partition elimination.                                                                                                                                                                            |
| Minimal reorganization                                                                                                                     | MDC                        | MDC maintains clustering, which reduces the need to reorganize.                                                                                                                                                                                                                       |
| Rollout a month or more of data during a traditional offline window                                                                        | Table partitioning         | Data partitioning addresses this need fully. MDC adds nothing and would be less suitable on its own.                                                                                                                                                                                  |
| Rollout a month or more of data during a micro-offline window (less than 1 minute)                                                         | Table partitioning         | Data partitioning addresses this need fully. MDC adds nothing and would be less suitable on its own.                                                                                                                                                                                  |
| Rollout a month or more of data while keeping the table fully available for business users submitting queries without any loss of service. | MDC                        | MDC only addresses this need somewhat. Table partitioning would not be suitable due to the short period the table goes offline.                                                                                                                                                       |
| Load data daily ( <b>LOAD</b> or <b>INGEST</b> command)                                                                                    | Table partitioning and MDC | MDC provides most of the benefit here. Table partitioning provides incremental benefits.                                                                                                                                                                                              |
| Load data "continually" ( <b>LOAD</b> command with ALLOW READ ACCESS or <b>INGEST</b> command)                                             | Table partitioning and MDC | MDC provides most of the benefit here. Table partitioning provides incremental benefits.                                                                                                                                                                                              |
| Query execution performance for "traditional BI" queries                                                                                   | Table partitioning and MDC | MDC is especially good for querying cubes/multiple dimensions. Table partitioning helps via partition elimination.                                                                                                                                                                    |
| Minimize reorganization pain, by avoiding the need for reorganization or reducing the pain associated with performing the task             | MDC                        | MDC maintains clustering which reduces the need to reorg. If MDC is used, data partitioning does not provide incremental benefits. However if MDC is not used, table partitioning helps reduce the need for reorg by maintaining some coarse grain clustering at the partition level. |

*Example 1:*

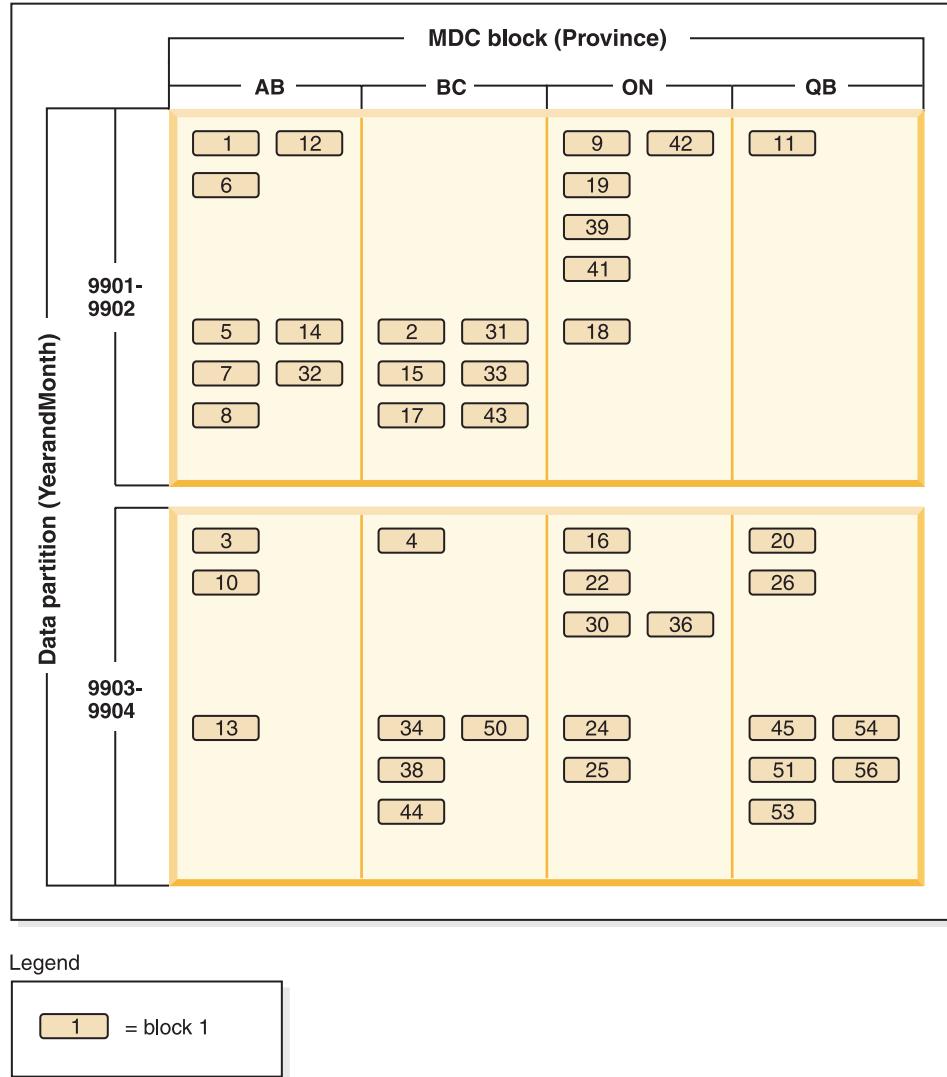
Consider a table with key columns YearAndMonth and Province. A reasonable approach to planning this table might be to partition by date with 2 months per data partition. In addition, you might also organize by Province, so that all rows for a particular province within any two month date range are clustered together, as shown in Figure 47 on page 222.

```

CREATE TABLE orders (YearAndMonth INT, Province CHAR(2))
PARTITION BY RANGE (YearAndMonth)
(STARTING 9901 ENDING 9904 EVERY 2)
ORGANIZE BY (Province);

```

**Table orders**



*Figure 47. A table partitioned by YearAndMonth and organized by Province*

*Example 2:*

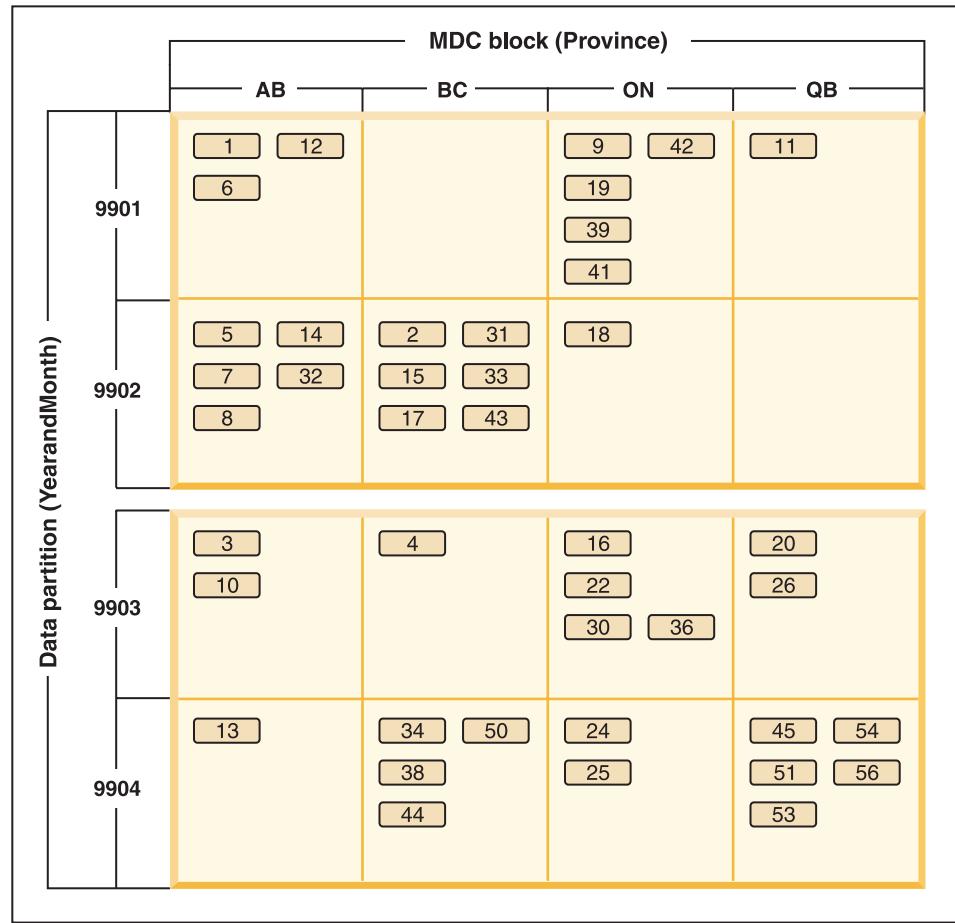
Finer granularity can be achieved by adding YearAndMonth to the ORGANIZE BY DIMENSIONS clause, as shown in Figure 48 on page 223.

```

CREATE TABLE orders (YearAndMonth INT, Province CHAR(2))
PARTITION BY RANGE (YearAndMonth)
(STARTING 9901 ENDING 9904 EVERY 2)
ORGANIZE BY (YearAndMonth, Province);

```

**Table orders**



*Figure 48. A table partitioned by YearAndMonth and organized by Province and YearAndMonth*

In cases where the partitioning is such that there is only a single value in each range, nothing is gained by including the table partitioning column in the MDC key.

## Considerations

- Compared to a basic table, both MDC tables and partitioned tables require more storage. These storage needs are additive but are considered reasonable given the benefits.
- If you choose not to combine table partitioning and MDC functionality in your partitioned database environment, table partitioning is best in cases where you can confidently predict the data distribution, which is generally the case for the types of systems discussed here. Otherwise, MDC should be considered.
- For a data-partitioned MDC table created with DB2 Version 9.7 Fix Pack 1 or later releases, the MDC block indexes on the table are partitioned. For a data-partitioned MDC table created with DB2 V9.7 or earlier releases, the MDC block indexes on the table are nonpartitioned.

---

## Optimization strategies for partitioned tables

Data partition elimination refers to the database server's ability to determine, based on query predicates, that only a subset of the data partitions in a table need to be accessed to answer a query. Data partition elimination is particularly useful when running decision support queries against a partitioned table.

A partitioned table uses a data organization scheme in which table data is divided across multiple storage objects, called data partitions or ranges, according to values in one or more table partitioning key columns of the table. Data from a table is partitioned into multiple storage objects based on specifications provided in the PARTITION BY clause of the CREATE TABLE statement. These storage objects can be in different table spaces, in the same table space, or a combination of both.

The following example demonstrates the performance benefits of data partition elimination.

```
create table custlist(
 subsdate date, province char(2), accountid int)
partition by range(subsdate) (
 starting from '1/1/1990' in ts1,
 starting from '1/1/1991' in ts1,
 starting from '1/1/1992' in ts1,
 starting from '1/1/1993' in ts2,
 starting from '1/1/1994' in ts2,
 starting from '1/1/1995' in ts2,
 starting from '1/1/1996' in ts3,
 starting from '1/1/1997' in ts3,
 starting from '1/1/1998' in ts3,
 starting from '1/1/1999' in ts4,
 starting from '1/1/2000' in ts4,
 starting from '1/1/2001'
 ending '12/31/2001' in ts4)
```

Assume that you are only interested in customer information for the year 2000.

```
select * from custlist
 where subsdate between '1/1/2000' and '12/31/2000'
```

As Figure 49 on page 225 shows, the database server determines that only one data partition in table space TS4 must be accessed to resolve this query.

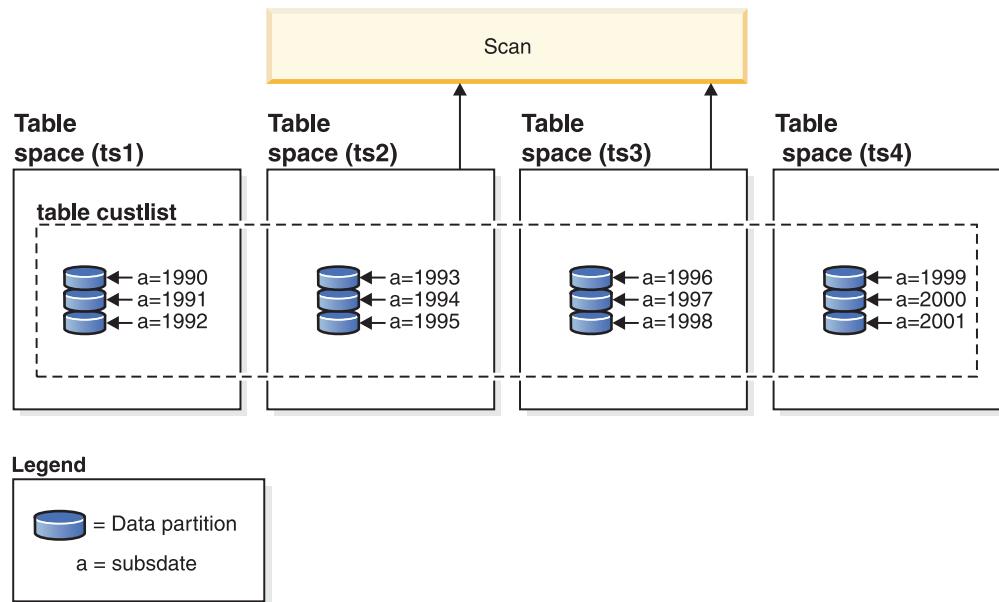


Figure 49. The performance benefits of data partition elimination

Another example of data partition elimination is based on the following scheme:

```

create table multi (
 sale_date date, region char(2))
partition by (sale_date) (
 starting '01/01/2005'
 ending '12/31/2005'
 every 1 month)

create index sx on multi(sale_date)

create index rx on multi(region)

```

Assume that you issue the following query:

```

select * from multi
where sale_date between '6/1/2005'
and '7/31/2005' and region = 'NW'

```

Without table partitioning, one likely plan is index ANDing. Index ANDing performs the following tasks:

- Reads all relevant index entries from each index
- Saves both sets of row identifiers (RIDs)
- Matches RIDs to determine which occur in both indexes
- Uses the RIDs to fetch the rows

As Figure 50 on page 226 demonstrates, with table partitioning, the index is read to find matches for both REGION and SALE\_DATE, resulting in the fast retrieval of matching rows.

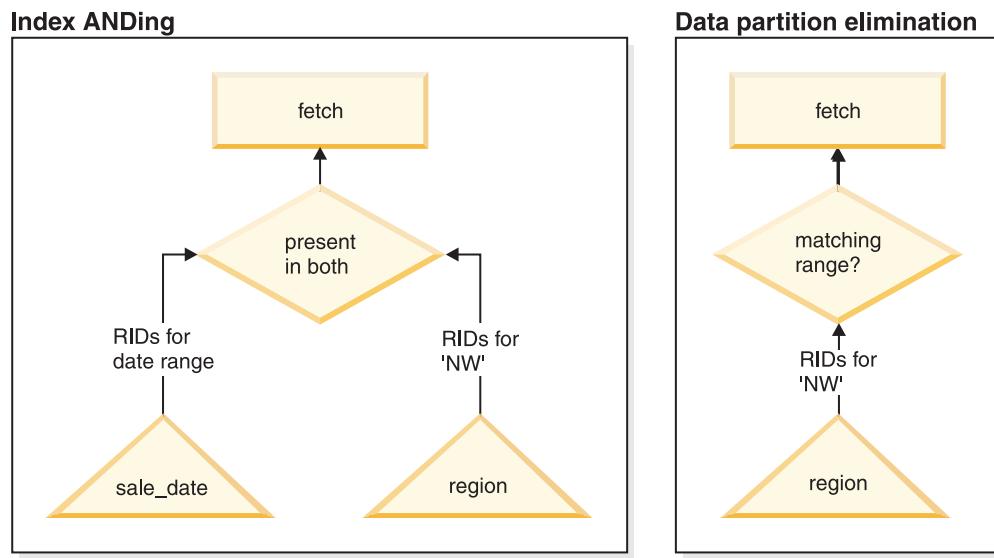


Figure 50. Optimizer decision path for both table partitioning and index ANDing

## DB2 Explain

You can also use the explain facility to determine the data partition elimination plan that was chosen by the query optimizer. The “DP Elim Predicates” information shows which data partitions are scanned to resolve the following query:

```
select * from custlist
 where subsdate between '12/31/1999' and '1/1/2001'
```

Arguments:

```

DPESTFLG: (Number of data partitions accessed are Estimated)
 FALSE
DPLSTPRT: (List of data partitions accessed)
 9-11
DPNUMPRT: (Number of data partitions accessed)
 3
```

DP Elim Predicates:

```

Range 1)
Stop Predicate: (Q1.A <= '01/01/2001')
Start Predicate: ('12/31/1999' <= Q1.A)
```

Objects Used in Access Plan:

```

Schema: MRSRINI
Name: CUSTLIST
Type: Data Partitioned Table
Time of creation: 2005-11-30-14.21.33.857039
Last statistics update: 2005-11-30-14.21.34.339392
Number of columns: 3
Number of rows: 100000
Width of rows: 19
Number of buffer pool pages: 1200
Number of data partitions: 12
Distinct row values: No
Tablespace name: <VARIOUS>
```

## Multi-column support

Data partition elimination works in cases where multiple columns are used as the table partitioning key. For example:

```
create table sales (
 year int, month int)
partition by range(year, month) (
 starting from (2001,1)
 ending at (2001,3) in ts1,
 ending at (2001,6) in ts2,
 ending at (2001,9) in ts3,
 ending at (2001,12) in ts4,
 ending at (2002,3) in ts5,
 ending at (2002,6) in ts6,
 ending at (2002,9) in ts7,
 ending at (2002,12) in ts8)

select * from sales where year = 2001 and month < 8
```

The query optimizer deduces that only data partitions in TS1, TS2, and TS3 must be accessed to resolve this query.

**Note:** In the case where multiple columns make up the table partitioning key, data partition elimination is only possible when you have predicates on the leading columns of the composite key, because the non-leading columns that are used for the table partitioning key are not independent.

## Multi-range support

It is possible to obtain data partition elimination with data partitions that have multiple ranges (that is, those that are ORed together). Using the SALES table that was created in the previous example, execute the following query:

```
select * from sales
where (year = 2001 and month <= 3)
or (year = 2002 and month >= 10)
```

The database server only accesses data for the first quarter of 2001 and the last quarter of 2002.

## Generated columns

You can use generated columns as table partitioning keys. For example:

```
create table sales (
 a int, b int generated always as (a / 5))
 in ts1,ts2,ts3,ts4,ts5,ts6,ts7,ts8,ts9,ts10
partition by range(b) (
 starting from (0)
 ending at (1000) every (50))
```

In this case, predicates on the generated column are used for data partition elimination. In addition, when the expression that is used to generate the columns is monotonic, the database server translates predicates on the source columns into predicates on the generated columns, which enables data partition elimination on the generated columns. For example:

```
select * from sales where a > 35
```

The database server generates an extra predicate on b ( $b > 7$ ) from a ( $a > 35$ ), thus allowing data partition elimination.

## Join predicates

Join predicates can also be used in data partition elimination, if the join predicate is pushed down to the table access level. The join predicate is only pushed down to the table access level on the inner join of a nested loop join (NLJN).

Consider the following tables:

```
create table t1 (a int, b int)
partition by range(a,b) (
 starting from (1,1)
 ending (1,10) in ts1,
 ending (1,20) in ts2,
 ending (2,10) in ts3,
 ending (2,20) in ts4,
 ending (3,10) in ts5,
 ending (3,20) in ts6,
 ending (4,10) in ts7,
 ending (4,20) in ts8)

create table t2 (a int, b int)
```

The following two predicates will be used:

P1: T1.A = T2.A  
P2: T1.B > 15

In this example, the exact data partitions that will be accessed at compile time cannot be determined, due to unknown outer values of the join. In this case, as well as cases where host variables or parameter markers are used, data partition elimination occurs at run time when the necessary values are bound.

During run time, when T1 is the inner of an NLJN, data partition elimination occurs dynamically, based on the predicates, for every outer value of T2.A. During run time, the predicates T1.A = 3 and T1.B > 15 are applied for the outer value T2.A = 3, which qualifies the data partitions in table space TS6 to be accessed.

Suppose that column A in tables T1 and T2 have the following values:

| Outer table T2:<br>column A | Inner table T1:<br>column A | Inner table T1:<br>column B | Inner table T1: data<br>partition location |
|-----------------------------|-----------------------------|-----------------------------|--------------------------------------------|
| 2                           | 3                           | 20                          | TS6                                        |
| 3                           | 2                           | 10                          | TS3                                        |
| 3                           | 2                           | 18                          | TS4                                        |
|                             | 3                           | 15                          | TS6                                        |
|                             | 1                           | 40                          | TS3                                        |

To perform a nested loop join (assuming a table scan for the inner table), the database manager performs the following steps:

1. Reads the first row from T2. The value for A is 2.
2. Binds the T2.A value (which is 2) to the column T2.A in the join predicate T1.A = T2.A. The predicate becomes T1.A = 2.
3. Applies data partition elimination using the predicates T1.A = 2 and T1.B > 15. This qualifies data partitions in table space TS4.
4. After applying T1.A = 2 and T1.B > 15, scans the data partitions in table space TS4 of table T1 until a row is found. The first qualifying row found is row 3 of T1.
5. Joins the matching row.

6. Scans the data partitions in table space TS4 of table T1 until the next match (T1.A = 2 and T1.B > 15) is found. No more rows are found.
7. Repeats steps 1 through 6 for the next row of T2 (replacing the value of A with 3) until all the rows of T2 have been processed.

## Indexes over XML data

Starting in DB2 Version 9.7 Fix Pack 1, you can create an index over XML data on a partitioned table as either partitioned or nonpartitioned. The default is a partitioned index.

Partitioned and nonpartitioned XML indexes are maintained by the database manager during table insert, update, and delete operations in the same way as any other relational indexes on a partitioned table are maintained. Nonpartitioned indexes over XML data on a partitioned table are used in the same way as indexes over XML data on a nonpartitioned table to speed up query processing. Using the query predicate, it is possible to determine that only a subset of the data partitions in the partitioned table need to be accessed to answer the query.

Data partition elimination and indexes over XML columns can work together to enhance query performance. Consider the following partitioned table:

```
create table employee (a int, b xml, c xml)
 index in tbspx
 partition by (a) (
 starting 0 ending 10,
 ending 20,
 ending 30,
 ending 40)
```

Now consider the following query:

```
select * from employee
 where a > 21
 and xmlexist('$doc/Person/Name/First[.= "Eric"]'
 passing "EMPLOYEE"."B" as "doc")
```

The optimizer can immediately eliminate the first two partitions based on the predicate `a > 21`. If the nonpartitioned index over XML data on column B is chosen by the optimizer in the query plan, an index scan using the index over XML data will be able to take advantage of the data partition elimination result from the optimizer and only return results belonging to partitions that were not eliminated by the relational data partition elimination predicates.

---

## Partitioned materialized query table (MQT) behavior

All types of materialized query tables (MQTs) are supported with partitioned tables. When working with partitioned MQTs, there are a number of guidelines that can help you to administer attached and detached data partitions most effectively.

The following guidelines and restrictions apply when working with partitioned MQTs or partitioned tables with detached dependent tables:

- If you issue an `ALTER TABLE ... DETACH PARTITION` statement, the `DETACH` operation creates the target table for the detached partition data. If there are any dependent tables that need to be incrementally maintained with respect to the detached data partition (these dependent tables are referred to as detached dependent tables), the `SET INTEGRITY` statement is required to be run on the detached dependent tables to incrementally maintain the tables. With DB2 V9.7

Fix Pack 1 or later releases, after the SET INTEGRITY statement is run on all detached dependent tables, the asynchronous partition detach task makes the data partition into a stand-alone target table. Until the asynchronous partition detach operation completes, the target table is unavailable. The target table will be marked 'L' in the TYPE column of the SYSCAT.TABLES catalog view. This is referred to as a detached table. This prevents the target table from being read, modified or dropped until the SET INTEGRITY statement is run to incrementally maintain the detached dependent tables. After the SET INTEGRITY statement is run on all detached dependent tables, the data partition is logically detached from the source table and the asynchronous partition detach operation detaches data partition from the source table into the target table. Until the asynchronous partition detach operation completes, the target table is unavailable.

- To detect that a detached table is not yet accessible, query the SYSCAT.TABDETACHEDDEP catalog view. If any inaccessible detached tables are detected, run the SET INTEGRITY statement with the IMMEDIATE CHECKED option on all the detached dependent tables to transition the detached table to a regular accessible table. If you try to access a detached table before all its detached dependents are maintained, error code SQL20285N is returned.
- The DATAPARTITIONNUM function cannot be used in an materialized query table (MQT) definition. Attempting to create an MQT using this function returns an error (SQLCODE SQL20058N, SQLSTATE 428EC).
- When creating a nonpartitioned index on a table with detached data partitions with STATUS 'D' in SYSCAT.DATAPARTITIONS, the index does not include the data in the detached data partitions unless the detached data partition has a dependent materialized query table (MQT) that needs to be incrementally refreshed with respect to it. In this case, the index includes the data for this detached data partition.
- Altering a table with attached data partitions to an MQT is not allowed.
- Partitioned staging tables are not supported.
- Attaching to an MQT is not directly supported. See Example 1 for details.

#### **Example 1: Converting a partitioned MQT to an ordinary table**

Although the ATTACH operation is not directly supported on partitioned MQTs, you can achieve the same effect by converting a partitioned MQT to an ordinary table, performing the desired roll-in and roll-out of table data, and then converting the table back into an MQT. The following CREATE TABLE and ALTER TABLE statements demonstrate the effect:

```

CREATE TABLE lineitem (
 l_orderkey DECIMAL(10,0) NOT NULL,
 l_quantity DECIMAL(12,2),
 l_shipdate DATE,
 l_year_month INT GENERATED ALWAYS AS (YEAR(l_shipdate)*100 + MONTH(l_shipdate)))
 PARTITION BY RANGE(l_shipdate)
 (STARTING ('1/1/1992') ENDING ('12/31/1993') EVERY 1 MONTH);
CREATE TABLE lineitem_ex (
 l_orderkey DECIMAL(10,0) NOT NULL,
 l_quantity DECIMAL(12,2),
 l_shipdate DATE,
 l_year_month INT,
 ts TIMESTAMP,
 msg CLOB(32K));

CREATE TABLE quan_by_month (
 q_year_month, q_count) AS
 (SELECT l_year_month AS q_year_month, COUNT(*) AS q_count

```

```

 FROM lineitem
 GROUP BY l_year_month)
 DATA INITIALLY DEFERRED REFRESH IMMEDIATE
 PARTITION BY RANGE(q_year_month)
 (STARTING (199201) ENDING (199212) EVERY (1),
 STARTING (199301) ENDING (199312) EVERY (1));
CREATE TABLE quan_by_month_ex(
 q_year_month INT,
 q_count INT NOT NULL,
 ts TIMESTAMP,
 msg CLOB(32K));

SET INTEGRITY FOR quan_by_month IMMEDIATE CHECKED;
CREATE INDEX qbmx ON quan_by_month(q_year_month);

ALTER TABLE quan_by_month DROP MATERIALIZED QUERY;
ALTER TABLE lineitem DETACH PARTITION part0 INTO li_reuse;
ALTER TABLE quan_by_month DETACH PARTITION part0 INTO qm_reuse;

SET INTEGRITY FOR li_reuse OFF;
ALTER TABLE li_reuse ALTER l_year_month SET GENERATED ALWAYS
AS (YEAR(l_shipdate)*100 + MONTH(l_shipdate));

LOAD FROM part_mqt_rotate.del OF DEL MODIFIED BY GENERATEDIGNORE
MESSAGES load.msg REPLACE INTO li_reuse;

DECLARE load_cursor CURSOR FOR
 SELECT l_year_month, COUNT(*)
 FROM li_reuse
 GROUP BY l_year_month;
LOAD FROM load_cursor OF CURSOR MESSAGES load.msg
 REPLACE INTO qm_reuse;

ALTER TABLE lineitem ATTACH PARTITION STARTING '1/1/1994'
 ENDING '1/31/1994' FROM li_reuse;

SET INTEGRITY FOR lineitem ALLOW WRITE ACCESS IMMEDIATE CHECKED
FOR EXCEPTION IN lineitem USE lineitem_ex;

ALTER TABLE quan_by_month ATTACH PARTITION STARTING 199401
 ENDING 199401 FROM qm_reuse;

SET INTEGRITY FOR quan_by_month IMMEDIATE CHECKED
FOR EXCEPTION IN quan_by_month USE quan_by_month_ex;

ALTER TABLE quan_by_month ADD MATERIALIZED QUERY
 (SELECT l_year_month AS q_year_month, COUNT(*) AS q_count
 FROM lineitem
 GROUP BY l_year_month)
 DATA INITIALLY DEFERRED REFRESH IMMEDIATE;

SET INTEGRITY FOR QUAN_BY_MONTH ALL IMMEDIATE UNCHECKED;

```

Use the SET INTEGRITY statement with the IMMEDIATE CHECKED option to check the attached data partition for integrity violations. This step is required before changing the table back to an MQT. The SET INTEGRITY statement with the IMMEDIATE UNCHECKED option is used to bypass the required full refresh of the MQT. The index on the MQT is necessary to achieve optimal performance. The use of exception tables with the SET INTEGRITY statement is recommended, where appropriate.

Typically, you create a partitioned MQT on a large fact table that is also partitioned. If you do roll out or roll in table data on the large fact table, you must adjust the partitioned MQT manually, as demonstrated in Example 2.

### Example 2: Adjusting a partitioned MQT manually

Alter the MQT (quan\_by\_month) to convert it to an ordinary partitioned table:

```
ALTER TABLE quan_by_month DROP MATERIALIZED QUERY;
```

Detach the data to be rolled out from the fact table (lineitem) and the MQT and re-load the staging table li\_reuse with the new data to be rolled in:

```
ALTER TABLE lineitem DETACH PARTITION part0 INTO li_reuse;
LOAD FROM part_mqt_rotate.del OF DEL MESSAGES load.msg REPLACE INTO li_reuse;
ALTER TABLE quan_by_month DETACH PARTITION part0 INTO qm_reuse;
```

Prune qm\_reuse before doing the insert. This deletes the detached data before inserting the subselect data. This is accomplished with a load replace into the MQT where the data file of the load is the content of the subselect.

```
db2 load from datafile.del of del replace into qm_reuse
```

You can refresh the table manually using INSERT INTO ... (SELECT ...) This is only necessary on the new data, so the statement should be issued before attaching:

```
INSERT INTO qm_reuse
 (SELECT COUNT(*) AS q_count, l_year_month AS q_year_month
 FROM li_reuse
 GROUP BY l_year_month);
```

Now you can roll in the new data for the fact table:

```
ALTER TABLE lineitem ATTACH PARTITION STARTING '1/1/1994'
ENDING '1/31/1994' FROM TABLE li_reuse;
SET INTEGRITY FOR lineitem ALLOW WRITE ACCESS IMMEDIATE CHECKED FOR
EXCEPTION IN li_reuse USE li_reuse_ex;
```

Next, roll in the data for the MQT:

```
ALTER TABLE quan_by_month ATTACH PARTITION STARTING 199401
ENDING 199401 FROM TABLE qm_reuse;
SET INTEGRITY FOR quan_by_month IMMEDIATE CHECKED;
```

After attaching the data partition, the new data must be verified to ensure that it is in range.

```
ALTER TABLE quan_by_month ADD MATERIALIZED QUERY
 (SELECT COUNT(*) AS q_count, l_year_month AS q_year_month
 FROM lineitem
 GROUP BY l_year_month)
 DATA INITIALLY DEFERRED REFRESH IMMEDIATE;
SET INTEGRITY FOR QUAN_BY_MONTH ALL IMMEDIATE UNCHECKED;
```

The data is not accessible until it has been validated by the SET INTEGRITY statement. Although the REFRESH TABLE operation is supported, this scenario demonstrates the manual maintenance of a partitioned MQT through the ATTACH PARTITION and DETACH PARTITION operations. The data is marked as validated by the user through the IMMEDIATE UNCHECKED clause of the SET INTEGRITY statement.

---

## Large object behavior in partitioned tables

A partitioned table uses a data organization scheme in which table data is divided across multiple storage objects, called data partitions or ranges, according to values in one or more table partitioning key columns of the table. Data from a given table is partitioned into multiple storage objects based on the specifications provided in

the PARTITION BY clause of the CREATE TABLE statement. These storage objects can be in different table spaces, in the same table space, or a combination of both.

A large object for a partitioned table is, by default, stored in the same table space as its corresponding data object. This applies to partitioned tables that use only one table space or use multiple table spaces. When a partitioned table's data is stored in multiple table spaces, the large object data is also stored in multiple table spaces.

Use the LONG IN clause of the CREATE TABLE statement to override this default behavior. You can specify a list of table spaces for the table where long data is to be stored. If you choose to override the default behavior, the table space specified in the LONG IN clause must be a large table space. If you specify that long data be stored in a separate table space for one or more data partitions, you must do so for all the data partitions of the table. That is, you cannot have long data stored remotely for some data partitions and stored locally for others. Whether you are using the default behavior or the LONG IN clause to override the default behavior, a long object is created to correspond to each data partition. All the table spaces used to store long data objects corresponding to each data partition must have the same: pagesize, extentsize, storage mechanism (DMS or AMS), and type (regular or large). Remote large table spaces must be of type LARGE and cannot be SMS.

For example, the following CREATE TABLE statement creates objects for the CLOB data for each data partition in the same table space as the data:

```
CREATE TABLE document(id INT, contents CLOB)
PARTITION BY RANGE(id)
(STARTING FROM 1 ENDING AT 100 IN tbsp1,
 STARTING FROM 101 ENDING AT 200 IN tbsp2,
 STARTING FROM 201 ENDING AT 300 IN tbsp3,
 STARTING FROM 301 ENDING AT 400 IN tbsp4);
```

You can use LONG IN to place the CLOB data in one or more large table spaces, distinct from those the data is in.

```
CREATE TABLE document(id INT, contents CLOB)
PARTITION BY RANGE(id)
(STARTING FROM 1 ENDING AT 100 IN tbsp1 LONG IN large1,
 STARTING FROM 101 ENDING AT 200 IN tbsp2 LONG IN large1,
 STARTING FROM 201 ENDING AT 300 IN tbsp3 LONG IN large2,
 STARTING FROM 301 ENDING AT 400 IN tbsp4 LONG IN large2);
```

**Note:** Only a single LONG IN clause is allowed at the table level and for each data partition.

---

## Data partitions and ranges

Partitioned tables use a data organization scheme in which table data is divided across multiple storage objects called *data partitions* according to values in one or more table partitioning key columns of the table. The ranges specified for each data partition can be generated automatically or manually when creating a table.

Data partitions are referred to in various ways throughout the DB2 library. The following list represents the most common references:

- DATAPARTITIONNAME is the permanent name assigned to a data partition for a given table at create time. This column value is stored in the SYSCAT.DATAPARTITIONS catalog view. This name is not preserved on an attach or detach operation.

- DATAPARTITIONID is the permanent identifier assigned to a data partition for a given table at create time. It is used to uniquely identify a particular data partition in a given table. This identifier is not preserved on an attach or detach operation. This value is system-generated and might appear in output from various utilities.
- SEQNO indicates the order of a particular data partition range with regards to other data partition ranges in the table, with detached data partitions sorting after all visible and attached data partitions.

## **Adding data partitions to partitioned tables**

You can use the ALTER TABLE statement to modify a partitioned table after the table is created. Specifically, you can use the ADD PARTITION clause to add a new data partition to an existing partitioned table.

### **About this task**

Adding a data partition to a partitioned table is more appropriate than attaching a data partition when data is added to the data partition over time, when data is trickling in rather than rolling in from an external source, or when you are inserting or loading data directly into a partitioned table. Specific examples include daily loads of data into a data partition for January data or ongoing inserts of individual rows.

To add the new data partition to a specific table space location, the IN clause is added as an option on the ALTER TABLE ADD PARTITION statement.

To add the partitioned index of a new data partition to a specific table space location separate from the table space location of the data partition, the partition level INDEX IN clause is added as an option on the ALTER TABLE ADD PARTITION statement. If the INDEX IN option is not specified, by default any partitioned indexes on the new data partition reside in the same table space as the data partition. If any partitioned indexes exist on the partitioned table, the ADD PARTITION clause creates the corresponding empty index partitions for the new partition. A new entry is inserted into the SYSCAT.INDEXPARTITIONS catalog view for each partitioned index.

To add the LONG, LOB, or XML data of a new data partition to a specific table space location that is separate from the table space location of the data partition, the partition-level LONG IN clause is added as an option on the ALTER TABLE ADD PARTITION statement.

With DB2 V10.1 and later releases, when adding a data partition to a partitioned table by using the ALTER TABLE statement with the ADD PARTITION clause, the target partitioned table remains online, and dynamic queries against the table, running under the RS, CS, or UR isolation level, continue to run.

### **Restrictions and usage guidelines**

- You cannot add a data partition to a nonpartitioned table. For details on migrating an existing table to a partitioned table, see “Migrating existing tables and views to partitioned tables” on page 270.
- The range of values for each new data partition are determined by the STARTING and ENDING clauses.
- One or both of the STARTING and ENDING clauses must be supplied.
- The new range must not overlap with the range of an existing data partition.

- When adding a new data partition before the first existing data partition, the STARTING clause must be specified. Use MINVALUE to make this range open ended.
- Likewise, the ENDING clause must be specified if you want to add a new data partition after the last existing data partition. Use MAXVALUE to make this range open ended.
- If the STARTING clause is omitted, then the database manufactures a starting bound just after the ending bound of the previous data partition. Likewise, if the ENDING clause is omitted, the database creates an ending bound just before the starting bound of the next data partition.
- The start-clause and end-clause syntax is the same as specified in the CREATE TABLE statement.
- If the IN, INDEX IN, or LONG IN clauses are not specified for ADD PARTITION, the table space in which to place the data partition is chosen by using the same method as is used by the CREATE TABLE statement.
- Packages are invalidated during the ALTER TABLE...ADD PARTITION operation.
- The newly added data partition is available once the ALTER TABLE statement is committed.
- If a table has a nonpartitioned index, you cannot access a new data partition in that table within the same transaction as the add or attach operation that created the partition, if the transaction does not have the table locked in exclusive mode (SQL0668N, reason code 11).

Omitting the STARTING or ENDING bound for an ADD operation is also used to fill a gap in range values. Here is an example of filling in a gap by using the ADD operation where only the starting bound is specified:

```
CREATE TABLE hole (c1 int) PARTITION BY RANGE (c1)
(STARTING FROM 1 ENDING AT 10, STARTING FROM 20 ENDING AT 30);
DB20000I The SQL command completed successfully.

ALTER TABLE hole ADD PARTITION STARTING 15;
DB20000I The SQL command completed successfully.

SELECT SUBSTR(tabname, 1,12) tabname,
SUBSTR(datapartitionname, 1, 12) datapartitionname,
seqno, SUBSTR(lowvalue, 1, 4) lowvalue, SUBSTR(highvalue, 1, 4) highvalue
FROM SYSCAT.DATAPARTITIONS WHERE TABNAME='HOLE' ORDER BY seqno;
TABNAME DATAPARTITIONNAME SEQNO LOWVALUE HIGHVALUE

HOLE PART0 0 1 10
HOLE PART2 1 15 20
HOLE PART1 2 20 30

3 record(s) selected.
```

*Example 1:* Add a data partition to an existing partitioned table that holds a range of values 901 - 1000 inclusive. Assume that the SALES table holds nine ranges: 0 - 100, 101 - 200, and so on, up to the value of 900. The example adds a range at the end of the table, indicated by the exclusion of the STARTING clause:

```
ALTER TABLE sales ADD PARTITION dp10
ENDING AT 1000 INCLUSIVE
```

To add the partitioned index of a new data partition to a specific table space location separate from the table space location of the data partition, the partition level INDEX IN clause is added as an option on the ALTER TABLE ADD PARTITION statement. If no INDEX IN option is specified, by default any partitioned indexes on the new data partition reside in the same table space as the

data partition. If any partitioned indexes exist on the partitioned table, ADD PARTITION creates the corresponding empty index partitions for the new partition. A new entry is inserted into the SYSCAT.INDEXPARTITIONS catalog view for each partitioned index.

*Example 2:* Add a data partition to an existing partitioned table by separating out long data and indexes from the rest of the data partition.

```
ALTER TABLE newbusiness ADD PARTITION IN tsnewdata
INDEX IN tsnewindex LONG IN tsnewlong
```

## Attaching data partitions

Table partitioning allows for the efficient roll-in and roll-out of table data. The ALTER TABLE statement with the ATTACH PARTITION clause makes data roll-in easier.

### Before you begin

If data integrity checking, including range validation and other constraints checking, can be done through application logic that is independent of the data server before an attach operation, newly attached data can be made available for use much sooner. You can optimize the data roll-in process by using the SET INTEGRITY...ALL IMMEDIATEUNCHECKED statement to skip range and constraints violation checking. In this case, the table is brought out of SET INTEGRITY pending state, and the new data is available for applications to use immediately, as long as all user indexes on the target table are partitioned indexes.

If there are nonpartitioned indexes (except XML column path indexes) on the table to maintain after an attach operation, the SET INTEGRITY...ALL IMMEDIATEUNCHECKED statement behaves as though it were a SET INTEGRITY...IMMEDIATE CHECKED statement. All integrity processing, nonpartitioned index maintenance, and table state transitions are performed as though a SET INTEGRITY...IMMEDIATE CHECKED statement was issued. This behavior ensures that a roll-in script that uses SET INTEGRITY...ALL IMMEDIATEUNCHECKED does not stop working if a nonpartitioned index is created for the target table some time after the roll-in script is put into service.

To alter a table to attach a data partition, the privileges held by the authorization ID of the statement must include at least one of the following authorities or privileges on the source table:

- SELECT privilege on the table and DROPIN privilege on the schema of the table
- CONTROL privilege on the table
- DATAACCESS authority

### About this task

Attaching data partitions takes an existing table (source table) and attaches it to the target table as a new data partition. With DB2 V10.1 and later releases, when attaching a data partition to a partitioned table by using the ALTER TABLE statement with the ATTACH PARTITION clause, the target partitioned table remains online, and dynamic queries against the table, running under the RS, CS, or UR isolation level, continue to run.

### Restrictions and usage guidelines

The following conditions must be met before you can attach a data partition:

- The target table to which you want to attach the new data partition must be an existing partitioned table.
- The source table must be an existing nonpartitioned table or a partitioned table with a single data partition and no attached data partitions or detached data partitions. To attach multiple data partitions, you must issue multiple ATTACH statements.
- The source table cannot be a typed table.
- The source table cannot be a range-clustered table.
- The source and target table definitions must match.
- The number, type, and ordering of source and target columns must match.
- Source and target table columns must match in terms of whether they contain default values.
- Source and target table columns must match in terms of whether they allow null values.
- Source and target table compression specifications, including the VALUE COMPRESSION and COMPRESS SYSTEM DEFAULT clauses, must match.
- Source and target table specifications for the DATA CAPTURE, ACTIVATE NOT LOGGED INITIALLY, and APPEND options must match.
- Attaching a data partition is allowed even when a target column is a generated column and the corresponding source column is not a generated column. The following statement generates the values for the generated column of the attached rows:

```
SET INTEGRITY FOR table-name
 ALLOW WRITE ACCESS
 IMMEDIATE CHECKED FORCE GENERATED
```

The source table column that matches a generated column must match in type and nullability; however, a default value is not required. The recommended approach is to guarantee that the source table for the attach operation has the correct generated value in the generated column. If you follow the recommended approach, you are not required to use the FORCE GENERATED option, and the following statements can be used.

```
SET INTEGRITY FOR table-name
 GENERATED COLUMN
 IMMEDIATE UNCHECKED
```

This statement indicates that checking of the generated column is to be bypassed.

```
SET INTEGRITY FOR table-name
 ALLOW WRITE ACCESS
 IMMEDIATE CHECKED
 FOR EXCEPTION IN table-name USE table-name
```

This statement performs integrity checking of the attached data partition but does not check the generated column.

- Attaching a data partition is allowed even when the target column is an identity column and the source column is not an identity column. The statement SET INTEGRITY IMMEDIATE CHECKED does not generate identity values for the attached rows. The statement SET INTEGRITY FOR T GENERATE IDENTITY ALLOW WRITE ACCESS IMMEDIATE CHECKED fills in the identity values for the attached rows. The column that matches an identity column must match in type and nullability. There is no requirement on the default values of this column. The recommended approach is for you to fill in the correct identity

values at the staging table. Then after the ATTACH, there is no requirement to use the GENERATE IDENTITY option because the identity values are already guaranteed in the source table.

- For tables whose data is distributed across database partitions, the source table must also be distributed, in the same database partition group using the same distribution key and the same distribution map.
- The source table must be dropable (that is, it cannot have RESTRICT DROP set).
- If a data partition name is specified, it must not exist in the target table.
- If the target table is a multidimensional clustering (MDC) table, the source table must also be an MDC table.
- Using a nonpartitioned table, the data table space for the source table must match the data table spaces for the target table in type (that is, DMS or SMS), page size, extent size, and database partition group. A warning is returned if the prefetch size does not match. The index table space for the source table must match the index table spaces used by the partitioned indexes for the target table in type, database partition group, page size, and extent size. The large table space for the source table must match the large table spaces for the target table in type, database partition group, and page size. Using a partitioned table, the data table space for the source table must match the data table spaces for the target table in type, page size, extent size, and database partition group.
- When you issue the ALTER TABLE ATTACH statement to a partitioned table with any structured, XML, or LOB columns, the INLINE LENGTH of any structured, XML, or LOB columns on the source table must match with the INLINE LENGTH of the corresponding structured, XML, or LOB columns on the target table.
- When you use the REQUIRE MATCHING INDEXES clause with the ATTACH PARTITION clause, if there are any partitioned indexes on the target table that do not have a match on the source table, SQL20307N is returned.
- Attaching a source table that does not have a matching index for each partitioned unique index on the target table causes the attach operation to fail with error SQL20307N, reason code 17.
- When a table has a deferred index cleanup operation in progress as the result of an MDC rollout, since MDC rollout using the deferred index cleanup mechanism is not supported for partitioned indexes, the attach operation is not allowed if there are any RID indexes on the source table that are kept during the attach operation, not rebuilt, and are pending asynchronous index cleanup of the rolled-out blocks.
- Attaching a source table with an XML data format that is different from the XML data format of the target table is not supported.
- If a table contains XML columns that use the Version 9.5 or earlier XML record format, attaching the table to a partitioned table that contains XML columns that use the Version 9.7 or later record format is not supported.

Before attaching the table, you must update the XML record format of the table to match the record format of the target partitioned table. Either of the following two methods updates the XML record format of a table:

- Perform an online table move on the table by using the ADMIN\_MOVE\_TABLE procedure.
- Perform the following steps:
  1. Use the EXPORT command to create a copy of the table data.
  2. Use the TRUNCATE statement to delete all the rows from the table and release the storage allocated to the table.
  3. Use the LOAD command to add the data into the table.

After the XML record format of the table is updated, attach the table to the target partitioned table.

- If a table has a nonpartitioned index, you cannot access a new data partition in that table within the same transaction as the add or attach operation that created the partition, if the transaction does not have the table locked in exclusive mode (SQL0668N, reason code 11).

Before running the attach operation, create indexes on the source table that match each of the partitioned indexes in the target table. Matching the partitioned indexes makes the roll-in operation more efficient and less active log space is needed. If the indexes on the source table are not properly prepared, the database manager is required to maintain them for you. To ensure that your roll-in does not incur any additional cost to maintain the partitioned indexes, you can specify REQUIRE MATCHING INDEXES on the attach partition operation. Specifying REQUIRE MATCHING INDEXES ensures that the attach operation fails if a source table does not have indexes to match the partitioned indexes on the target. You can then take the corrective action and reissue the attach operation.

In addition, drop any extra indexes on the source table before running the attach operation. Extra indexes are those indexes on the source table that either do not have a match on the target table, or that match nonpartitioned indexes on the target table. Dropping extra indexes before running the attach operation makes it run faster.

For example, assume that a partitioned table called ORDERS has 12 data partitions (one for each month of the year). At the end of each month, a separate table called NEWORDERS is attached to the partitioned ORDERS table.

1. Create partitioned indexes on the ORDERS table.

```
CREATE INDEX idx_delivery_date ON orders(delivery) PARTITIONED
CREATE INDEX idx_order_price ON orders(price) PARTITIONED
```

2. Prepare for the attach operation by creating the corresponding indexes on the NEWORDERS table.

```
CREATE INDEX idx_delivery_date_for_attach ON neworders(delivery)
CREATE INDEX idx_order_price_for_attach ON neworders(price)
```

3. There are two steps to the attach operation:

- a. ATTACH. The indexes on the NEWORDERS table that match the partitioned indexes on the ORDERS table are kept.

```
ALTER TABLE orders ATTACH PARTITION part_jan2009
STARTING FROM ('01/01/2009')
ENDING AT ('01/31/2009') FROM TABLE neworders
```

The ORDERS table is automatically placed into the Set Integrity Pending state. Both the idx\_delivery\_date\_for\_attach index and the idx\_order\_price\_for\_attach index become part of the ORDERS table after the completion of the attach operation. No data movement occurs during this operation.

- b. SET INTEGRITY. A range check is done on the newly attached partition. Any constraints that exist are enforced. Upon completion, the newly attached data becomes visible within the database.

```
SET INTEGRITY FOR orders IMMEDIATE CHECKED
```

When nonpartitioned indexes exist on the target table, the SET INTEGRITY statement has to maintain the index along with other tasks, such as range validation and constraints checking on the data from the newly attached partition. Nonpartitioned index maintenance requires a large amount of active log space that

is proportional to the data volumes in the newly attached partition, the key size of each nonpartitioned index, and the number of nonpartitioned indexes.

Each partitioned index on the new data partition is given an entry in the SYSINDEXPARTITIONS catalog table using the table space identifier and object identifier from the source table. The identifier information is taken from either the SYSINDEXES table (if the table is nonpartitioned) or the SYSINDEXPARTITIONS table (if the table is partitioned). The index identifier is taken from the partitioned index of the matching target table.

When the source table is partitioned, those partitioned indexes on the source table that match the partitioned indexes on the target table are kept as part of the attach operation. Index partition entries in the SYSINDEXPARTITIONS table are updated to show that they are index partitions on the new target table with new index identifiers.

When attaching data partitions, some statistics for indexes as well as data are carried over from the source table to the target table for the new partition. Specifically, all fields in the SYSDATAPARTITIONS and SYSINDEXPARTITIONS tables for the new partition on the target are populated from the source. When the source table is nonpartitioned, these statistics come from the SYSTABLES and SYSINDEXES tables. When the source table is a single-partition partitioned table, these statistics come from the SYSDATAPARTITIONS and SYSINDEXPARTITIONS tables of the single source partition.

**Note:** Execute a runstats operation after the completion of an attach operation, because the statistics that are carried over will not affect the aggregated statistics in the SYSINDEXES and SYSTABLES tables.

**Nonpartitioned index maintenance during SET INTEGRITY...ALL IMMEDIATE UNCHECKED.** When SET INTEGRITY...ALL IMMEDIATE UNCHECKED is issued on a partitioned table to skip range checking for a newly attached partition, if there are any nonpartitioned indexes (except the XML column path index) on the table, SET INTEGRITY...ALL IMMEDIATE UNCHECKED performs as follows:

- If the SET INTEGRITY...ALL IMMEDIATE UNCHECKED statement references one target table, the behavior is as though a SET INTEGRITY...ALLOW WRITE ACCESS...IMMEDIATE CHECKED statement was issued instead. The SET INTEGRITY...ALL IMMEDIATE UNCHECKED statement maintains all nonpartitioned indexes (except XML column path indexes), performs all other integrity processing, updates the constraints checking flag values in the CONST\_CHECKED column in the SYSCAT.TABLES catalog view, and returns errors and stops immediately when constraints violations are detected.
- If the SET INTEGRITY...ALL IMMEDIATE UNCHECKED statement references more than one target table, an error is returned (SQL20209N with reason code 13).

**Rebuild of invalid partitioned indexes during SET INTEGRITY.** The SET INTEGRITY statement can detect whether the partitioned index object for a newly attached partition is invalid and performs a partitioned index rebuild if necessary.

### **Guidelines for attaching data partitions to partitioned tables**

This topic provides guidelines for correcting various types of mismatches that can occur when attempting to attach a data partition to a partitioned table when issuing the ALTER TABLE ...ATTACH PARTITION statement. You can achieve

agreement between tables by modifying the source table to match the characteristics of the target table, or by modifying the target table to match the characteristics of the source table.

The source table is the existing table you want to attach to a target table. The target table is the table to which you want to attach the new data partition.

One suggested approach to performing a successful attach is to use the exact CREATE TABLE statement for the source table as you did for the target table, but without the PARTITION BY clause. In cases where it is difficult to modify the characteristics of either the source or target tables for compatibility, you can create a new source table that is compatible with the target table. For details on creating a new source, see “Creating tables like existing tables”.

To help you prevent a mismatch from occurring, see the restrictions and usage guidelines section of “Attaching data partitions”. The section outlines conditions that must be met before you can successfully attach a data partition. Failure to meet the listed conditions returns error SQL20408N or SQL20307N.

The following sections describe the various types of mismatches that can occur and provides the suggested steps to achieve agreement between tables:

**The (value) compression clause (the COMPRESSION column of SYSCAT.TABLES) does not match. (SQL20307N reason code 2)**

To achieve value compression agreement, use one of the following statements:

ALTER TABLE... ACTIVATE VALUE COMPRESSION  
or  
ALTER TABLE... DEACTIVATE VALUE COMPRESSION

To achieve row compression agreement use one of the following statements:

ALTER TABLE... COMPRESS YES  
or  
ALTER TABLE... COMPRESS NO

**The APPEND mode of the tables does not match. (SQL20307N reason code 3)**

To achieve append mode agreement use one of the following statements:

ALTER TABLE ... APPEND ON  
or  
ALTER TABLE ... APPEND OFF

**The code pages of the source and target table do not match. (SQL20307N reason code 4)**

Create a new source

**The source table is a partitioned table with more than one data partition or with attached or detached data partitions. (SQL20307N reason code 5)**

Detach data partitions from the source table until there is a single visible data partition using the statement:

ALTER TABLE ... DETACH PARTITION

Detached partitions remain detached until each of the following steps has been completed:

1. Execute any necessary SET INTEGRITY statements to incrementally refresh detached dependents.
2. In Version 9.7.1 and later, wait for the detach to complete asynchronously. To expedite this process, ensure that all access to the table that started prior to the detach operation either completes or is terminated.
3. If the source table has nonpartitioned indexes, wait for the asynchronous index cleanup to complete. To expedite this process, one option might be to drop the nonpartitioned indexes on the source table.

If you want to perform an attach operation immediately, one option might be to create a new source table.

**The source table is a system table, a view, a typed table, a table ORGANIZED BY KEY SEQUENCE, a created temporary table, or a declared temporary table. (SQL20307N reason code 6)**

Create a new source.

**The target and source table are the same. (SQL20307N reason code 7)**

You cannot attach a table to itself. Determine the correct table to use as the source or target table.

**The NOT LOGGED INITIALLY clause was specified for either the source table or the target table, but not for both. (SQL20307N reason code 8)**

Either make the table that is not logged initially be logged by issuing the COMMIT statement, or make the table that is logged be not logged initially by entering the statement:

```
ALTER TABLE ... ACTIVATE NOT LOGGED INITIALLY
```

**The DATA CAPTURE CHANGES clause was specified for either the source table or the target table, but not both. (SQL20307N reason code 9)**

To enable data capture changes on the table that does not have data capture changes turned on, run the following statement:

```
ALTER TABLE ... DATA CAPTURE CHANGES
```

To disable data capture changes on the table that does have data capture changes turned on, run the statement:

```
ALTER TABLE ... DATA CAPTURE NONE
```

**The distribution clauses of the tables do not match. The distribution key must be the same for the source table and the target table. (SQL20307N reason code 10)**

It is recommended that you create a new source table. You cannot change the distribution key of a table spanning multiple database partitions. To change a distribution key on tables in single-partition database, run the following statements:

```
ALTER TABLE ... DROP DISTRIBUTION;
ALTER TABLE ... ADD DISTRIBUTION(key-specification)
```

**An error is returned when there are missing indexes during an attach operation (SQL20307N reason code 18)**

The attach operation implicitly builds missing indexes on the source table corresponding to the partitioned indexes on the target table. The implicit creation of the missing indexes does take time to complete. You have an option to create and error condition if the attach operation encounters any missing indexes. The option is called ERROR ON MISSING INDEXES and is one of the attach operation options. The error returned when this happens is SQL20307N, SQLSTATE 428GE, reason code 18. Information on the nonmatching indexes is placed in the administration log.

The attach operation drops indexes on the source table that do not match the partitioned indexes on the target table. The identification and dropping of these nonmatching indexes takes time to complete. You should drop these indexes before attempting the attach operation.

**An error is returned when the nonmatching indexes on the target table are unique indexes, or the XML indexes are defined with the REJECT INVALID VALUES clause, during an attach operation (SQL20307N reason code 17)**

When there are partitioned indexes on the target table with no matching indexes on the source table and the ERROR ON MISSING INDEXES is not used, then you could expect the following results:

1. If the nonmatching indexes on the target table are unique indexes, or the XML indexes are defined with the REJECT INVALID VALUES clause, then the attach operation will fail and return the error message SQL20307N, SQLSTATE 428GE, reason code 17.
2. If the nonmatching indexes on the target table do not meet the conditions in the previous point, the index object on the source table is marked invalid during the attach operation. The attach operation completes successfully, but the index object on the new data partition is marked invalid. The SET INTEGRITY operation is used to rebuild the index objects on the newly attached partition. Typically this is the next operation you would perform following the attaching of a data partition. The recreation of the indexes takes time.

The administration log will have details about any mismatches between the indexes on the source and target tables.

**Only one of the tables has an ORGANIZE BY DIMENSIONS clause specified or the organizing dimensions are different. (SQL20307N reason code 11)**

Create a new source.

**The data type of the columns (TYPENAME) does not match. (SQL20408N reason code 1)**

To correct a mismatch in data type, issue the statement:

`ALTER TABLE ... ALTER COLUMN ... SET DATA TYPE...`

**The nullability of the columns (NULLS) does not match. (SQL20408N reason code 2)**

To alter the nullability of the column that does not match for one of the tables issue one of the following statements:

`ALTER TABLE... ALTER COLUMN... DROP NOT NULL`  
or  
`ALTER TABLE... ALTER COLUMN... SET NOT NULL`

**The implicit default value ( SYSCAT.COLUMNS IMPLICITVALUE) of the columns are incompatible. (SQL20408N reason code 3)**

Create a new source table. Implicit defaults must match exactly if both the target table column and source table column have implicit defaults (if IMPLICITVALUE is not NULL).

If IMPLICITVALUE is not NULL for a column in the target table and IMPLICITVALUE is not NULL for the corresponding column in the source table, each column was added after the original CREATE TABLE statement for the table. In this case, the value stored in IMPLICITVALUE must match for this column.

There is a situation, where through migration from a pre-V9.1 table or through attach of a data partition from a pre-V9.1 table, that IMPLICITVALUE is not NULL because the system did not know whether or not the column was added after the original CREATE TABLE statement. If the database is not certain whether the column is added or not, it is treated as added. An added column is a column created as the result of an ALTER TABLE ...ADD COLUMN statement. In this case, the statement is not allowed because the value of the column could become corrupted if the attach were allowed to proceed. You must copy the data from the source table to a new table (with IMPLICITVALUE for this column NULL) and use the new table as the source table for the attach operation.

**The code page (COMPOSITE\_CODEPAGE) of the columns does not match. (SQL20408N reason code 4)**

Create a new source table.

**The system compression default clause (COMPRESS) does not match. (SQL20408N reason code 5)**

To alter the system compression of the column issue one of the following statements to correct the mismatch:

ALTER TABLE ... ALTER COLUMN ... COMPRESS SYSTEM DEFAULT  
or  
ALTER TABLE ... ALTER COLUMN ... COMPRESS OFF

### **Conditions for matching a source table index with a target table partitioned index during ATTACH PARTITION**

All index key columns of the partitioned index on the target table must match with the index key columns of the index on the source table. If all other properties of the index are the same, then the index on the source table is considered a match to the partitioned index on the target table. That is, the index on the source table can be used as an index on the target table. The table here can be used to determine if the indexes are considered a match or not.

The following table is only useful and applicable when the target index is partitioned. The target index property is assumed by the source index in all cases where they are considered a match.

*Table 21. Determining whether the source index matches when the target index property is different from the source index property.*

| Rule number | Target index property | Source index property | Does the source index match?           |
|-------------|-----------------------|-----------------------|----------------------------------------|
| 1.          | non-unique            | unique                | Yes, if the index is not an XML index. |

*Table 21. Determining whether the source index matches when the target index property is different from the source index property. (continued)*

| Rule number | Target index property         | Source index property         | Does the source index match?                                                                                                                                                                                                                                |
|-------------|-------------------------------|-------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 2.          | unique                        | non-unique                    | No                                                                                                                                                                                                                                                          |
| 3.          | column X is descending        | column X is ascending         | No                                                                                                                                                                                                                                                          |
| 4.          | column X is ascending         | column X is descending        | No                                                                                                                                                                                                                                                          |
| 5.          | partitioned                   | nonpartitioned                | No. Note: this assumes the source table is partitioned.                                                                                                                                                                                                     |
| 6.          | pctfree n1                    | pctfree n2                    | Yes                                                                                                                                                                                                                                                         |
| 7.          | level2pctfree n1              | level2pctfree n2              | Yes                                                                                                                                                                                                                                                         |
| 8.          | minpctused n1                 | minpctused n2                 | Yes                                                                                                                                                                                                                                                         |
| 9.          | disallow reverse scans        | allow reverse scans           | Yes, the physical index structure is the same irrespective of whether reverse scans are allowed or not.                                                                                                                                                     |
| 10.         | allow reverse scans           | disallow reverse scans        | Yes, the same reason as (9).                                                                                                                                                                                                                                |
| 11.         | pagesplit [L H S]             | pagesplit [L H S]             | Yes                                                                                                                                                                                                                                                         |
| 12.         | sampled statistics            | detailed statistics           | Yes                                                                                                                                                                                                                                                         |
| 13.         | detailed statistics           | sampled statistics            | Yes                                                                                                                                                                                                                                                         |
| 14.         | not clustered                 | CLUSTER                       | Yes                                                                                                                                                                                                                                                         |
| 15.         | CLUSTER                       | not clustered                 | Yes. The index will become a clustering index but the data will not be clustered according to this index until the data is reorganized. You can use a partition level reorganization after attaching to cluster the data according to this index partition. |
| 16.         | ignore invalid                | reject invalid                | Yes                                                                                                                                                                                                                                                         |
| 17.         | reject invalid                | ignore invalid                | No. The target index property of rejecting invalid values needs to be respected and the source table may have rows that violate this index constraint.                                                                                                      |
| 18.         | Index compression enabled     | Index compression not enabled | Yes. Note: compression of the underlying index data will not occur until the index is rebuilt.                                                                                                                                                              |
| 19.         | Index compression not enabled | Index compression enabled     | Yes. Note: decompressing the index data will not occur until the index is rebuilt.                                                                                                                                                                          |

**Note:** With rule number 5, an ALTER TABLE ... ATTACH PARTITION statement fails returning error message SQL20307N, SQLSTATE 428GE, if you attempt to attach a multidimensional clustering (MDC) table created using DB2 Version 9.7 or earlier releases (having nonpartitioned block indexes) to a new MDC partitioned table created using DB2 Version 9.7 Fix Pack 1 or later releases (having partitioned block indexes) and the ERROR ON MISSING INDEXES clause is used. Removing

the ERROR ON MISSING INDEXES clause allows the attachment to complete because the database manager maintains the indexes during the attach operation. If you received error message SQL20307N, SQLSTATE 428GE, you should consider removing the ERROR ON MISSING INDEXES clause.

An alternative is to use the online table move procedure to convert an MDC partitioned table that has nonpartitioned block indexes to a table that has partitioned block indexes.

## Detaching data partitions

Table partitioning allows for the efficient roll-in and roll-out of table data. This efficiency is achieved by using the ATTACH PARTITION and DETACH PARTITION clauses of the ALTER TABLE statement.

### Before you begin

To detach a data partition from a partitioned table you must have the following authorities or privileges:

- The user performing the DETACH PARTITION operation must have the authority necessary to ALTER, to SELECT from and to DELETE from the source table.
- The user must also have the authority necessary to create the target table. Therefore, to alter a table to detach a data partition, the privilege held by the authorization ID of the statement must include at least one of the following authorities or privileges on the target table:
  - DBADM authority
  - CREATETAB authority on the database and USE privilege on the table spaces used by the table as well as one of:
    - IMPLICIT\_SCHEMA authority on the database, if the implicit or explicit schema name of the table does not exist
    - CREATEIN privilege on the schema, if the schema name of the table refers to an existing schema.

**Note:** When detaching a data partition, the authorization ID of the statement is going to effectively perform a CREATE TABLE statement and therefore must have the necessary privileges to perform that operation. The authorization ID of the ALTER TABLE statement becomes the definer of the new table with CONTROL authority, as if the user had issued the CREATE TABLE statement. No privileges from the table being altered are transferred to the new table. Only the authorization ID of the ALTER TABLE statement and users with DBADM or DATAACCESS authority have access to the data immediately after the ALTER TABLE...DETACH PARTITION statement.

### About this task

Rolling-out partitioned table data allows you to easily separate ranges of data from a partitioned table. Once a data partition is detached into a separate table, the table can be handled in several ways. You can drop the separate table (whereby, the data from the data partition is destroyed); archive it or otherwise use it as a separate table; attach it to another partitioned table such as a history table; or you can manipulate, cleanse, transform, and reattach to the original or some other partitioned table.

With DB2 Version 9.7 Fix Pack 1 and later releases, when detaching a data partition from a partitioned table by using the ALTER TABLE statement with the DETACH PARTITION clause, the source partitioned table remains online. Queries running against the table continue to run. The data partition being detached is converted into a stand-alone table in the following two-phase process:

1. The ALTER TABLE...DETACH PARTITION operation logically detaches the data partition from the partitioned table.
2. An asynchronous partition detach task converts the logically detached partition into a stand-alone table.

If there are any dependent tables that need to be incrementally maintained with respect to the detached data partition (these dependent tables are referred to as detached dependent tables), the asynchronous partition detach task starts only after the SET INTEGRITY statement is run on all detached dependent tables.

In absence of detached dependent tables, the asynchronous partition detach task starts after the transaction issuing the ALTER TABLE...DETACH PARTITION statement commits.

#### Restrictions

If the source table is an MDC table created by DB2 Version 9.7 or earlier releases, block indexes are not partitioned. Access to the newly detached table is not allowed in the same unit of work as the ALTER TABLE...DETACH PARTITION operation. MDC tables do not support partitioned block indexes. In that case, block indexes are created upon first access to the table after the ALTER TABLE...DETACH PARTITION operation is committed. If the source table had any other partitioned indexes before detach time then the index object for the target table is marked invalid to allow for creation of the block indexes. As a result access time is increased while the block indexes are created and any partitioned indexes are recreated.

When the source table is an MDC created by DB2 V9.7 Fix Pack 1 or later releases, the block indexes are partitioned, and partitioned indexes become indexes on the target table of detach without the need to be recreated.

You must meet the following conditions before you can perform a DETACH PARTITION operation:

- The table to be detached from (source table) must exist and be a partitioned table.
- The data partition to be detached must exist in the source table.
- The source table must have more than one data partition. A partitioned table must have at least one data partition. Only visible and attached data partitions pertain in this context. An attached data partition is a data partition that is attached but not yet validated by the SET INTEGRITY statement.
- The name of the table to be created by the DETACH PARTITION operation (target table) must not exist.
- DETACH PARTITION is not allowed on a table that is the parent of an enforced referential integrity (RI) relationship. If you have tables with an enforced RI relationship and want to detach a data partition from the parent table, a workaround is available. In the following example, all statements are run within the same unit of work (UOW) to lock out concurrent updates:

```

// Change the RI constraint to informational:
ALTER TABLE child ALTER FOREIGN KEY fk NOT ENFORCED;

ALTER TABLE parent DETACH PARTITION p0 INTO TABLE pdet;

SET INTEGRITY FOR child OFF;

// Change the RI constraint back to enforced:
ALTER TABLE child ALTER FOREIGN KEY fk ENFORCED;

SET INTEGRITY FOR child ALL IMMEDIATE UNCHECKED;
// Assuming that the CHILD table does not have any dependencies on partition P0,
// and that no updates on the CHILD table are permitted
// until this UOW is complete, no RI violation is possible during this UOW.

COMMIT WORK;

```

- If there are any dependent tables that need to be incrementally maintained with respect to the detached data partition (these dependent tables are referred to as detached dependent tables), the SET INTEGRITY statement is required to be run on the detached dependent tables to incrementally maintain the tables. With DB2 V9.7 Fix Pack 1 or later releases, after the SET INTEGRITY statement is run on all detached dependent tables, the asynchronous partition detach task makes the data partition into a stand-alone target table. Until the asynchronous partition detach operation completes, the target table is unavailable.

## Procedure

1. To alter a partitioned table and to detach a data partition from the table, issue the ALTER TABLE statement with the DETACH PARTITION clause.
2. Optional: If you wish to have the same constraints on the newly detached stand-alone table, run the ALTER TABLE... ADD CONSTRAINT on the target table after completing the detach operation.  
If the index was partitioned on the source table, any indexes necessary to satisfy the constraint already exist on the target table.

## Results

The detached partition is renamed with a system-generated name (using the form SQLyymmddhhmmssxxx) so that a subsequent attach can reuse the detached partition name immediately.

Each of the index partitions defined on the source table for the data partition being detached becomes an index on the target table. The index object is not physically moved during the detach partition operation. However, the metadata for the index partitions of the table partition being detached are removed from the catalog table SYSINDEXPARTITIONS. New index entries are added in SYSINDEXES for the new table as a result of the detach partition operation. The original index identifier (IID) is kept and stays unique just as it was on the source table.

The index names for the surviving indexes on the target table are system-generated (using the form SQLyymmddhhmmssxxx). The schema for these indexes is the same as the schema of the target table except for any path indexes, regions indexes, and MDC or ITC block indexes, which are in the SYSIBM schema. Other system-generated indexes like those to enforce unique and primary key constraints will have a schema of the target table because the indexes are carried over to the detached table but the constraints are not. You can use the RENAME statement to rename the indexes that are not in the SYSIBM schema.

The table level INDEX IN option specified when creating the source table is not inherited by the target table. Rather, the partition level INDEX IN (if specified) or the default index table space for the detach partition continues to be the index table space for the target table.

When detaching data partitions, some statistics are carried over from the partition being detached into the target table. Specifically, statistics from SYSINDEXPARTITIONS for partitioned indexes will be carried over to the entries SYSINDEXES for the newly detached table. Statistics from SYSDATAPARTITIONS will be copied over to SYSTABLES for the newly detached table.

## What to do next

Run **RUNSTATS** after the completion of the DETACH PARTITION operation on both the new detached table and the source table, because many of the statistics will not be carried over following the completion of the detach partition operation.

### Attributes of detached data partitions

When you detach a data partition from a partitioned table using the DETACH PARTITION clause of the ALTER TABLE statement, it becomes a stand-alone, nonpartitioned target table. Many attributes of the new target table are inherited from the source table. Any attributes not inherited from the source table are set as if the user executing the DETACH operation is creating the target table.

The target table after DETACH will inherit all the partitioned indexes defined on the source table. These indexes includes both system-generated indexes or user-defined indexes. The index object is not physically moved during the detach operation. The index partition metadata of the datapartition being detached is removed from the SYSINDEXPARTITIONS catalog. New entries are added in SYSINDEXES for the new table. The index identifier (IID) for any given partitioned index from the source table will be the IID for the index on the target table (the IID will remain unique with respect to the table, and unchanged during the detach).

The index name for the surviving indexes on the new table are system-generated with the form: SQLyymmddhhmmssxx. Path indexes, region indexes, and MDC or ITC indexes are made part of the SYSIBM schema. All other indexes are made part of the schema of the new table. System-generated indexes like those to enforce unique and primary key constraints are made part of the schema of the new table because the indexes are carried over to the new table. Constraints on the source table will not be inherited by the target table after DETACH.

You can use the RENAME statement to rename the indexes not in the SYSIBM schema at another time.

You can use the ALTER TABLE ... ADD CONSTRAINT statement on the new table following the completion of the detach operation to enforce the same constraints on the new table as on the source table.

The table space location specified by the table-level INDEX IN clause on the source table is not inherited by the new target table. Rather, the table space location specified by the partition-level INDEX IN clause, or the default index table space for the new table, continues as the index table space location for the new table.

### Attributes inherited by the target table

Attributes inherited by the target table include:

- The following column definitions:
  - Column name
  - Data type (includes length and precision for types that have length and precision, such as CHAR and DECIMAL)
  - NULLability
  - Column default values
  - INLINE LENGTH
  - Code page (CODEPAGE column of SYSCAT.COLUMNS catalog view)
  - Logging for LOBs (LOGGED column of SYSCAT.COLUMNS catalog view)
  - Compaction for LOBs (COMPACT column of SYSCAT.COLUMNS catalog view)
  - Compression (COMPRESS column of SYSCAT.COLUMNS catalog view)
  - Type of hidden column (HIDDEN column of SYSCAT.COLUMNS catalog view)
  - Column order
- If the source table is a multidimensional clustering (MDC) or insert time clustering (ITC) table, the target table is also an MDC or ITC table, defined with the same dimension columns.
- Block index definitions. The indexes are rebuilt on first access to the newly detached independent table after the DETACH operation is committed.
- The table space id and table object id are inherited from the data partition, not from the source table. This is because no table data is moved during a DETACH operation. In catalog terms, the TBSPACEID column of the SYSCAT.DATAPARTITIONS catalog view from the source data partition becomes the TBSPACEID column of the SYSCAT.TABLES catalog view. When translated into a table space name, it is the TBSPACE column of SYSCAT.TABLES catalog view in the target table. The PARTITIONOBJECTID column of the SYSCAT.DATAPARTITIONS catalog view from the source data partition becomes the TABLEID column of the SYSCAT.TABLES catalog view in the target table.
- The LONG\_TBSPACEID column of the SYSCAT.DATAPARTITIONS catalog view from the source data partition is translated into a table space name and becomes the LONG\_TBSPACE column of SYSCAT.TABLES of the target table.
- The INDEX\_TBSPACEID column value in the SYSDATAPARTITIONS for the source data partition (the partition level index table space) is translated into a table space name and becomes the INDEX\_TBSPACE value in SYSTABLES for the target table. The index table space specified by table level INDEX IN <table space> in the CREATE TABLE statement will not be inherited by the target table.
- Table space location
- ID of distribution map for a multi-partition database (PMAP\_ID column of SYSCAT.TABLES catalog view)
- Percent free (PCTFREE column of SYSCAT.TABLES catalog view)
- Append mode (APPEND\_MODE column of SYSCAT.TABLES catalog view)
- Preferred lock granularity (LOCKSIZE column of SYSCAT.TABLES catalog view)
- Data Capture (DATA\_CAPTURE column of SYSCAT.TABLES catalog view)
- VOLATILE (VOLATILE column of SYSCAT.TABLES catalog view)
- DROPRULE (DROPRULE column of SYSCAT.TABLES catalog view)
- Compression (COMPRESSION column of SYSCAT.TABLES catalog view)
- Maximum free space search (MAXFREESPACESEARCH column of SYSCAT.TABLES catalog view)

**Note:** Partitioned hierarchical or temporary tables, range-clustered tables, and partitioned views are not supported.

#### Attributes not inherited from the source table

Attributes not inherited from the source table include:

- The target table type is not inherited. The target table is always a regular table.
- Privileges and Authorities
- Schema
- Generated columns, identity columns, check constraints, referential constraints. In the case where a source column is a generated column or an identity column, the corresponding target column has no explicit default value, meaning it has a default value of NULL.
- Table level index table space (INDEX\_TBSPACE column of the SYSCAT.TABLES catalog view). Indexes for the table resulting from the DETACH will be in the same table space as the table.
- Triggers
- Primary key constraints and unique key constraints
- Statistics for nonpartitioned indexes will not be inherited.
- All other attributes not mentioned in the list of attributes explicitly inherited from the source table.

#### Data partition detach phases

With DB2 Version 9.7 Fix Pack 1 and later releases, detaching a data partition from a data partitioned table consists of two phases. The first phase logically detaches the partition from the table, the second phase converts the data partition into a stand-alone table.

The detach process is initiated when an ALTER TABLE...DETACH PARTITION statement is issued:

1. The ALTER TABLE...DETACH PARTITION operation logically detaches the data partition from the partitioned table.
2. An asynchronous partition detach task converts the logically detached partition into the stand-alone table.

If there are any dependent tables that need to be incrementally maintained with respect to the detached data partition (these dependent tables are referred to as detached dependent tables), the asynchronous partition detach task starts only after the SET INTEGRITY statement is run on all detached dependent tables.

In absence of detached dependent tables, the asynchronous partition detach task starts after the transaction issuing the ALTER TABLE...DETACH PARTITION statement commits.

#### DETACH operation

The ALTER TABLE...DETACH PARTITION operation performs in the following manner:

- The DETACH operation does not wait for dynamic uncommitted read (UR) isolation level queries before it proceeds, nor does it interrupt any currently running dynamic UR queries. This behavior occurs even when the UR query is accessing the partition being detached.

- If dynamic non-UR queries (read or write queries) did not lock the partition to be detached, the DETACH operation can complete while dynamic non-UR queries are running against the table.
- If dynamic non-UR queries locked the partition to be detached, the DETACH operation waits for the lock to be released.
- Hard invalidation must occur on all static packages that are dependent on the table before the DETACH operation can proceed.
- The following restrictions that apply to data definition language (DDL) statements also apply to a DETACH operation because DETACH requires catalogs to be updated:
  - New queries cannot be compiled against the table.
  - A bind or rebind cannot be performed on queries that run against the table.

To minimize the impact of these restrictions, issue a COMMIT immediately after a DETACH operation.

During the DETACH operation, the data partition name is changed to a system-generated name of the form SQLyyymmddhhmmssxxx, and in SYSCAT.DATAPARTITIONS, the status of the partition is set to 'L' if there are no detached dependent tables, or 'D' if there are detached dependent tables.

During the DETACH operation, an entry is created in SYSCAT.TABLES for the target table. If there are detached dependent tables, the table TYPE is set to 'L'. After SET INTEGRITY is run on all detached dependent tables, the TYPE is set to 'T', however, the target table continues to be unavailable. The asynchronous partition detach task completes the detach and makes the target table available.

Soft invalidation of dynamic SQL during the DETACH operation allows dynamic SQL queries that started before the ALTER TABLE...DETACH PARTITION statement to continue running concurrently with the DETACH operation. The ALTER TABLE...DETACH PARTITION statement acquires an IX lock on the partitioned table and an X lock on the data partition being detached.

### **Asynchronous partition detach task**

After the DETACH operation commits and any detached dependent tables are refreshed, the asynchronous partition detach task converts the logically detached partition into the stand-alone table.

The asynchronous partition detach task waits for the completion of all access on the partitioned table that started before phase 1 of the detach operation. If the partitioned table has nonpartitioned indexes, the asynchronous partition detach task creates the asynchronous index cleanup task for deferred indexed cleanup. After the access completes, the asynchronous partition detach task completes phase 2 of the detached operation, by converting the logically detached partition into a stand-alone table.

The **LIST UTILITIES** command can be used to monitor the process of the asynchronous partition detach task. The **LIST UTILITIES** command indicates whether the asynchronous partition detach task is in one of the following states:

- Waiting for old access to the partitioned table to complete
- Finalizing the detach operation and making the target table available

## **Asynchronous partition detach for data partitioned tables**

For DB2 Version 9.7 Fix Pack 1 and later releases, the asynchronous partition detach task completes the detach of a data partition from a partitioned table that was initiated by an ALTER TABLE...DETACH operation. The task is an asynchronous background process (ABP) that is initiated after the partition becomes a logically detached partition.

The asynchronous partition detach task accelerates the process of detaching a data partition from a partitioned table. If the partitioned table has dependent materialized query tables (MQTs), the task is not initiated until after a SET INTEGRITY statement is executed on the MQTs.

By completing the detach of the data partition asynchronously, queries accessing the partitioned table that started prior to issuing ALTER TABLE...DETACH PARTITION statement continue while the partition is immediately detached.

If there are any dependent tables that need to be incrementally maintained with respect to the detached data partition (these dependent tables are referred to as detached dependent tables), the asynchronous partition detach task starts only after the SET INTEGRITY statement is run on all detached dependent tables.

In the absence of detached dependents, the asynchronous partition detach task starts after the transaction issuing the ALTER TABLE...DETACH PARTITION statement commits.

The asynchronous partition detach task performs the following operations:

- Performs hard invalidation on cached statements on which the ALTER TABLE...DETACH operation previously performed soft invalidation.
- Updates catalog entries for source partitioned table and target stand-alone table and makes the target table available.
- For multidimensional clustering (MDC) tables with nonpartitioned block indexes and no other partitioned indexes, creates an index object for target table. The block indexes are created upon first access to the target table after the asynchronous partition detach task commits.
- Creates the system path index on the target table for table containing XML columns.
- Updates the minimum recovery time (MRT) of the table space containing the detached partition.
- Creates asynchronous index cleanup AIC tasks for nonpartitioned indexes. The AIC task performs index cleanup after asynchronous partition detach completes.
- Releases the data partition ID if nonpartitioned indexes do not exist on the table.

## **Asynchronous partition detach task impact on performance**

An asynchronous partition detach task incurs minimal performance impact. The task waits for all access to the detached partition to complete by performing a hard invalidation on cached statements on which the ALTER TABLE...DETACH operation previously performed soft invalidation. Then the task acquires the required locks on the table and on the partition and continues the process to make the detached partition a stand-alone table.

## Monitoring the asynchronous partition detach task

The distribution daemon and asynchronous partition detach task agents are internal system applications that appear in **LIST APPLICATIONS** command output with the application names **db2taskd** and **db2apd**, respectively. To prevent accidental disruption, system applications cannot be forced. The distribution daemon remains online as long as the database is active. The tasks remain active until detach completes. If the database is deactivated while detach is in progress, the asynchronous partition detach task resumes when the database is reactivated.

The **LIST UTILITIES** command indicates whether the asynchronous partition detach task is in one of the following states:

- Waiting for old access to the partitioned table to complete
- Finalizing the detach operation and making the target table available

The following sample output for the **LIST UTILITIES SHOW DETAIL** command shows asynchronous partition detach task activity in the WSDB database:

```
ID = 1
Type = ASYNCHRONOUS PARTITION DETACH
Database Name = WSDB
Partition Number = 0
Description = Finalize the detach for partition '4' of table 'USER1.ORDERS'.
Start Time = 07/15/2009 14:52:14.476131
State = Executing
Invocation Type = Automatic
Progress Monitoring:
 Description = Waiting for old access to the partitioned table to complete.
 Start Time = 07/15/2009 14:52:51.268119
```

In the output of the **LIST UTILITIES** command, the main description for the asynchronous partition detach task identifies the data partition being detached and the target table created by the detach operation. The progress monitoring description provides information about the current state of the asynchronous partition detach task.

**Note:** The asynchronous partition detach task is an asynchronous process. To know when the target table of a detach operation is available, a stored procedure can be created that queries the **STATUS** column of the **SYSCAT.DATAPARTITIONS** catalog view and returns when the detach operation completes.

## Asynchronous partition detach processing in a partitioned database environment

One asynchronous partition detach task is created for each **DETACH** operation independent of the number of database partitions in a partitioned database environment. The task is created on the catalog database partition and distributes work to the remaining database partitions, as needed.

## Error handling for the asynchronous partition detach task

The asynchronous partition detach task is transaction based. All the changes made by a task will be rolled back internally if it fails. Any errors during asynchronous partition detach processing are logged in a **db2diag** log file. A failed task is retried later by the system.

## Dropping data partitions

To drop a data partition, you detach the partition, and drop the table created by the detach operation. Use the ALTER TABLE statement with the DETACH PARTITION clause to detach the partition and create a stand-alone table, and use the DROP TABLE statement to drop the table.

### Before you begin

To detach a data partition from a partitioned table the user must have the following authorities or privileges:

- The user performing the DETACH operation must have the authority to ALTER, to SELECT from and to DELETE from the source table.
- The user must also have the authority to CREATE the target table. Therefore, in order to alter a table to detach a data partition, the privilege held by the authorization ID of the statement must include at least one of the following on the target table:
  - DBADM authority
  - CREATETAB authority on the database and USE privilege on the table spaces used by the table as well as one of:
    - IMPLICIT\_SCHEMA authority on the database, if the implicit or explicit schema name of the table does not exist
    - CREATEIN privilege on the schema, if the schema name of the table refers to an existing schema.

To drop a table the user must have the following authorities or privileges:

- You must either be the definer as recorded in the DEFINER column of SYSCAT.TABLES, or have at least one of the following privileges:
  - DBADM authority
  - DROPIN privilege on the schema for the table
  - CONTROL privilege on the table

**Note:** The implication of the detach data partition case is that the authorization ID of the statement is going to effectively issue a CREATE TABLE statement and therefore must have the necessary privileges to perform that operation. The table space is the one where the data partition that is being detached already resides. The authorization ID of the ALTER TABLE statement becomes the definer of the new table with CONTROL authority, as if the user issued the CREATE TABLE statement. No privileges from the table being altered are transferred to the new table. Only the authorization ID of the ALTER TABLE statement and DBADM or SYSADM have access to the data immediately after the ALTER TABLE...DETACH PARTITION operation.

### Procedure

To detach a data partition of a partitioned table, issue the ALTER TABLE statement with the DETACH PARTITION clause.

### Example

In the following example, the dec01 data partition is detached from table STOCK and placed in table JUNK. After ensuring that the asynchronous partition detach task made the target table JUNK available, you can drop the table JUNK, effectively dropping the associated data partition.

```
ALTER TABLE stock DETACH PART dec01 INTO junk;
-- After the target table becomes available, issue the DROP TABLE statement
DROP TABLE junk;
```

## What to do next

To make the ALTER TABLE...DETACH as fast as possible with DB2 Version 9.7 Fix Pack 1 and later releases, the asynchronous partition detach task completes the detach operation asynchronously. If there are detached dependent tables, the asynchronous partition detach task does not start and the detached data partition does not become a stand-alone table. In this case, the SET INTEGRITY statement must be issued on all detached dependent tables. After SET INTEGRITY completes, the asynchronous partition detach task starts and makes the target table accessible. When the target table is accessible it can be dropped.

---

## Creating partitioned tables

Partitioned tables use a data organization scheme in which table data is divided across multiple storage objects, called data partitions or ranges, according to values in one or more table partitioning key columns of the table. Data from a given table is partitioned into multiple storage objects based on the specifications provided in the PARTITION BY clause of the CREATE TABLE statement. These storage objects can be in different table spaces, in the same table space, or a combination of both.

### Before you begin

To create a table, the privileges held by the authorization ID of the statement must include at least one of the following authorities or privileges:

- CREATETAB authority on the database and USE privilege on all the table spaces used by the table, as well as one of:
  - IMPLICIT\_SCHEMA authority on the database, if the implicit or explicit schema name of the table does not exist
  - CREATEIN privilege on the schema, if the schema name of the table refers to an existing schema
- DBADM authority

### About this task

You can create a partitioned table by using the CREATE TABLE statement.

### Procedure

To create a partitioned table from the command line, issue the CREATE TABLE statement:

```
CREATE TABLE NAME (column_name data_type null_attribute) IN
 PARTITION BY RANGE (column_expression)
STARTING FROM constant ENDING constant EVERY constant
```

For example, the following statement creates a table where rows with  $a \geq 1$  and  $a \leq 20$  are in PART0 (the first data partition), rows with  $21 \leq a \leq 40$  are in PART1 (the second data partition), up to  $81 \leq a \leq 100$  are in PART4 (the last data partition).

```
CREATE TABLE foo(a INT)
PARTITION BY RANGE (a) (STARTING FROM (1)
ENDING AT (100) EVERY (20))
```

## Defining ranges on partitioned tables

You can specify a range for each data partition when you create a partitioned table. A partitioned table uses a data organization scheme in which table data is divided across multiple data partitions according to the values of the table partitioning key columns of the table.

### About this task

Data from a given table is partitioned into multiple storage objects based on the specifications provided in the PARTITION BY clause of the CREATE TABLE statement. A range is specified by the STARTING FROM and ENDING AT values of the PARTITION BY clause.

To completely define the range for each data partition, you must specify sufficient boundaries. The following is a list of guidelines to consider when defining ranges on a partitioned table:

- The STARTING clause specifies a low boundary for the data partition range. This clause is mandatory for the lowest data partition range (although you can define the boundary as MINVALUE). The lowest data partition range is the data partition with the lowest specified bound.
- The ENDING (or VALUES) clause specifies a high boundary for the data partition range. This clause is mandatory for the highest data partition range (although you can define the boundary as MAXVALUE). The highest data partition range is the data partition with the highest specified bound.
- If you do not specify an ENDING clause for a data partition, then the next greater data partition must specify a STARTING clause. Likewise, if you do not specify a STARTING clause, then the previous data partition must specify an ENDING clause.
- MINVALUE specifies a value that is smaller than any possible value for the column type being used. MINVALUE and INCLUSIVE or EXCLUSIVE cannot be specified together.
- MAXVALUE specifies a value that is larger than any possible value for the column type being used. MAXVALUE and INCLUSIVE or EXCLUSIVE cannot be specified together.
- INCLUSIVE indicates that all values equal to the specified value are to be included in the data partition containing this boundary.
- EXCLUSIVE indicates that all values equal to the specified value are NOT to be included in the data partition containing this boundary.
- The NULLS FIRST and NULLS LAST clauses of the CREATE TABLE statement specify whether null values are to be sorted high or low when considering data partition placement. By default, null values are sorted high. Null values in the table partitioning key columns are treated as positive infinity, and are placed in a range ending at MAXVALUE. If no such data partition is defined, null values are considered to be out-of-range values. Use the NOT NULL constraint if you want to exclude null values from table partitioning key columns. LAST specifies that null values are to appear last in a sorted list of values. FIRST specifies that null values are to appear first in a sorted list of values.
- When using the long form of the syntax, each data partition must have at least one bound specified.

**Tip:** Before you begin defining data partitions on a table it is important to understand how tables benefit from table partitioning and what factors influence the columns you choose as partitioning columns.

The ranges specified for each data partition can be generated automatically or manually.

### Automatically generated

Automatic generation is a simple method of creating many data partitions quickly and easily. This method is appropriate for equal sized ranges based on dates or numbers.

Examples 1 and 2 demonstrate how to use the CREATE TABLE statement to define and generate automatically the ranges specified for each data partition.

#### Example 1:

Issue a create table statement with the following ranges defined:

```
CREATE TABLE lineitem (
 l_orderkey DECIMAL(10,0) NOT NULL,
 l_quantity DECIMAL(12,2),
 l_shipdate DATE,
 l_year_month INT GENERATED ALWAYS AS (YEAR(l_shipdate)*100 + MONTH(l_shipdate)))
 PARTITION BY RANGE(l_shipdate)
 (STARTING ('1/1/1992') ENDING ('12/31/1992') EVERY 1 MONTH);
```

This statement results in 12 data partitions each with 1 key value (*l\_shipdate*) >= ('1/1/1992'), (*l\_shipdate*) < ('3/1/1992'), (*l\_shipdate*) < ('4/1/1992'), (*l\_shipdate*) < ('5/1/1992'), ..., (*l\_shipdate*) < ('12/1/1992'), (*l\_shipdate*) < ('12/31/1992').

The starting value of the first data partition is inclusive because the overall starting bound ('1/1/1992') is inclusive (default). Similarly, the ending bound of the last data partition is inclusive because the overall ending bound ('12/31/1992') is inclusive (default). The remaining STARTING values are inclusive and the remaining ENDING values are all exclusive. Each data partition holds n key values where n is given by the EVERY clause. Use the formula (start + every) to find the end of the range for each data partition. The last data partition might have fewer key values if the EVERY value does not divide evenly into the START and END range.

#### Example 2:

Issue a create table statement with the following ranges defined:

```
CREATE TABLE t(a INT, b INT)
 PARTITION BY RANGE(b) (STARTING FROM (1)
 EXCLUSIVE ENDING AT (1000) EVERY (100))
```

This statement results in 10 data partitions each with 100 key values (1 < b <= 101, 101 < b <= 201, ..., 901 < b <= 1000).

The starting value of the first data partition (b > 1 and b <= 101) is exclusive because the overall starting bound (1) is exclusive. Similarly the ending bound of the last data partition (b > 901 and b <= 1000) is inclusive because the overall ending bound (1000) is inclusive. The remaining STARTING values are all exclusive and the remaining ENDING values are all inclusive. Each data partition holds n key values where n is given by the EVERY clause. Finally, if both the starting and ending bound of the overall clause are exclusive, the starting value of the first data partition is exclusive because the overall starting bound (1) is exclusive. Similarly the ending bound of the last data partition is exclusive because the overall ending bound (1000) is exclusive. The remaining STARTING values are all exclusive and

the ENDING values are all inclusive. Each data partition (except the last) holds n key values where n is given by the EVERY clause.

### Manually generated

Manual generation creates a new data partition for each range listed in the PARTITION BY clause. This form of the syntax allows for greater flexibility when defining ranges thereby increasing your data and LOB placement options.

Examples 3 and 4 demonstrate how to use the CREATE TABLE statement to define and generate manually the ranges specified for a data partition.

#### Example 3:

This statement partitions on two date columns both of which are generated. Notice the use of the automatically generated form of the CREATE TABLE syntax and that only one end of each range is specified. The other end is implied from the adjacent data partition and the use of the INCLUSIVE option:

```
CREATE TABLE sales(invoice_date date, inv_month int NOT NULL
GENERATED ALWAYS AS (month(invoice_date)), inv_year INT NOT
NULL GENERATED ALWAYS AS (year(invoice_date)),
item_id int NOT NULL,
cust_id int NOT NULL) PARTITION BY RANGE (inv_year,
inv_month)
(PART Q1_02 STARTING (2002,1) ENDING (2002, 3) INCLUSIVE,
PART Q2_02 ENDING (2002, 6) INCLUSIVE,
PART Q3_02 ENDING (2002, 9) INCLUSIVE,
PART Q4_02 ENDING (2002,12) INCLUSIVE,
PART CURRENT ENDING (MAXVALUE, MAXVALUE));
```

Gaps in the ranges are permitted. The CREATE TABLE syntax supports gaps by allowing you to specify a STARTING value for a range that does not line up against the ENDING value of the previous data partition.

#### Example 4:

Creates a table with a gap between values 101 and 200.

```
CREATE TABLE foo(a INT
PARTITION BY RANGE(a)
(STARTING FROM (1) ENDING AT (100),
STARTING FROM (201) ENDING AT (300))
```

Use of the ALTER TABLE statement, which allows data partitions to be added or removed, can also cause gaps in the ranges.

When you insert a row into a partitioned table, it is automatically placed into the proper data partition based on its key value and the range it falls within. If it falls outside of any ranges defined for the table, the insert fails and the following error is returned to the application:

```
SQL0327N The row cannot be inserted into table <tablename>
because it is outside the bounds of the defined data partition ranges.
SQLSTATE=22525
```

### Restrictions

- Table level restrictions:
  - Tables created using the automatically generated form of the syntax (containing the EVERY clause) are constrained to use a numeric or date time type in the table partitioning key.

- Statement level restrictions:
  - MINVALUE and MAXVALUE are not supported in the automatically generated form of the syntax.
  - Ranges are ascending.
  - Only one column can be specified in the automatically generated form of the syntax.
  - The increment in the EVERY clause must be greater than zero.
  - The ENDING value must be greater than or equal to the STARTING value.

## Placement of the data, index and long data of a data partition

By its very nature, creating a partitioned table allows you to place the various parts of the table and the associated table objects in specific table spaces.

When creating a table you can specify in which table space the entire table data and associated table objects will be placed. Or, you can place the table's index, long or large data, or table partitions in specific table spaces. All of the table spaces must be in the same database partition group.

The CREATE TABLE statement has the following clauses which demonstrate this ability to place the table data and associated table objects within specific table spaces:

```
CREATE TABLE table_name IN table_space_name1
 INDEX IN table_space_name2
 LONG IN table_space_name3
 PARTITIONED BY ...
 PARTITION partition_name | boundary specification | IN table_space_name4
 INDEX IN table_space_name5
 LONG IN table_space_name6
```

Each of the partitions of the partitioned table can be placed in different table spaces.

You can also specify the table space for a user-created nonpartitioned index on a partitioned table using the CREATE INDEX ... IN *table\_space\_name1* statement, which can be different from the index table space specified in the CREATE TABLE ... INDEX IN *table\_space\_name2* statement. The IN clause of the CREATE INDEX statement is used for partitioned tables only. If the INDEX IN clause is not specified on the CREATE TABLE or CREATE INDEX statements, the index is placed in the same table space as the first visible or attached partition of the table.

System generated nonpartitioned indexes, such as XML column paths indexes, are placed in the table space specified in the INDEX IN clause of the CREATE TABLE statement.

On a partitioned table with XML data, the XML region index is always partitioned in the same way as the table data. The table space of the partitioned indexes is defined at the partition level.

XML data resides in the table spaces used by the long data for a table. XML data placement on a partitioned table follows the long data placement rules.

The table space for long data can be specified explicitly by you or determined by the database manager implicitly. For a partitioned table, the table level LONG IN

clause can be used together with the partition level LONG IN clause. If both are specified, the partition level LONG IN clause takes precedence over any table level LONG IN clauses.

---

## Altering partitioned tables

All relevant clauses of the ALTER TABLE statement are supported for a partitioned table. In addition, the ALTER TABLE statement allows you to add new data partitions, roll-in (*attach*) new data partitions, and roll-out (*detach*) existing data partitions.

### Before you begin

To alter a partitioned table to detach a data partition the user must have the following authorities or privileges:

- The user performing the DETACH PARTITION operation must have the authority necessary to ALTER, to SELECT from, and to DELETE from the source table.
- The user must also have the authority necessary to create the target table. Therefore, to alter a table to detach a data partition, the privilege held by the authorization ID of the statement must include at least one of the following authorities or privileges on the target table:
  - DBADM authority
  - CREATETAB authority on the database and USE privilege on the table spaces used by the table as well as one of:
    - IMPLICIT\_SCHEMA authority on the database, if the implicit or explicit schema name of the table does not exist
    - CREATEIN privilege on the schema, if the schema name of the table refers to an existing schema.

To alter a partitioned table to attach a data partition, the privileges held by the authorization ID of the statement must include at least one of the following authorities or privileges on the source table:

- DATAACCESS authority or SELECT privilege on the source table and DBADM authority or DROPIN privilege on the schema of the source table
- CONTROL privilege on the source table

To alter a partitioned table to add a data partition, the privileges held by the authorization ID of the statement must have privileges to use the table space where the new partition is added, and include at least one of the following authorities or privileges on the source table:

- ALTER privilege
- CONTROL privilege
- DBADM
- ALTERIN privilege on the table schema

### About this task

- Each ALTER TABLE statement issued with the PARTITION clause must be in a separate SQL statement.
- No other ALTER operations are permitted in an SQL statement containing an ALTER TABLE ... PARTITION operation. For example, you cannot attach a data partition and add a column to the table in a single SQL statement.

- Multiple ALTER statements can be executed, followed by a single SET INTEGRITY statement.

## **Procedure**

To alter a partitioned table from the command line, issue the ALTER TABLE statement.

## **Guidelines and restrictions on altering partitioned tables**

This topic identifies the most common alter table actions and special considerations in the presence of attached and detached data partitions.

The STATUS column of the SYSCAT.DATAPARTITIONS catalog view contains the state information for the partitions of a table.

- If the STATUS is the empty string, the partition is visible and is in the normal state.
- If the STATUS is 'A', the partition is newly attached and the SET INTEGRITY statement must be issued to bring the attached partition into the normal state.
- If the STATUS is 'D', 'L', or 'T', the partition is being detached, but the detach operation has not completed.
  - For a partition in the 'D' state, the SET INTEGRITY statement must be issued on all detached dependent tables in order to transition the partition to the logically detached state.
  - For a partition in the 'L' state, the partition is a logically detached partition and the asynchronous partition detach task is completing the detach of the partition for DB2 Version 9.7 Fix Pack 1 and later releases.
  - For a partition in the 'T' state the asynchronous partition detach task has completed and asynchronous index cleanup is updating nonpartitioned indexes defined on the partition.

### **Adding or attaching a data partition - locking behavior**

For information about locking behavior during an add partition or attach partition operation, see the “Scenarios: Rolling in and rolling out partitioned table data” topic.

### **Adding or altering a constraint**

Adding a check or a foreign key constraint is supported with attached and detached data partitions. When a partitioned table has detached partitions in state 'D' or 'L', adding a primary or unique constraint will return an error if the system has to generate a new partitioned index to enforce the constraint. For a partition in the 'L' state, the operation returns SQL20285N (SQLSTATE 55057). For a partition in the 'D' state, the operation returns SQL20054 (SQLSTATE 55019).

### **Adding a column**

When adding a column to a table with attached data partitions, the column is also added to the attached data partitions. When adding a column to a table with detached data partitions in the 'T' state, the column is not added to the detached data partitions because the detached data partitions are no longer physically associated to the table.

For a detached partition in the 'L' or 'D' state, the operation fails and an error is returned. For a partition in the 'L' state, the operation returns SQL20285N (SQLSTATE 55057). For a partition in the 'D' state, the operation returns SQL20296N (SQLSTATE 55057).

### **Altering a column**

When altering a column in a table with attached data partitions, the column will also be altered on the attached data partitions. When altering a column in a table with detached data partitions, the column is not altered on the detached data partitions, because the detached data partitions are no longer physically associated to the table.

When dropping or renaming a column when a partition is detached in the 'L' or 'D' state the operation fails and an error is returned. For a partition in the 'L' state, the operation returns SQL20285N (SQLSTATE 55057). For a partition in the 'D' state, the operation returns SQL0270N (SQLSTATE 42997).

### **Adding a generated column**

When adding a generated column to a partitioned table with attached or detached data partitions, it must respect the rules for adding any other types of columns.

### **Adding or modifying a nonpartitioned index**

When creating, recreating, or reorganizing an index on a table with attached data partitions, the index does not include the data in the attached data partitions because the SET INTEGRITY statement maintains all indexes for all attached data partitions. When creating, recreating or reorganizing an index on a table with detached data partitions, the index does not include the data in the detached data partitions, unless the detached data partition has a detached dependent table or staging tables that need to be incrementally refreshed with respect to the data partition, the partition is in the 'D' state. In this case, the index includes the data for this detached data partition.

### **Adding or modifying a partitioned index**

When creating a partitioned index in the presence of attached data partitions, an index partition for each attached data partition will be created. The index entries for index partitions on attached data partitions will not be visible until the SET INTEGRITY statement is run to bring the attached data partitions online. Note that because create index includes the attached data partitions, creation of a unique partitioned index may find rows in the attached data partition which are duplicate key values and thus fail the index creation. It is recommended that users do not attempt to create partitioned indexes in the presence of attached partitions to avoid this problem.

If the table has any detached dependent tables, creation of partitioned indexes is not supported on partitioned tables with detached dependent tables. Any attempt to create a partitioned index in this situation will result in SQLSTATE 55019. When creating a partitioned index on a table that has partitions in 'L' state, the operation returns SQL20285N (SQLSTATE 55057).

### **WITH EMPTY TABLE**

You cannot empty a table with attached data partitions.

### **ADD MATERIALIZED QUERY AS**

Altering a table with attached data partitions to an MQT is not allowed.

### **Altering additional table attributes that are stored in a data partition**

The following table attributes are also stored in a data partition. Changes to these attributes are reflected on the attached data partitions, but not on the detached data partitions.

- DATA CAPTURE

- VALUE COMPRESSION
- APPEND
- COMPACT/LOGGED FOR LOB COLUMNS

#### **Creating and accessing data partitions within the same transaction**

If a table has a nonpartitioned index, you cannot access a new data partition in that table within the same transaction as the add or attach operation that created the partition, if the transaction does not have the table locked in exclusive mode (SQL0668N, reason code 11).

### **Special considerations for XML indexes when altering a table to ADD, ATTACH, or DETACH a partition**

Similar to a nonpartitioned relational index, a nonpartitioned index over an XML column is an independent object that is shared among all data partitions of a partitioned table. XML region indexes and column path indexes are affected when you alter a table by adding, attaching, or detaching a partition. Indexes over XML column paths are always nonpartitioned, and indexes over XML data are generated as partitioned by default.

#### **XML regions index**

ADD PARTITION will create a new regions index partition for the new empty data partition being added. A new entry for the regions index partition will be added to the SYSINDEXPARTITIONS table. The table space for the partitioned index object on the new partition will be determined by the INDEX IN <table space> in the ADD PARTITION clause. If no INDEX IN <table space> is specified for the ADD PARTITION clause, the table space for the partitioned index object will be the same as the table space used by the corresponding data partition by default.

The system-generated XML regions index on a partitioned table is always partitioned. A partitioned index uses an index organization scheme in which index data is divided across multiple storage objects, called index partitions, according to the table partitioning scheme of the table. Each index partition only refers to table rows in the corresponding data partition.

For ATTACH, since the regions index on a partitioned table with XML column is always partitioned, the region index on the source table can be kept as the new regions index partition for the new table partition after completing the ATTACH operation. Data and index objects do not move, therefore the catalog table entries need to be updated. The catalog table entry for the regions index on the source table will be removed on ATTACH and one regions index partition will be added in the SYSINDEXPARTITIONS table. The pool ID and object ID will remain the same as they were on the source table. The index ID (IID) will be modified to match that of the regions index on the target.

After completing the DETACH operation, the regions index will be kept on the detached table. The index partition entry associated to the partition being detached will be removed from the SYSINDEXPARTITIONS table. One new regions index entry will be added in the SYSINDEXES catalog table for the detached table, which will have the same pool ID and object ID as the region index partition before the DETACH.

## **Index over XML data**

Starting in DB2 Version 9.7 Fix Pack 1, you can create an index over XML data on a partitioned table as either partitioned or nonpartitioned. The default is a partitioned index.

Partitioned and nonpartitioned indexes over XML data are treated like any other relational indexes during ATTACH and DETACH operations.

Indexes on the source table will be dropped during the ATTACH operation. This applies to both the logical and physical XML indexes. Their entries in the system catalogs will be removed during the ATTACH operation.

Set integrity must be run after ATTACH, to maintain the nonpartitioned indexes over XML data on the target table.

For DETACH, nonpartitioned indexes over XML columns on the source table are not inherited by the target table.

## **XML column path indexes**

Indexes over XML column paths are always nonpartitioned indexes. The XML column path indexes on the source and target tables are maintained during roll-in and rollout operations.

For ATTACH, the DB2 database manager will maintain the nonpartitioned XML column path indexes on the target table (this is unlike other nonpartitioned indexes, which are maintained during SET INTEGRITY after completing the ATTACH operation). Afterwards, the XML column path indexes on the source table will be dropped and their catalog entries will be removed because the column path indexes on the target table are nonpartitioned.

For rollout, recall that the XML column path indexes are nonpartitioned, and nonpartitioned indexes are not carried along to the standalone target table. However, XML column path indexes (one for each column) must exist on a table with XML columns before the table can be accessible to external user, therefore XML column path indexes must be created on the target table before it can be used. The time at which the column path indexes will be created depends on whether there are any detached dependent tables during the DETACH operation. If there are no detached dependent tables, then the paths indexes will be created during the DETACH operation, otherwise they will be created by SET INTEGRITY or MQT refresh to maintain the detach dependent objects.

After DETACH, the XML column path indexes created on the target table will reside in the same index object along with all other indexes on that table.

## **Scenario: Rotating data in a partitioned table**

Rotating data in DB2 databases refers to a method of reusing space in a data partition by removing obsolete data from a table (a detach partition operation) and then adding new data (an attach partition operation).

### **Before you begin**

Alternatively, you can archive the detached partition and load the new data into a different source table before an attach operation is performed. In the following

scenario, a detach operation precedes the other steps; it could as easily be the last step, depending on your specific requirements.

To alter a table to detach a data partition, the authorization ID of the statement must hold the following privileges and authorities:

- At least one of the following authorities on the target table of the detached partition:
  - CREATETAB authority on the database, and USE privilege on the table spaces used by the table, as well as one of the following authorities or privileges:
    - IMPLICIT\_SCHEMA authority on the database, if the implicit or explicit schema name of the new table does not exist
    - CREATEIN privilege on the schema, if the schema name of the new table refers to an existing schema
  - DBADM authority
- At least one of the following privileges and authorities on the source table:
  - SELECT, ALTER, and DELETE privileges on the table
  - CONTROL privilege on the table
  - DATAACCESS authority

To alter a table to attach a data partition, the authorization ID of the statement must include the following privileges and authorities:

- At least one of the following authorities or privileges on the source table:
  - SELECT privilege on the table and DROPIN privilege on the schema of the table
  - CONTROL privilege on the table
  - DATAACCESS authority
- A least one of the following authorities or privileges on the target table:
  - ALTER and INSERT privileges on the table
  - CONTROL privilege on the table
  - DATAACCESS authority

## Procedure

To rotate data in a partitioned table, issue the ALTER TABLE statement. The following example shows how to update the STOCK table by removing the data from December 2008 and replacing it with the latest data from December 2010.

1. Remove the old data from the STOCK table.

```
ALTER TABLE stock DETACH PARTITION dec08 INTO newtable;
```

2. Load the new data. Using the LOAD command with the REPLACE option overwrites existing data.

```
LOAD FROM data_file OF DEL REPLACE INTO newtable
```

**Note:** If there are detached dependents, issue the SET INTEGRITY statement on the detached dependents before loading the detached table. If SQL20285N is returned, wait until the asynchronous partition detach task is complete before issuing the SET INTEGRITY statement again.

3. If necessary, perform data cleansing activities, which can include the following actions:

- Filling in missing values
- Deleting inconsistent and incomplete data

- Removing redundant data arriving from multiple sources
  - Transforming data
    - *Normalization.* Data from different sources that represents the same value in different ways must be reconciled as part of rolling the data into the warehouse.
    - *Aggregation.* Raw data that is too detailed to store in the warehouse must be aggregated before being rolled in.
4. Attach the data as a new range.
- ```
ALTER TABLE stock
  ATTACH PARTITION dec10
    STARTING '12/01/2008' ENDING '12/31/2010'
  FROM newtable;
```
5. Use the SET INTEGRITY statement to update indexes and other dependent objects. Read and write access is permitted during execution of the SET INTEGRITY statement.
- ```
SET INTEGRITY FOR stock
 ALLOW WRITE ACCESS
 IMMEDIATE CHECKED
 FOR EXCEPTION IN stock USE stock_ex;
```

## Scenarios: Rolling in and rolling out partitioned table data

A common administrative operation in data warehouses is to periodically roll in new data and roll out obsolete data. The following scenarios illustrate these tasks.

### Scenario 1: Rolling out obsolete data by detaching a data partition

The following example shows how to detach an unneeded data partition (DEC01) from a partitioned table named STOCK. The detached data partition is used to create a table named STOCK\_DROP without any data movement.

```
ALTER TABLE stock DETACH PART dec01 INTO stock_drop;
COMMIT WORK;
```

To expedite the detach operation, index cleanup on the source table is done automatically and in the background through an asynchronous index cleanup process. If there are no detached dependent tables defined on the source table, there is no need to issue a SET INTEGRITY statement to complete the detach operation.

The new table can be dropped or attached to another table, or it can be truncated and loaded with new data before being reattached to the source table. You can perform these operations immediately, even before asynchronous index cleanup completes, unless the source table detached dependent tables.

To determine whether a detached table is accessible, query the SYSCAT.TABDETACHEDDEP catalog view. If a detached table is found to be inaccessible, issue the SET INTEGRITY statement with the IMMEDIATE CHECKED option against all of the detached dependent tables. If you try to access a detached table before all of its detached dependent tables are maintained, an error (SQL20285N) is returned.

## Scenario 2: Creating a new, empty range

The following example shows how to add an empty data partition (DEC02) to a partitioned table named STOCK. The STARTING FROM and ENDING AT clauses specify the range of values for the new data partition.

```
ALTER TABLE stock ADD PARTITION dec02
STARTING FROM '12/01/2002' ENDING AT '12/31/2002';
```

This ALTER TABLE...ADD PARTITION statement drains existing static or repeatable-read queries that are running against the STOCK table and invalidates packages on the table; that is, the statement allows such queries to complete normally before it exclusively locks the table (by using a Z lock) and performs the add operation. Existing dynamic non-repeatable-read queries against the STOCK table continue, and can run concurrently with the add operation. Any new queries attempting to access the STOCK table after the add operation starts must wait until the transaction in which the statement is issued commits. The STOCK table is Z-locked (completely inaccessible) during this period.

**Tip:** Issue a COMMIT statement immediately after the add operation to make the table available for use sooner.

Load data into the table:

```
LOAD FROM data_file OF DEL
INSERT INTO stock
ALLOW READ ACCESS;
```

Issue a SET INTEGRITY statement to validate constraints and refresh dependent materialized query tables (MQTs). Any rows that violate defined constraints are moved to the exception table STOCK\_EX.

```
SET INTEGRITY FOR stock
ALLOW READ ACCESS
IMMEDIATE CHECKED
FOR EXCEPTION IN stock USE stock_ex;

COMMIT WORK;
```

## Scenario 3: Rolling in new data by attaching a loaded data partition

The following example shows how an attach operation can be used to facilitate loading a new range of data into an existing partitioned table (the target table named STOCK). Data is loaded into a new, empty table (DEC03), where it can be checked and cleansed, if necessary, without impacting the target table. Data cleansing activities include:

- Filling in missing values
- Deleting inconsistent and incomplete data
- Removing redundant data that arrived from multiple sources
- Transforming the data through normalization or aggregation:
  - *Normalization*. Data from different sources that represents the same values in different ways must be reconciled as part of the roll-in process.
  - *Aggregation*. Raw data that is too detailed to store in a warehouse must be aggregated during roll-in.

After the data is prepared in this way, the newly loaded data partition can be attached to the target table.

```

CREATE TABLE dec03(...);
LOAD FROM data_file OF DEL REPLACE INTO dec03;
(data cleansing, if necessary)
ALTER TABLE stock ATTACH PARTITION dec03
STARTING FROM '12/01/2003' ENDING AT '12/31/2003'
FROM dec03;

```

During an attach operation, one or both of the STARTING FROM and ENDING AT clauses must be specified, and the lower bound (STARTING FROM clause) must be less than or equal to the upper bound (ENDING AT clause). The newly attached data partition must not overlap an existing data partition range in the target table. If the high end of the highest existing range is defined as MAXVALUE, any attempt to attach a new high range fails, because that new range would overlap the existing high range. A similar restriction applies to low ranges that end at MINVALUE. Moreover, you cannot add or attach a new data partition in the middle, unless its new range falls within a gap in the existing ranges. If boundaries are not specified by the user, they are determined when the table is created.

This ALTER TABLE...ATTACH PARTITION statement drains existing static or repeatable-read queries that are running against the STOCK table and invalidates packages on the table; that is, the statement allows such queries to complete normally before it exclusively locks the table (by using a Z lock) and performs the attach operation. Existing dynamic non-repeatable-read queries against the STOCK table continue, and can run concurrently with the attach operation. Any new queries attempting to access the STOCK table after the attach operation starts must wait until the transaction in which the statement is issued commits. The STOCK table is Z-locked (completely inaccessible) during this period.

**Tip:**

- Issue a COMMIT statement immediately after the attach operation to make the table available for use.
- Issue a SET INTEGRITY statement immediately after the attach operation commits to make the data from the new data partition available sooner.

The data in the attached data partition is not yet visible because it is not yet validated by the SET INTEGRITY statement. The SET INTEGRITY statement is necessary to verify that the newly attached data is within the defined range. It also performs any necessary maintenance activities on indexes and other dependent objects, such as MQTs. New data is not visible until the SET INTEGRITY statement commits; however, if the SET INTEGRITY statement is running online, existing data in the STOCK table is fully accessible for both read and write operations.

**Tip:** If data integrity checking, including range validation and other constraints checking, can be done through application logic that is independent of the data server before an attach operation, newly attached data can be made available for use much sooner. You can optimize the data roll-in process by using the SET INTEGRITY...ALL IMMEDIATE UNCHECKED statement to skip range and constraints violation checking. In this case, the table is brought out of SET INTEGRITY pending state, and the new data is available for applications to use immediately, as long as there are no nonpartitioned user indexes on the target table.

**Note:** You cannot execute data definition language (DDL) statements or utility operations against the table while the SET INTEGRITY statement is running. These operations include, but are not restricted to, the following statements and commands:

- LOAD command
- REDISTRIBUTE DATABASE PARTITION GROUP command
- REORG INDEXES/TABLE command
- ALTER TABLE statement
  - ADD COLUMN
  - ADD PARTITION
  - ATTACH PARTITION
  - DETACH PARTITION
- CREATE INDEX statement

The SET INTEGRITY statement validates the data in the newly attached data partition:

```
SET INTEGRITY FOR stock
 ALLOW WRITE ACCESS
 IMMEDIATE CHECKED
 FOR EXCEPTION IN stock USE stock_ex;
```

Committing the transaction makes the table available for use:

```
COMMIT WORK;
```

Any rows that are out of range, or that violate other constraints, are moved to the exception table STOCK\_EX. You can query this table, fix the rows, and insert them into the STOCK table.

## Migrating existing tables and views to partitioned tables

You can migrate a nonpartitioned table or a UNION ALL view to an empty partitioned table.

### Before you begin

Attaching a data partition is not allowed if SYSCAT.COLUMNS.IMPLICITVALUE for a specific column is a nonnull value for both the source column and the target column, and the values do not match. In this case, you must drop the source table and then recreate it.

A column can have a nonnull value in the SYSCAT.COLUMNS IMPLICITVALUE field if any one of the following conditions is met:

- The IMPLICITVALUE field is propagated from a source table during an attach operation.
- The IMPLICITVALUE field is inherited from a source table during a detach operation.
- The IMPLICITVALUE field is set during migration from V8 to V9, where it is determined to be an added column, or might be an added column. An added column is a column that is created as the result of an ALTER TABLE...ADD COLUMN statement.

Always create the source and target tables involved in an attach operation with the same columns defined. In particular, never use the ALTER TABLE statement to add columns to the target table of an attach operation.

For advice on avoiding a mismatch when working with partitioned tables, see “Guidelines for attaching data partitions to partitioned tables” on page 240.

## About this task

When migrating regular tables, unload the source table by using the **EXPORT** command or high performance unload. Create a new, empty partitioned table, and use the **LOAD** command to populate that partitioned table. To move the data from the old table directly into the partitioned table without any intermediate steps, use the **LOAD FROM CURSOR** command (see Step 1).

You can convert nonpartitioned data in a UNION ALL view to a partitioned table (see Step 2). UNION ALL views are used to manage large tables and achieve easy roll-in and roll-out of table data while providing the performance advantages of branch elimination. Using the ALTER TABLE...ATTACH PARTITION statement, you can achieve conversion with no movement of data in the base table.

Nonpartitioned indexes and dependent views or materialized query tables (MQTs) must be recreated after the conversion. The recommended strategy to convert UNION ALL views to partitioned tables is to create a partitioned table with a single dummy data partition, then attach all of the tables of the union all view. Be sure to drop the dummy data partition early in the process to avoid problems with overlapping ranges.

## Procedure

1. Migrate a regular table to a partitioned table. Use the **LOAD FROM CURSOR** command to avoid any intermediate steps. The following example shows how to migrate table T1 to the SALES\_DP table.

- a. Create and populate a regular table T1.

```
CREATE TABLE t1 (c1 int, c2 int);
INSERT INTO t1 VALUES (0,1), (4, 2), (6, 3);
```

- b. Create an empty partitioned table.

```
CREATE TABLE sales_dp (c1 int, c2 int)
PARTITION BY RANGE (c1)
(STARTING FROM 0 ENDING AT 10 EVERY 2);
```

- c. Use the **LOAD FROM CURSOR** command to pull the data from an SQL query directly into the new partitioned table.

```
SELECT * FROM t1;
DECLARE c1 CURSOR FOR SELECT * FROM t1;
LOAD FROM c1 of CURSOR INSERT INTO sales_dp;SELECT * FROM sales_dp;
```

2. Convert nonpartitioned data in a UNION ALL view to a partitioned table. The following example shows how to convert the UNION ALL view named ALL\_SALES to the SALES\_DP table.

- a. Create the UNION ALL view.

```
CREATE VIEW all_sales AS
(
 SELECT * FROM sales_0198
 WHERE sales_date BETWEEN '01-01-1998' AND '01-31-1998'
 UNION ALL
 SELECT * FROM sales_0298
 WHERE sales_date BETWEEN '02-01-1998' AND '02-28-1998'
 UNION ALL
 ...
 UNION ALL
 SELECT * FROM sales_1200
 WHERE sales_date BETWEEN '12-01-2000' AND '12-31-2000'
);
```

- b. Create a partitioned table with a single dummy partition. Choose the range so that it does not overlap with the first data partition to be attached.

- ```

CREATE TABLE sales_dp (
    sales_date DATE NOT NULL,
    prod_id INTEGER,
    city_id INTEGER,
    channel_id INTEGER,
    revenue DECIMAL(20,2))
PARTITION BY RANGE (sales_date)
(PART dummy STARTING FROM '01-01-1900' ENDING AT '01-01-1900');

c. Attach the first table.

ALTER TABLE sales_dp ATTACH PARTITION
    STARTING FROM '01-01-1998' ENDING AT '01-31-1998'
    FROM sales_0198;

d. Drop the dummy partition.

ALTER TABLE sales_dp DETACH PARTITION dummy
    INTO dummy;
DROP TABLE dummy;

e. Attach the remaining partitions.

ALTER TABLE sales_dp ATTACH PARTITION
    STARTING FROM '02-01-1998' ENDING AT '02-28-1998'
    FROM sales_0298;
...
ALTER TABLE sales_dp ATTACH PARTITION
    STARTING FROM '12-01-2000' ENDING AT '12-31-2000'
    FROM sales_1200;

f. Issue the SET INTEGRITY statement to make data in the newly attached
partition accessible to queries.

SET INTEGRITY FOR sales_dp IMMEDIATE CHECKED
FOR EXCEPTION IN sales_dp USE sales_ex;

Tip: If data integrity checking, including range validation and other
constraints checking, can be done through application logic that is
independent of the data server before an attach operation, newly attached
data can be made available for use much sooner. You can optimize the data
roll-in process by using the SET INTEGRITY...ALL IMMEDIATE
UNCHECKED statement to skip range and constraints violation checking. In
this case, the table is brought out of SET INTEGRITY pending state, and the
new data is available for applications to use immediately, as long as there
are no nonpartitioned user indexes on the target table.

g. Create indexes, as appropriate.

```

Converting existing indexes to partitioned indexes

System-created and user-created indexes might need to be migrated from nonpartitioned to partitioned. User-created indexes can be converted while maintaining availability to the table and indexes for most of the migration. System-created indexes used to enforce primary key constraints or unique constraints will not be able to have the constraints maintained while the conversion is done.

Before you begin

Indexes created in an earlier release of the product might be nonpartitioned. This could include both indexes created by you, or system-created indexes created by the database manager. Examples of system-created indexes are indexes to enforce unique and primary constraints and the block indexes of an MDC table.

About this task

Indexes created by you can be converted from nonpartitioned to partitioned while having continuous availability to the data using the index. You can create a partitioned index with the same keys as the corresponding nonpartitioned index. While the partitioning index is created, you can still use the current indexes and the table where the index is being created. Once the partitioned index is created, you can drop the corresponding nonpartitioned index and rename the new partitioned index if desired.

Results

The following examples demonstrate how to convert existing nonpartitioned indexes into partitioned indexes.

Example

Here is an example of converting a nonpartitioned index created by you to one that is a partitioned index:

```
UPDATE COMMAND OPTIONS USING C OFF;
CREATE INDEX data_part ON sales(sale_date) PARTITIONED;
DROP INDEX dateidx;
RENAME INDEX data_part TO dateidx;
COMMIT;
```

Here is an example of converting a nonpartitioned index created by the database manager to one that is a partitioned index. In this case, there will be a period of time between the dropping of the original constraint, and the creation of the new constraint.

```
ALTER TABLE employees DROP CONSTRAINT emp_uniq;
ALTER TABLE employees ADD CONSTRAINT emp_uniq UNIQUE (employee_id);
```

MDC tables created using DB2 Version 9.7 and earlier releases have nonpartitioned block indexes. To take advantage of partitioned table data availability features such as data roll in and roll out and partition level reorganization of table data and indexes, the data in the multidimensional clustering (MDC) table created using DB2 V9.7 and earlier releases must be moved to a partitioned MDC table with partitioned block indexes created using DB2 V9.7 Fix Pack 1 or a later release.

Online move of a partitioned MDC table to use partitioned block indexes

You can move data from a MDC table with nonpartitioned block indexes to an MDC table with partitioned block indexes using an online table move.

In the following example, company1.parts table has **region** and **color** as the MDC key columns; and the corresponding block indexes are nonpartitioned.

```
CALL SYSPROC.ADMIN_MOVE_TABLE(
  'COMPANY1',    --Table schema
  'PARTS',       --Table name
  ' ',           --null; No change to columns definition
  ' ',           --null; No additional options
  'MOVE');        --Move the table in one step
```

Offline move of a partitioned MDC table to use partitioned block indexes

To minimize data movement, you can move data from a MDC table with nonpartitioned block indexes to an MDC table with partitioned block indexes when the table is offline. The process uses the following steps:

1. Create a new, single-partition MDC table with the same definition as the table to be converted. When specifying the range for the partition, use a range outside the ranges of the partitioned MDC table to be converted.

The block indexes of new, single-partition MDC table are partitioned. The partition created when specifying the range is detached in a later step.

2. Detach each partition of the MDC table. Each partition becomes a stand-alone MDC table.

When a partition is detached, the partition data is attached to a new, target table without moving the data in the partition.

Note: The last partition of the MDC table cannot be detached. It is a single-partition MDC table with nonpartitioned block indexes.

3. For each stand-alone table created by detaching the MDC table partitions, and the single-partition MDC table with nonpartitioned block indexes, attach the table to the new partitioned MDC table created in Step 1.

When the table is attached, the table data is attached to the new partitioned MDC table without moving the data, and the block indexes are created as partitioned block indexes.

4. After attaching the first stand-alone MDC table, you can detach the empty partition created when you created the new MDC table.
5. Issue SET INTEGRITY statement on the new partitioned MDC table.

What to do next

Chapter 10. Range-clustered tables

A range-clustered table (RCT) has a table layout scheme in which each record in the table has a predetermined record ID (RID). The RID is an internal identifier that is used to locate a record in the table.

An algorithm is used to associate a record key value with the location of a specific table row. This approach provides exceptionally fast access to specific table rows. The algorithm does not use hashing, because hashing does not preserve key-value order. Preserving this order eliminates the need to reorganize the table data over time.

Each record key value in the table must be:

- Unique
- Not null
- An integer (SMALLINT, INTEGER, or BIGINT)
- Monotonically increasing
- Within a predetermined set of ranges based on each column in the key. (If necessary, use the ALLOW OVERFLOW option on the CREATE TABLE statement to allow rows with key values that are outside of the defined range of values.)

In addition to direct access to specific table rows, there are other advantages to using range-clustered tables.

- Less maintenance is required. A secondary structure, such as a B+ tree index, which would need to be updated after every insert, update, or delete operation, does not exist.
- Less logging is required for RCTs, when compared to similarly-sized regular tables with B+ tree indexes.
- Less buffer pool memory is required. There is no additional memory required to store a secondary structure, such as a B+ tree index.

Space for an RCT is pre-allocated and reserved for use by the table even when records do not yet exist. Consequently, range-clustered tables have no need for free space control records (FSCR). At table creation time, there are no records in the table; however, the entire range of pages is pre-allocated. Preallocation is based on the record size and the maximum number of records to be stored. If a variable-length field (such as VARCHAR) is defined, the maximum length of the field is used, and the overall record size is of fixed length. This can result in less than optimal use of space. If key values are sparse, the unused space has a negative impact on range scan performance. Range scans must visit all possible rows within a range, even rows that do not yet contain data.

If a schema modification on a range-clustered table is required, the table must be recreated with a new schema name and then populated with the data from the old table. For example, if a table's ranges need to be altered, create a table with new ranges and populate it with data from the old table.

If an RCT allows overflow records, and a new record has a key value that falls outside of the defined range of values, the record is placed in an overflow area, which is dynamically allocated. As more records are added to this overflow area,

operations against the table that involve the overflow area require more processing time. The larger the overflow area, the more time is required to access it. If this becomes a problem, consider reducing the size of the overflow area by exporting the data to a new RCT with wider ranges.

Guidelines for using range-clustered tables

This topic lists some guidelines to follow when working with range-clustered tables (RCT).

- Because the process of creating a range-clustered table pre-allocates the required disk space, that space must be available.
- When defining the range of key values, the minimum value is optional; if it is not specified, the default is 1. A negative minimum value must be specified explicitly. For example:
ORGANIZE BY KEY SEQUENCE (f1 STARTING FROM -100 ENDING AT -10)
- You cannot create a regular index on the same key values that are used to define the range-clustered table.
- ALTER TABLE statement options that affect the physical structure of the table are not allowed.

Scenarios: Range-clustered tables

Range-clustered tables can have single-column or multiple-column keys, and can allow or disallow rows with key values that are outside of the defined range of values. This section contains scenarios that illustrate how such tables can be created.

Scenario 1: Creating a range-clustered table (overflow allowed)

The following example shows a range-clustered table that can be used to retrieve information about a specific student. Each student record contains the following information:

- School ID
- Program ID
- Student number
- Student ID
- Student first name
- Student last name
- Student grade point average (GPA)

```
CREATE TABLE students (
    school_id      INT NOT NULL,
    program_id     INT NOT NULL,
    student_num    INT NOT NULL,
    student_id     INT NOT NULL,
    first_name     CHAR(30),
    last_name      CHAR(30),
    gpa            FLOAT
)
ORGANIZE BY KEY SEQUENCE
(student_id STARTING FROM 1 ENDING AT 1000000)
ALLOW OVERFLOW
;
```

In this example, the STUDENT_ID column, which serves as the table key, is used to add, update, or delete student records.

The size of each record is based on the sum of the column lengths. In this example, each record is 97 bytes long (10-byte header + 4 + 4 + 4 + 4 + 30 + 30 + 8 + 3 bytes for nullable columns). With a 4-KB (or 4096-byte) page size, after accounting for overhead, there are 4038 bytes (enough for 41 records) available per page. A total of 24391 such pages is needed to accommodate 1 million student records.

Assuming four pages for table overhead and three pages for extent mapping, 24384 4-KB pages would be pre-allocated when this table is created. (The extent mapping assumes a single three-page container for the table.)

Scenario 2: Creating a range-clustered table (overflow not allowed)

In the following example, a school board administers 200 schools, each having 20 classrooms with a capacity of 35 students per classroom. This school board can accommodate a maximum of 140,000 students.

```
CREATE TABLE students (
    school_id      INT NOT NULL,
    class_id       INT NOT NULL,
    student_num    INT NOT NULL,
    student_id     INT NOT NULL,
    first_name     CHAR(30),
    last_name      CHAR(30),
    gpa            FLOAT
)
ORGANIZE BY KEY SEQUENCE
(school_id STARTING FROM 1 ENDING AT 200,
 class_id STARTING FROM 1 ENDING AT 20,
 student_num STARTING FROM 1 ENDING AT 35)
DISALLOW OVERFLOW
;
```

In this example, the SCHOOL_ID, CLASS_ID, and STUDENT_NUM columns together serve as the table key, which is used to add, update, or delete student records.

Overflow is not allowed, because school board policy restricts the number of students in each classroom, and there is a fixed number of schools and classrooms being administered by this school board. Some smaller schools (schools with fewer classrooms than the largest school) will have pre-allocated space in the table that will likely never be used.

Restrictions on range-clustered tables

There are contexts in which range-clustered tables cannot be used, and there are certain utilities that cannot operate on range-clustered tables.

The following restrictions apply to range-clustered tables:

- Range-clustered tables cannot be specified in a DB2 pureScale environment (SQLSTATE 42997).
- Partitioned tables cannot be range-clustered tables.
- Declared temporary tables and created temporary tables cannot be range-clustered tables.
- Automatic summary tables (AST) cannot be range-clustered tables.
- The load utility is not supported. Data can be inserted into a range-clustered table through the import utility or through a parallel insert application.
- The **REORG** utility is not supported. Range-clustered tables that are defined with the DISALLOW OVERFLOW option do not need to be reorganized.

Range-clustered tables that are defined with the ALLOW OVERFLOW option cannot have the data in this overflow region reorganized.

- The DISALLOW OVERFLOW clause on the CREATE TABLE statement cannot be specified if the table is a range-clustered materialized query table.
- The design advisor will not recommend range-clustered tables.
- Multidimensional clustering and clustering indexes are incompatible with range-clustered tables.
- Value and default compression are not supported.
- Reverse scans on range-clustered tables are not supported.
- The **REPLACE** parameter on the **IMPORT** command is not supported.
- The WITH EMPTY TABLE option on the ALTER TABLE...ACTIVATE NOT LOGGED INITIALLY statement is not supported.

Part 3. High Availability and Diagnostics

The availability of a database solution is a measure of how successful user applications are at performing their required database tasks.

If user applications cannot connect to the database, or if their transactions fail because of errors or time out because of load on the system, the database solution is not very available. If user applications are successfully connecting to the database and performing their work, the database solution is highly available.

Designing a highly available database solution, or increasing the availability of an existing solution requires an understanding of the needs of the applications accessing the database. To get the greatest benefit from the expense of additional storage space, faster processors, or more software licenses, focus on making your database solution as available as required to the most important applications for your business at the time when those applications need it most.

Unplanned outages

Unexpected system failures that could affect the availability of your database solution to users include: power interruption; network outage; hardware failure; operating system or other software errors; and complete system failure in the event of a disaster. If such a failure occurs at a time when users expect to be able to do work with the database, a highly available database solution must do the following:

- Shield user applications from the failure, so the user applications are not aware of the failure. For example, DB2 Data Server can reroute database client connections to alternate database servers if a database server fails.
- Respond to the failure to contain its effect. For example, if a failure occurs on one machine in a cluster, the cluster manager can remove that machine from the cluster so that no further transactions are routed to be processed on the failed machine.
- Recover from the failure to return the system to normal operations. For example, if standby database takes over database operations for a failed primary database, the failed database might restart, recover, and take over once again as the primary database.

These three tasks must be accomplished with a minimum effect on the availability of the solution to user applications.

Planned outage

In a highly available database solution, the impact of maintenance activities on the availability of the database to user applications must be minimized as well.

For example, if the database solution serves a traditional store front that is open for business between the hours of 9am to 5pm, then maintenance activities can occur offline, outside of those business hours without affecting the availability of the database for user applications. If the database solution serves an online banking business that is expected to be available for customers to access through the Internet 24 hours per day, then maintenance activities must be run online, or scheduled for off-peak activity periods to have minimal impact on the availability of the database to the customers.

When you are making business decisions and design choices about the availability of your database solution, you must weigh the following two factors:

- The cost to your business of the database being unavailable to customers
- The cost of implementing a certain degree of availability

For example, consider an Internet-based business that makes a certain amount of revenue, X , every hour the database solution is serving customers. A high availability strategy that saves 10 hours of downtime per year will earn the business 10X extra revenue per year. If the cost of implementing this high availability strategy is less than the expected extra revenue, it would be worth implementing.

Chapter 11. Developing a backup and recovery strategy

A database can become unusable because of hardware or software failure, or both. You might, at one time or another, encounter storage problems, power interruptions, or application failures, and each failure scenario requires a different recovery action.

Protect your data against the possibility of loss by having a well rehearsed recovery strategy in place.

Some of the questions that you should answer when developing your recovery strategy are:

- Will the database be recoverable?
- How much time can be spent recovering the database?
- How much time will pass between backup operations?
- How much storage space can be allocated for backup copies and archived logs?
- Will table space level backups be sufficient, or will full database backups be necessary?
- Should I configure a standby system, either manually or through high availability disaster recovery (HADR)?

A database recovery strategy should ensure that all information is available when it is required for database recovery. It should include a regular schedule for taking database backups and, in the case of partitioned database environments, include backups when the system is scaled (when database partition servers or nodes are added or dropped). Your overall strategy should also include procedures for recovering command scripts, applications, user-defined functions (UDFs), stored procedure code in operating system libraries, and load copies.

Different recovery methods are discussed in the sections that follow, and you will discover which recovery method is best suited to your business environment.

The concept of a database *backup* is the same as any other data backup: taking a copy of the data and then storing it on a different medium in case of failure or damage to the original. The simplest case of a backup involves shutting down the database to ensure that no further transactions occur, and then simply backing it up. You can then recreate the database if it becomes damaged or corrupted in some way.

The recreation of the database is called *recovery*. *Version recovery* is the restoration of a previous version of the database, using an image that was created during a backup operation. *Rollforward recovery* is the reapplication of transactions recorded in the database log files after a database or a table space backup image has been restored.

Crash recovery is the automatic recovery of the database if a failure occurs before all of the changes that are part of one or more units of work (transactions) are completed and committed. This is done by rolling back incomplete transactions and completing committed transactions that were still in memory when the crash occurred.

Recovery log files and the recovery history file are created automatically when a database is created (Figure 51). These log files are important if you need to recover data that is lost or damaged.

Each database includes *recovery logs*, which are used to recover from application or system errors. In combination with the database backups, they are used to recover the consistency of the database right up to the point in time when the error occurred.

The *recovery history file* contains a summary of the backup information that can be used to determine recovery options, if all or part of the database must be recovered to a given point in time. It is used to track recovery-related events such as backup and restore operations, among others. This file is located in the database directory.

The *table space change history file*, which is also located in the database directory, contains information that can be used to determine which log files are required for the recovery of a particular table space.

You cannot directly modify the recovery history file or the table space change history file; however, you can delete entries from the files using the **PRUNE HISTORY** command. You can also use the **rec_his_retentn** database configuration parameter to specify the number of days that these history files will be retained.

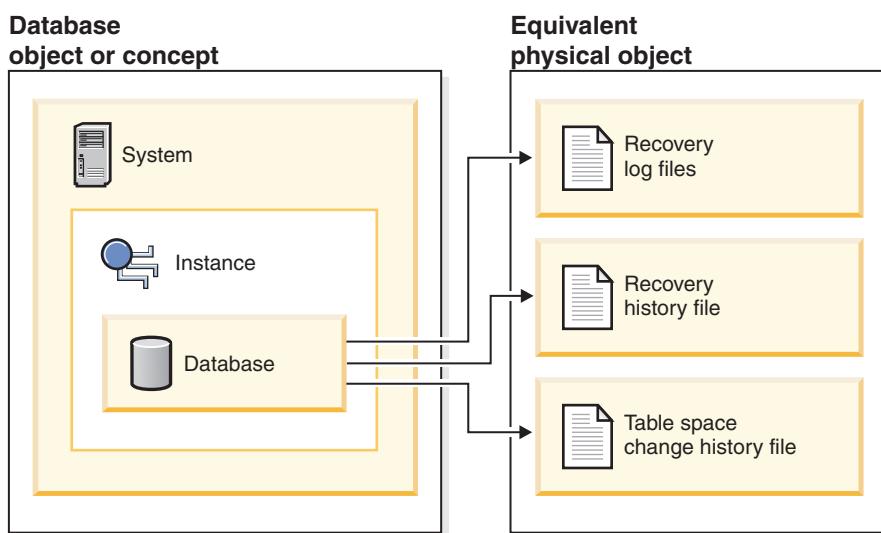


Figure 51. Database recovery files

Data that is easily re-created can be stored in a non-recoverable database. This includes data from an outside source that is used for read-only applications, and tables that are not often updated, for which the small amount of logging does not justify the added complexity of managing log files and rolling forward after a restore operation. If both the **logarchmeth1** and **logarchmeth2** database configuration parameters are set to OFF then the database is *Non-recoverable*. This means that the only logs that are kept are those required for crash recovery. These logs are known as *active logs*, and they contain current transaction data. Version recovery using *offline* backups is the primary means of recovery for a non-recoverable database. (An offline backup means that no other application can use the database when the backup operation is in progress.) Such a database can

only be restored offline. It is restored to the state it was in when the backup image was taken and rollforward recovery is not supported.

Data that *cannot* be easily recreated should be stored in a recoverable database. This includes data whose source is destroyed after the data is loaded, data that is manually entered into tables, and data that is modified by application programs or users after it is loaded into the database. *Recoverable databases* have the **logarchmeth1** or **logarchmeth2** database configuration parameters set to a value other than OFF. Active logs are still available for crash recovery, but you also have the *archived logs*, which contain committed transaction data. Such a database can only be restored offline. It is restored to the state it was in when the backup image was taken. However, with rollforward recovery, you can roll the database forward (that is, past the time when the backup image was taken) by using the active and archived logs to either a specific point in time, or to the end of the active logs.

Recoverable database backup operations can be performed either offline or *online* (online meaning that other applications can connect to the database during the backup operation). Online table space restore and rollforward operations are supported only if the database is recoverable. If the database is non-recoverable, database restore and rollforward operations must be performed offline. During an online backup operation, rollforward recovery ensures that *all* table changes are captured and reapplied if that backup is restored.

If you have a recoverable database, you can back up, restore, and roll individual table spaces forward, rather than the entire database. When you back up a table space online, it is still available for use, and simultaneous updates are recorded in the logs. When you perform an online restore or rollforward operation on a table space, the table space itself is not available for use until the operation completes, but users are not prevented from accessing tables in other table spaces.

Automated backup operations

Since it can be time-consuming to determine whether and when to run maintenance activities such as backup operations, you can use automatic maintenance. With automatic maintenance, you specify your maintenance objectives, including when automatic maintenance can run. DB2 then uses these objectives to determine if the maintenance activities need to be done and then runs only the required maintenance activities during the next available maintenance window (a user-defined time period for the running of automatic maintenance activities).

Note: You can still perform manual backup operations when automatic maintenance is configured. DB2 will only perform automatic backup operations if they are required.

Database logging

Database logging is an important part of your highly available database solution design because database logs make it possible to recover from a failure, and they make it possible to synchronize primary and secondary databases.

All databases have logs associated with them. These logs keep records of database changes. If a database needs to be restored to a point beyond the last full, offline backup, logs are required to roll the data forward to the point of failure.

Two types of database logging are supported: *circular* and *archive*. Each provides a different level of recovery capability:

- “Circular logging”
- “Archive logging” on page 285

The advantage of choosing archive logging is that rollforward recovery can use both archived logs and active logs to restore a database either to the end of the logs, or to a specific point in time. The archived log files can be used to recover changes made after the backup was taken. This is different from circular logging where you can only recover to the time of the backup, and all changes made after that are lost.

Circular logging

Circular logging is the default behavior when a new database is created. (The **logarchmeth1** and **logarchmeth2** database configuration parameters are set to OFF.)

With this type of logging, only full, offline backups of the database are allowed. The database must be offline (inaccessible to users) when a full backup is taken.

As the name suggests, circular logging uses a *ring* of online logs to provide recovery from transaction failures and system crashes. The logs are used and retained only to the point of ensuring the integrity of current transactions. Circular logging does not allow you to roll a database forward through transactions performed after the last full backup operation. All changes occurring since the last backup operation are lost. Since this type of restore operation recovers your data to the specific point in time at which a full backup was taken, it is called *version recovery*.

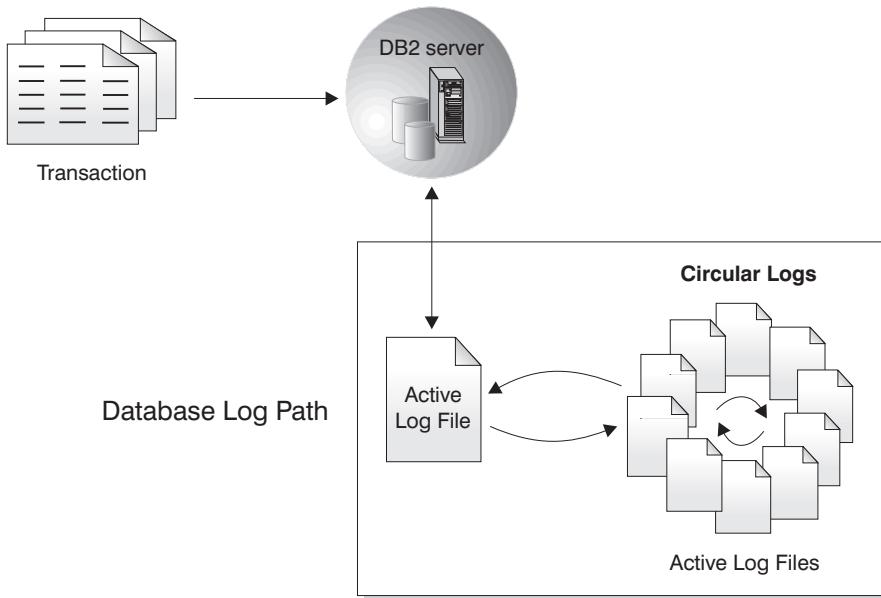


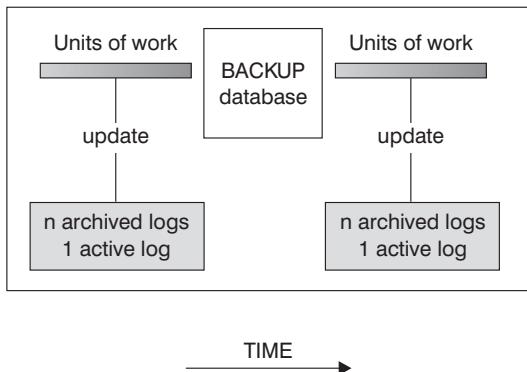
Figure 52. Circular Logging

Active logs are used during crash recovery to prevent a failure (system power or application error) from leaving a database in an inconsistent state. Active logs are located in the database log path directory.

Archive logging

Archive logging is used specifically for rollforward recovery. Archived logs are log files that are copied from the current log path or from the mirror log path to another location.

You can use the **logarchmeth1** database configuration parameter, the **logarchmeth2** database configuration parameter, or both to allow you or the database manager to manage the log archiving process.



Logs are used between backups to track the changes to the databases.

Figure 53. Active and archived database logs in rollforward recovery. There can be more than one active log in the case of a long-running transaction.

Taking online backups is supported only if you configure the database for archive logging. During an online backup operation, all activities against the database are logged. After an online backup is complete, the database manager forces the currently active log to close, and as a result, it is archived. This process ensures that your online backup has a complete set of archived logs available for recovery. When an online backup image is restored, the logs must be rolled forward at least to the point in time at which the backup operation completed. To facilitate this operation, archived logs must be made available when the database is restored.

You can use the **logarchmeth1** and **logarchmeth2** database configuration parameters to specify where archived logs are stored. You can use the **logarchmeth1** parameter to archive log files from the active log path that is set by the **logpath** configuration parameter. You can use the **logarchmeth2** parameter to archive additional copies of log files from the active log path to a second location. If you do not configure mirror logging, the additional copies are taken from the same log path that the **logarchmeth1** parameter uses. If you configure mirror logging, with the **mirrorlogpath** configuration parameter, the **logarchmeth2** configuration parameter archives log files from the mirror log path instead, which can improve resilience during rollforward recovery. The **newlogpath** parameter affects where active logs are stored.

In certain scenarios, you can compress archived log files to help reduce the storage cost that is associated with these files. If the **logarchmeth1** and **logarchmeth2** configuration parameters are set to DISK, TSM, or VENDOR, you can enable archived log file compression by setting the **logarchcompr1** and **logarchcompr2** configuration parameters to ON. If **logarchcompr1** and **logarchcompr2** are set dynamically, any log files that are already archived are not compressed.

If you use the LOGRETAIN option to specify a value that you want to manage the active logs, the database manager renames log files from the active log path after it archives these files and they are no longer needed for crash recovery. If you enable infinite logging, additional space is required for more active log files, so the database server renames the log files after it archives them. The database manager retains up to 8 extra log files in the active log path for renaming purposes.

Log control files

When a database restarts after a failure, the database manager applies transaction information stored in log files to return the database to a consistent state. To determine which records from the log files need to be applied to the database, the database manager uses information recorded in log control files.

Redundancy for database resilience

The database manager maintains two copies of the each member's log control file, SQLOGCTL.LFH.1 and SQLOGCTL.LFH.2, and two copies of the global log control file, SQLOGCTL.GLFH.1 and SQLOGCTL.GLFH.2, so that if one copy is damaged, the database manager can still use the other copy.

Performance considerations

Applying the transaction information contained in the log control files contributes to the overhead of restarting a database after a failure. You can configure the frequency at which the database manager writes buffer pool pages to disk in order to reduce the number of log records that need to be processed during crash recovery using the “softmax - Recovery range and soft checkpoint interval configuration parameter” in Database Administration Concepts and Configuration Reference.

Storage considerations for recovery

When deciding which recovery method to use, consider the storage space required. Backup and archived log file compression can help reduce the storage cost in your database environment.

The version recovery method requires space to hold the backup copy of the database and the restored database. The roll-forward recovery method requires space to hold the backup copy of the database or table spaces, the restored database, and the archived database logs.

If a table contains long field or large object (LOB) columns, you might consider placing this data into a separate table space. This action affects your storage space considerations, as well as affect your plan for recovery. With a separate table space for long field and LOB data, and knowing the time required to back up long field and LOB data, you might decide to use a recovery plan that only occasionally saves a backup of this table space. You can also choose, when creating or altering a table to include LOB columns, not to log changes to those columns. This action reduces the size of the required log space and the corresponding archived log file space.

To prevent media failure from destroying a database and your ability to restore it, keep the database backup, the database logs, and the database itself on different

devices. For this reason, it is highly recommended that you use the *newlogpath* configuration parameter to put database logs on a separate device once the database is created.

The database logs can use up a large amount of storage. If you plan to use the roll-forward recovery method, you must decide how to manage and compress the archived logs. Your choices are:

- Specify an archived log file method using the LOGARCHMETH1 or LOGARCHMETH2 configuration parameters.
- Enable archived log file compression with the LOGARCHCOMPR1 and LOGARCHCOMPR2 configuration parameters.
- Manually copy the logs to a storage device or directory other than the database log path directory after they are no longer in the active set of logs.
- Use a user exit program to copy these logs to another storage device in your environment.

Archived log file compression

As of DB2 V10.1, you can compress archived log files. This capability, in addition to data and index compression, along with backup compression, reduces the amount of disk space required for your database environment.

Archived log files are the third major space consumer for roll-forward recoverable databases. Archived log files contain a significant amount of data and these archives can grow quickly. If modified data is already in compressed tables, logging is reduced by virtue of including compressed record images in log records. Compression of archived log files further increases storage savings, even in these environments.

To use compression for your archived log files, you can use the **UPDATE DB CFG** command to set the **logarchcompr1** and **logarchcompr2** configuration parameters to ON.

Restrictions

- Archived log file compression does not take effect under the following conditions.
 - The corresponding archived log file method is not set to DISK, TSM, or VENDOR. When the corresponding archived log file method is set as described, the log files are physically moved out of the active log path, or the mirror log path.
 - Whenever archived log file compression is enabled, but the corresponding log archiving method is set to OFF, LOGRETAIN or USEREXIT, archived log file compression has no effect. Any update to the **logarchmeth1** and **logarchmeth2** or the **logarchcompr1** and **logarchcompr2** database configuration parameters which results in such a scenario returns a warning, SQL1663W.

Note: When the database is activated, SQL1663W is not returned when setting or changing archived log file compression database configuration parameters. Instead, SQL1363W is returned, which is a higher priority message. If the database is not activated, the SQL1663W warning message is returned.

- Manual archiving and retrieval with **db2adut1**.
 - The **db2adut1** utility does not perform compression or decompression during UPLOAD or EXTRACT operations. Movement of compressed log files to and from the archive location is fully supported by **db2adut1**.

- If logs are uploaded to Tivoli® Storage Manager with **db2adut1**, and you want to compress archived log files, archived log file compression must be enabled when the logs are archived to the disk location, before **db2adut1** picks them up. If compressed logs are retrieved manually with **db2adut1**, they are extracted on first access.
- Archived log file compression is not supported when raw devices are used for database logging.
 - Archived log file compression is not supported when either the **logpath** or the **newlogpath** database configuration parameters point to a raw device. Any database configuration update that results in archived log file compression being enabled while **logpath** or **newlogpath** database configuration parameters point to raw devices fails, SQL1665N.
- When enabling archived log file compression using the **logarchcompr1** and **logarchcompr2** database configuration parameters, logs already stored in a backup image are not affected.

Backup and restore operations between different operating systems and hardware platforms

DB2 database systems support some backup and restore operations between different operating systems and hardware platforms.

The supported platforms for DB2 backup and restore operations can be grouped into one of three families:

- Big-endian Linux and UNIX
- Little-endian Linux and UNIX
- Windows

A database backup from one platform family can only be restored on any system within the same platform family. For Windows operating systems, you can restore a database that was created on DB2 Version 9.7 on a DB2 Version 10.1 database system. For Linux and UNIX operating systems, as long as the endianness (big endian or little endian) of the backup and restore platforms is the same, you can restore backups that were produced on down level versions.

The following table shows each of the Linux and UNIX platforms DB2 supports and indicates whether the platforms are big endian or little endian:

Table 22. Endianness of supported Linux and UNIX operating systems DB2 supports

Platform	Endianness
AIX	big endian
HP on IA64	big endian
Solaris x64	little endian
Solaris SPARC	big endian
Linux on zSeries	big endian
Linux on pSeries	big endian
Linux on IA-64	little endian
Linux on AMD64 and Intel EM64T	little endian
32-bit Linux on x86	little endian

The target system must have the same (or later) version of the DB2 database product as the source system. You cannot restore a backup that was created on one version of the database product to a system that is running an earlier version of the database product. For example, you can restore a DB2 Version 9.7 on a DB2 Version 10.1 database system, but you cannot restore a DB2 Version 10.1 backup on a DB2 Version 9.7 database system.

Note: You can restore a database from a backup image that was taken on a 32-bit level into a 64-bit level, but not vice versa. The DB2 backup and restore utilities should be used to back up and restore your databases. Moving a file set from one machine to another is not recommended as this can compromise the integrity of the database.

In situations where certain backup and restore combinations are not allowed, you can move tables between DB2 databases using other methods:

- The **db2move** command
- The **export** command followed by the **import** or the **load** command

Note: Database configuration parameters are set to their defaults if the values in the backup are outside of the allowable range for the environment in which the database is being restored.

Log stream merging and log file management in a DB2 pureScale environment

In a DB2 pureScale environment, each member maintains its own set of transaction log files (that is, a *log stream*) on the shared disk, each set in a separate log path. The log files for a member contain a history of all data changes that occurred on that member.

Multiple applications, each accessing a different member simultaneously, might generate dependent transactions during run time. A dependency between two transactions can occur if, for example, both transactions change the same row. To effectively interpret the log records, the DB2 data server must examine the records from all log streams and order the records so that they reflect the order of the updates that occurred at run time. This ordering is known as a *log stream merge* operation. Several operation types in a DB2 pureScale environment require log stream merges; these include (among others) group crash recovery, database roll-forward operations, and table space roll-forward operations.

Logging configuration parameters in a DB2 pureScale environment

Table 23 shows which logging-related database configuration parameters are global in scope and which parameters are dynamically updatable.

Table 23. Logging-related database configuration parameters

Parameter	Global?	Dynamically updatable?
archretrydelay	Yes	Yes
blk_log_dsk_ful	No	Yes
failarchpath	Yes	Yes
logarchcompr1	Yes	Yes
logarchcompr2	Yes	Yes

Table 23. Logging-related database configuration parameters (continued)

Parameter	Global?	Dynamically updatable?
logarchmeth1	Yes	Yes
logarchmeth2	Yes	Yes
logarchopt1	Yes	Yes
logarchopt2	Yes	Yes
logbufsz	No	Yes
logfilsiz	Yes	No
logprimary	Yes	No
logsecond	Yes	Yes
max_log	No	Yes
mirrorlogpath ¹	Yes	No
newlogpath ¹	Yes	No
num_log_span	No	Yes
numarchretry	Yes	Yes
overflowlogpath	Yes	Yes
softmax	Yes	No
vendoropt	Yes	Yes

¹ The first member that connects to or activates the database processes the changes to this log path parameter. The DB2 database manager verifies that the path exists and that it has both read and write access to that path. It also creates member-specific subdirectories for the log files. If any one of these operations fails, the DB2 database manager rejects the specified path and brings the database online using the old path. If the database manager accepts the specified path, the new value is propagated to each member. If a member fails while trying to switch to the new path, subsequent attempts to activate the database or to connect to it fails, and SQL5099N is returned. All members must use the same log path.

Retrieving logs for a log stream merge operation in a DB2 pureScale environment

A subdirectory is created in the path for retrieved log files. The subdirectory has the following format: *log_path*/LOGSTREAMxxxx, where *log_path* represents the log path, overflow log path, or mirror log path, and xxxx is a 4-digit log stream identifier. (The log stream identifier is not necessarily equivalent to the associated member ID.) Within this subdirectory, if a member requires log retrieval, the DB2 database manager creates another level of subdirectories for retrieved logs from each member. For example, if you specify an overflow log path of /home/dbuser/overflow/ on a 3-member system, and an application on member 0 must retrieve logs that are owned by other members, the path for member 0 is /home/dbuser/overflow/NODE0000/LOGSTREAM0000, and subdirectories under this path contain retrieved logs that are owned by other members, as shown in the following example:

```

Member 0 retrieves its own logs here:
/home/dbuser/overflow/NODE0000/LOGSTREAM0000/LOGSTREAM0000
Member 0 retrieves logs that belong to member 1 here:
/home/dbuser/overflow/NODE0000/LOGSTREAM0000/LOGSTREAM0001
Member 0 retrieves logs that belong to member 2 here:
/home/dbuser/overflow/NODE0000/LOGSTREAM0000/LOGSTREAM0002

```

Note: Do not manually insert log files in to these retrieve subdirectories. If you want to manually retrieve log files, use the overflow log path instead.

When reading archived log files that are owned by other members, a member might need to retrieve log files in to its own log path or overflow log path. In this case, the log stream merge operation creates a **db2logmgr** engine dispatchable unit (EDU) for each log stream, as needed.

As mentioned earlier, there are three paths that can be used to store log files that are owned by other members, as shown in the following list:

1. If you set the **overflowlogpath** database configuration parameter, the overflow log path is used.

Tip: You can use **ROLLFORWARD DATABASE** and **RECOVER DATABASE** command options to specify an alternative overflow log path; the values of these options override the database configuration for purposes of the single recovery operation.

2. The primary log path
3. If you set the **mirrorlogpath** database configuration parameter, the mirror log path is used.

If the DB2 database manager is unable to store a log file in the first path, it attempts to use the next path in the list. If none of these paths is available, the utility that invoked the log stream merge operation returns an error that is specific to that utility.

Output from the **GET DATABASE CONFIGURATION** command in a DB2 pureScale environment identifies each log path followed by the name of the member. For example, if the mirror log path was set to `/home/dbuser/mirrorpath/`, for member 2, the output displays `/home/dbuser/mirrorpath/NODE0000/LOGSTREAM0002`.

If you must manually retrieve log files that are owned by other members, ensure that the database manager can access the log files by using the same directory structure that is automatically created. For example, to make logs from member 2 available in the overflow log path of member 1, place the logs in the `/home/dbuser/overflow/NODE0000/LOGSTREAM0001/LOGSTREAM0002` directory.

Retrieved log files are automatically deleted when they are no longer needed. Subdirectories that were created during a log stream merge operation are retained for future use.

Detection of missing logs during a log stream merge operation

If you accidentally deleted, moved, or archived and lost a log file that is required for a recovery operation, you can roll-forward recover the database to the last consistent point before the missing log file.

If, during a log stream merge operation, the DB2 database manager determines that there is a missing log file in one of the log streams, an error is returned. The roll-forward utility returns SQL1273N; the db2ReadLog API returns SQL2657N.

Figure 54 on page 292 shows an example of how two members could write log records to the log files in their active log stream. Each log file is represented by a box.

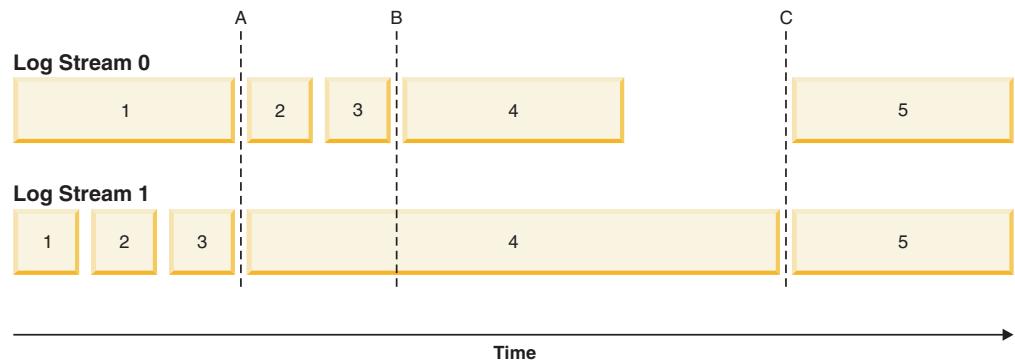


Figure 54. Log files in a DB2 pureScale environment

Consider a scenario where only log file 4 from log stream 1 is missing, a roll-forward operation to time A succeeds while roll-forward operations to time B, time C, or to the END OF LOGS fail. The ROLLFORWARD command returns SQL1273N because log file 4 is not available. Furthermore, since the log records in files 2 and 3 on log stream 0 were written during the same time period as the beginning of log file 4 on log stream 1, the roll-forward operation cannot process log files 2 and 3 until log file 4 from log stream 1 is available. The result is that the roll-forward operation stops at time A, and any subsequent roll-forward operations cannot proceed beyond time A until log 4 from stream 1 becomes available.

Consider another scenario where only log file 4 from log stream 0 is missing during a roll-forward operation. If you issue a **ROLLFORWARD** command with the **END OF LOGS** option (or anytime after time B), the operation will stop at time B and will return SQL1273N because log file 4 on stream 0 is missing. A roll-forward operation can replay log records from files 2 and 3 on log stream 0 and some logs from file 4 on stream 1 up to time B. The roll-forward operation must stop at time B even though additional logs from stream 1 are available because the log merge process requires that all the logs from all the streams be available.

If you can find the missing log file, make it available and reissue the **ROLLFORWARD DATABASE** command. If you cannot find the missing log file, issue the **ROLLFORWARD DATABASE...STOP** command to complete the roll-forward operation at the last consistent point just before the missing log file.

Although missing log detection ensures that database corruption does not occur as a result of missing log files, the presence of missing log files prevents some transactions from being replayed and, as a result, data loss could occur if the missing log files are not located.

Required resources

Log stream merge operations require additional EDUs. During database activation, one **db21fr** EDU is created on each member. When a log read operation that requires a log stream merge is initiated, one **db2shred** EDU and one **db21fr** EDU is created for each log stream. Although each **db21fr-db2shred** group allocates its own set of log page and log record buffers, this is not a significant amount of additional memory or system resources; approximately 400 KB is allocated for each member that is involved in the log stream merge.

During a log stream merge operation, a member retrieves log files that are owned by other members into its overflow log path, primary log path, or mirror log path. In a DB2 pureScale environment, ensure that there is adequate free disk space in

the retrieval path before starting a roll-forward operation. This allows the operation to retrieve the larger number of files from the archive, as required in a DB2 pureScale environment, without affecting performance. Use the following rule-of-thumb to calculate how much space you need to retrieve the active log files for all members: (**logprimary + logsecond**) * number of members.

Examples

- Update the **newlogpath** global database configuration parameter:
`db2 update db cfg for db mydb using newlogpath /home/dbuser/logdir`
- Update the **max_log** per-member database configuration parameter on a single member:
`db2 update db cfg for db mydb member 1 using max_log 5`
- Update the primary log path:

```
db2 connect to mydb  
db2 update db cfg for mydb using newlogpath /home/dbuser/newlogpath  
db2 get db cfg for mydb  
...  
Changed path to log files (NEWLOGPATH) = /home/dbuser/newlogpath/NODE0000/LOGSTREAM0000/  
Path to log files = /home/dbuser/dbuser/NODE0000/LOGSTREAM0000/  
...
```

The change does not take effect because the member is still active.

```
db2 terminate  
db2 deactivate db mydb  
db2 connect to mydb  
db2 get db cfg for mydb  
...  
Changed path to log files (NEWLOGPATH) =  
Path to log files = /home/dbuser/newlogpath/NODE0000/LOGSTREAM0000/  
...
```

Each member uses the `/home/dbuser/newlogpath/NODE0000/LOGSTREAMxxxx` log path, where `xxxx` is the log stream ID of the log stream that uses the path.

- Set a new primary log path while restoring a backup image:
`db2 restore db mydb newlogpath '/home/dbuser/newlogpath' without prompting`

Log sequence numbers in DB2 pureScale environments

DB2 databases use the log sequence number (LSN), a 64-bit identifier, to determine the order of the operations that generated the log records.

The LSN is an ever-increasing value. Each member writes to its own set of log files (*a log stream*), and the LSN within a single log stream is a unique number.

Because LSNs are generated independently on each member and there are multiple log streams, it is possible to have duplicate LSN values across different log streams. A log record identifier (LRI) is used to identify log records across log streams; each log record in any log stream in the database is assigned a unique LRI. Use the **db2pd** command to determine which LRI is being processed by a recovery operation.

Including log files with a backup image

When performing an online backup operation, you can specify that the log files required to restore and recover a database are included in the backup image.

This means that if you need to ship backup images to a disaster recovery site, you do not have to send the log files separately or package them together yourself. Further, you do not have to decide which log files are required to guarantee the consistency of an online backup. This provides some protection against the deletion of log files required for successful recovery.

To use this feature, specify the **INCLUDE LOGS** option of the **BACKUP DATABASE** command. When you specify this option, the backup utility truncates the currently active log file and copies the necessary set of log extents into the backup image.

To restore the log files from a backup image, use the **LOGTARGET** option of the **RESTORE DATABASE** command and specify a fully qualified path that exists on the DB2 server. The restore database utility then writes the log files from the image to the target path. If a log file with the same name exists in the target path, the restore operation fails and an error is returned. If the **LOGTARGET** option is not specified, no log files are restored from the backup image.

If the **LOGTARGET** option is specified and the backup image does not include any log files, an error is returned before an attempt is made to restore any table space data. The restore operation also fails if an invalid or read-only path is specified. During a database or table space restore where the **LOGTARGET** option is specified, if one or more log files cannot be extracted, the restore operation fails and an error is returned.

You can also choose to restore only the log files saved in the backup image. To do this, specify the **LOGS** option with the **LOGTARGET** option of the **RESTORE DATABASE** command. If the restore operation encounters any problems when restoring log files in this mode, the restore operation fails and an error is returned.

During an automatic incremental restore operation, only the logs included in the target image of the restore operation are retrieved from the backup image. Any logs that are included in intermediate images referenced during the incremental restore process are not extracted from those backup images. During a manual incremental restore, if you specify a log target directory when restoring a backup image that includes log files, the log files in that backup image are restored.

If you roll a database forward that was restored from an online backup image that includes log files, you might encounter error SQL1268N, which indicates roll-forward recovery stopped due to an error received when retrieving a log. This error is generated when the target system to which you are attempting to restore the backup image does not have access to the facility used by the source system to archive its transaction logs.

If you specify the **INCLUDE LOGS** option of the **BACKUP DATABASE** command when you back up a database, then perform a restore operation and a roll-forward operation that use that back up image, DB2 still searches for additional transaction logs when rolling the database forward, even though the backup image includes logs. It is standard rollforward behavior to continue to search for additional transaction logs until no more logs are found. It is possible to have more than 1 log file with the same timestamp. Consequently, DB2 does not stop as soon as it finds the first timestamp that matches the point-in-time to which you are rolling forward the database as there might be other log files that also have that timestamp. Instead, DB2 continues to look at the transaction log until it finds a timestamp greater than the point-in-time specified.

When no additional logs can be found, the rollforward operation ends successfully. However, if there is an error while searching for additional transaction log files, error SQL1268N is returned. Error SQL1268N can occur because during the initial restore, certain database configuration parameters were reset or overwritten. Three of these database configuration parameters are the TSM parameters, **tsm_nodename**, **tsm_owner**, and **tsm_password**. They are all reset to NULL. To rollforward to the end of logs, you need to reset these database configuration parameters to correspond to the source system before the rollforward operation. Alternatively, you can specify the **NORETRIEVE** option when you issue the **ROLLFORWARD DATABASE** command. This prevents the DB2 database system from trying to obtain potentially missing transaction logs elsewhere.

Note:

1. This feature is not supported for offline backups.
2. When logs are included in an online backup image, the resulting image cannot be restored on releases of DB2 database before Version 8.2.

Incremental backup and recovery

As the size of databases, and particularly warehouses, continues to expand into the terabyte and petabyte range, the time and hardware resources required to back up and recover these databases is also growing substantially.

Full database and table space backups are not always the best approach when dealing with large databases, because the storage requirements for multiple copies of such databases are enormous.

Consider the following issues:

- When a small percentage of the data in a warehouse changes, it should not be necessary to back up the entire database.
- Appending table spaces to existing databases and then taking only table space backups is risky, because there is no guarantee that nothing outside of the backed up table spaces has changed between table space backups.

To address these issues, DB2 provides incremental backup and recovery.

An *incremental backup* is a backup image that contains only pages that have been updated since the previous backup was taken. In addition to updated data and index pages, each incremental backup image also contains all of the initial database metadata (such as database configuration, table space definitions, database history, and so on) that is normally stored in full backup images.

Note:

1. If a table space contains long field or large object data and an incremental backup is taken, all of the long field or large object data will be copied into the backup image if any of the pages in that table space have been modified since the previous backup.
2. If you take an incremental backup of a table space that contains a dirty page (that is, a page that contains data that has been changed but has not yet been written to disk) then all large object data is backed up. Normal data is backed up only if it has changed.
3. Data redistribution might create table spaces for all new database partitions if the **ADD DBPARTITIONNUMS** parameter on the **REDISTRIBUTE DATABASE PARTITION GROUP** command is specified; this can affect incremental backup operations.

Two types of incremental backup are supported:

- *Incremental*. An incremental backup image is a copy of all database data that has changed since the most recent, successful, full backup operation. This is also known as a cumulative backup image, because a series of incremental backups taken over time will each have the contents of the previous incremental backup image. The predecessor of an incremental backup image is always the most recent successful full backup of the same object.
- *Delta*. A delta, or incremental delta, backup image is a copy of all database data that has changed since the last successful backup (full, incremental, or delta) of the table space in question. This is also known as a differential, or noncumulative, backup image. The predecessor of a delta backup image is the most recent successful backup containing a copy of each of the table spaces in the delta backup image.

The key difference between incremental and delta backup images is their behavior when successive backups are taken of an object that is continually changing over time. Each successive incremental image contains the entire contents of the previous incremental image, plus any data that has changed, or is new, since the previous full backup was produced. Delta backup images contain only the pages that have changed since the previous image of any type was produced.

Combinations of database and table space incremental backups are permitted, in both online and offline modes of operation. Be careful when planning your backup strategy, because combining database and table space incremental backups implies that the predecessor of a database backup (or a table space backup of multiple table spaces) is not necessarily a single image, but could be a unique set of previous database and table space backups taken at different times.

To restore the database or the table space to a consistent state, the recovery process must begin with a consistent image of the entire object (database or table space) to be restored, and must then apply each of the appropriate incremental backup images in the order described in the following list.

To enable the tracking of database updates, DB2 supports a new database configuration parameter, **trackmod**, which can have one of two accepted values:

- NO. Incremental backup is not permitted with this configuration. Database page updates are not tracked or recorded in any way. This is the default value.
- YES. Incremental backup is permitted with this configuration. When update tracking is enabled, the change becomes effective at the first successful connection to the database. Before an incremental backup can be taken on a particular table space, a full backup of that table space is necessary.

For SMS and DMS table spaces, the granularity of this tracking is at the table space level. In table space level tracking, a flag for each table space indicates whether or not there are pages in that table space that need to be backed up. If no pages in a table space need to be backed up, the backup operation can skip that table space altogether.

Although minimal, the tracking of updates to the database can have an impact on the runtime performance of transactions that update or insert data.

Restoring from incremental backup images

A restore operation from incremental backup images consists of four steps.

About this task

1. Identifying the incremental target image.

Determine the final image to be restored, and request an incremental restore operation from the DB2 restore utility. This image is known as the target image of the incremental restore, because it is the last image to be restored. The incremental target image is specified using the **TAKEN AT** parameter in the **RESTORE DATABASE** command.
2. Restoring the most recent full database or table space image to establish a baseline against which each of the subsequent incremental backup images can be applied.
3. Restoring each of the required full or table space incremental backup images, in the order in which they were produced, on top of the baseline image restored in Step 2.
4. Repeating Step 3 until the target image from Step 1 is read a second time. The target image is accessed twice during a complete incremental restore operation. During the first access, only initial data is read from the image; none of the user data is read. The complete image is read and processed only during the second access.

The target image of the incremental restore operation must be accessed twice to ensure that the database is initially configured with the correct history, database configuration, and table space definitions for the database that is created during the restore operation. In cases where a table space was dropped since the initial full database backup image was taken, the table space data for that image is read from the backup images but ignored during incremental restore processing.

There are two ways to restore incremental backup images: automatic and manual:

- For an automatic incremental restore, the **RESTORE DATABASE** command is issued only once specifying the target image to be used. DB2 for Linux, UNIX, and Windows then uses the database history to determine the remaining required backup images and restores them.
- For a manual incremental restore, the **RESTORE DATABASE** command must be issued once for each backup image that needs to be restored (as outlined in the steps listed previously).

Procedure

- To restore a set of incremental backup images using automatic incremental restore, issue the **RESTORE DATABASE** command specifying time stamp of the last image you want to restore with the **TAKEN AT** parameter, as follows:

```
db2 restore db sample incremental automatic taken at timestamp
```

This results in the restore utility performing each of the steps described at the beginning of this section automatically. During the initial phase of processing, the backup image with the specified time stamp (specified in the form *yyyymmddhhmmss*) is read, and the restore utility verifies that the database, its history, and the table space definitions exist and are valid.

During the second phase of processing, the database history is queried to build a chain of backup images required to perform the requested restore operation. If, for some reason this is not possible, and DB2 for Linux, UNIX, and Windows is unable to build a complete chain of required images, the restore operation terminates, and an error message is returned. In this case, an automatic incremental restore is not possible, and you must issue the **RESTORE DATABASE** command with the **INCREMENTAL ABORT** parameter. This will clean up any remaining resources so that you can proceed with a manual incremental restore.

Note: It is highly recommended that you not use the **WITH FORCE OPTION** of the **PRUNE HISTORY** command. The default operation of this command prevents you from deleting history entries that might be required for recovery from the most recent, full database backup image, but with the **WITH FORCE OPTION**, it is possible to delete entries that are required for an automatic restore operation.

During the third phase of processing, DB2 for Linux, UNIX, and Windows restores each of the remaining backup images in the generated chain. If an error occurs during this phase, you must issue the **RESTORE DATABASE** command with the **INCREMENTAL ABORT** option to clean up any remaining resources. You must then determine whether the error can be resolved before you reissue the **RESTORE DATABASE** command or attempt the manual incremental restore again.

- To restore a set of incremental backup images, using manual incremental restore, issue **RESTORE DATABASE** commands specifying time stamp of each image you want to restore with the **TAKEN AT** parameter, as follows:

1.

```
db2 restore database dbname incremental taken at timestamp
```

where *timestamp* points to the last incremental backup image (*the target image*) to be restored.

2.

```
db2 restore database dbname incremental taken at timestamp1
```

where *timestamp1* points to the initial full database (or table space) image.

3.

```
db2 restore database dbname incremental taken at timestampX
```

where *timestampX* points to each incremental backup image in creation sequence.

4.

Repeat Step 3, restoring each incremental backup image up to and including image *timestamp*.

If you are performing a database restore operation, and table space backup images have been produced, the table space images must be restored in the chronological order of their backup time stamps.

The **db2ckrst** utility can be used to query the database history and generate a list of backup image time stamps needed for an incremental restore. A simplified restore syntax for a manual incremental restore is also generated. It is recommended that you keep a complete record of backups, and use this utility only as a guide.

Limitations to automatic incremental restore

The automatic incremental restore is useful when you need to restore your database. However, you should consider the limitations of automatic incremental restore when you are deciding how you will recover your database to prevent unnecessary issues.

The following limitations affect automatic incremental restore:

1. If a table space name has been changed since the backup operation you want to restore from, and you use the new name when you issue a table space level restore operation, the required chain of backup images from the database history will not be generated correctly and an error will occur (SQL2571N).

Example:

```
db2 backup db sample -> <ts1>
db2 backup db sample incremental -> <ts2>
db2 rename tablespace from userspace1 to t1
db2 restore db sample tablespace ('t1') incremental automatic taken
at <ts2>
```

SQL2571N Automatic incremental restore is unable to proceed.
Reason code: "3".

Suggested workaround: Use manual incremental restore.

2. If you drop a database, the database history will be deleted. If you restore the dropped database, the database history will be restored to its state at the time of the restored backup and all history entries after that time will be lost. If you then attempt to perform an automatic incremental restore that would need to use any of these lost history entries, the RESTORE utility will attempt to restore an incorrect chain of backups and will return an "out of sequence" error (SQL2572N).

Example:

```
db2 backup db sample -> <ts1>
db2 backup db sample incremental -> <ts2>
db2 backup db sample incremental delta -> <ts3>
db2 backup db sample incremental delta -> <ts4>
db2 drop db sample
db2 restore db sample incremental automatic taken at <ts2>
db2 restore db sample incremental automatic taken at <ts4>
```

Suggested workarounds:

- Use manual incremental restore.
- Restore the history file first from image <ts4> before issuing an automatic incremental restore.

3. If you restore a backup image from one database into another database and then do an incremental (delta) backup, you can no longer use automatic incremental restore to restore this backup image.

Example:

```
db2 create db a
db2 create db b

db2 update db cfg for a using trackmod on

db2 backup db a -> ts1
db2 restore db a taken at ts1 into b

db2 backup db b incremental -> ts2

db2 restore db b incremental automatic taken at ts2
```

SQL2542N No match for a database image file was found based on the source database alias "B" and timestamp "ts1" provided.

Suggested workaround:

- Use manual incremental restore as follows:

```
db2 restore db b incremental taken at ts2
db2 restore db a incremental taken at ts1 into b
db2 restore db b incremental taken at ts2
```
- After the manual restore operation into database B, issue a full database backup to start a new incremental chain

Chapter 12. Backing up databases

Backing up a database makes a copy of the database data and stores it on a different medium. This database backup can then be used in the case of a failure or damage to the original data.

Before you begin

You do not need to be connected to the database that is to be backed up: the backup database utility automatically establishes a connection to the specified database, and this connection is terminated at the completion of the backup operation. If you are connected to a database that is to be backed up, you will be disconnected when the **BACKUP DATABASE** command is issued and the backup operation will proceed.

The database can be local or remote. The backup image remains on the database server, unless you are using a storage management product such as Tivoli Storage Manager (TSM) or DB2 Advanced Copy Services (ACS).

If you are performing an offline backup and if you have activated the database by using the **ACTIVATE DATABASE** command, you must deactivate the database before you run the offline backup. If there are active connections to the database, in order to deactivate the database successfully, a user with SYSADM authority must connect to the database, and issue the following commands:

```
CONNECT TO database-alias
QUIESCE DATABASE IMMEDIATE FORCE CONNECTIONS;
UNQUIESCE DATABASE;
TERMINATE;
DEACTIVATE DATABASE database-alias
```

In a partitioned database environment, you can use the **BACKUP DATABASE** command to back up database partitions individually, use the **ON DBPARTITIONNUM** command parameter to back up several of the database partitions at once, or use the **ALL DBPARTITIONNUMS** parameter to back up all of the database partitions simultaneously. You can use the **LIST DBPARTITIONNUMS** command to identify the database partitions that have user tables on them that you might want to back up.

Unless you are using a single system view (SSV) backup, if you are performing an *offline* backup in a partitioned database environment, you should back up the catalog partition separately from all other database partitions. For example, you can back up the catalog partition first, then back up the other database partitions. This action is necessary because the backup operation might require an exclusive database connection on the catalog partition, during which the other database partitions cannot connect. If you are performing an *online* backup, all database partitions (including the catalog partition) can be backed up simultaneously or in any order.

On a distributed request system, backup operations apply to the distributed request database and the metadata stored in the database catalog (wrappers, servers, nicknames, and so on). Data source objects (tables and views) are not backed up, unless they are stored in the distributed request database.

If a database was created with a previous release of the database manager, and the database has not been upgraded, you must upgrade the database before you can back it up.

Restrictions

The following restrictions apply to the backup utility:

- A table space backup operation and a table space restore operation cannot be run at the same time, even if different table spaces are involved.
- If you want to be able to do rollforward recovery in a partitioned database environment, you must regularly back up the database on the list of nodes. You must also have at least one backup image of the rest of the nodes in the system (even those nodes that do not contain user data for that database). Two situations require the backed-up image of a database partition at a database partition server that does not contain user data for the database:
 - You added a database partition server to the database system after taking the last backup, and you need to do forward recovery on this database partition server.
 - Point-in-time recovery is used, which requires that all database partitions in the system are in rollforward pending state.
- Online backup operations for DMS table spaces are incompatible with the following operations:
 - load
 - reorganization (online and offline)
 - drop table space
 - table truncation
 - index creation
 - not logged initially (used with the CREATE TABLE and ALTER TABLE statements)
- If you attempt to perform an offline backup of a database that is currently active, you will receive an error. Before you run an offline backup, you can make sure that the database is not active by issuing the **DEACTIVATE DATABASE** command.

Procedure

To invoke the backup utility:

- Issue the **BACKUP DATABASE** command in the command line processor (CLP).
- Run the ADMIN_CMD procedure with the BACKUP DATABASE parameter.
- Use the db2Backup application programming interface (API).
- Open the task assistant in IBM Data Studio for the **BACKUP DATABASE** command.

Example

Following is an example of the **BACKUP DATABASE** command issued through the CLP:

```
db2 backup database sample to c:\DB2Backups
```

What to do next

If you performed an offline backup, after the backup completes, you must reactivate the database:

```
ACTIVATE DATABASE sample
```

Performing a snapshot backup

A snapshot backup operation uses the fast copying technology of a storage device to perform the data copying portion of the backup.

Before you begin

To perform snapshot backup and restore operations, you need a DB2 ACS API driver for your storage device. For a list of supported storage hardware for the integrated driver, refer to this tech note.

Before you can perform a snapshot backup, you must enable DB2 Advanced Copy Services (ACS).

Restrictions

You cannot recover individual table spaces by using snapshot backups.

If you use integrated snapshot backups, you cannot perform a redirected restore. A FlashCopy® restore reverts the complete set of volume groups containing all database paths to a prior point in time.

Procedure

To perform a snapshot backup, use one of the following approaches:

- Issue the **BACKUP DATABASE** command with the **USE SNAPSHOT** parameter, as shown in the following example:

```
db2 backup db sample use snapshot
```

- Call the **ADMIN_CMD** procedure with **BACKUP DB** and **USE SNAPSHOT** parameters, as shown in the following example:

```
CALL SYSPROC.ADMIN_CMD  
      ('backup db sample use snapshot')
```

- Issue the **db2Backup** API with the **SQLU_SNAPSHOT_MEDIA** media type, as shown in the following example:

```
int sampleBackupFunction( char dbAlias[],  
                         char user[],  
                         char pswd[],  
                         char workingPath[] )  
{  
    db2MediaListStruct mediaListStruct = { 0 };  
  
    mediaListStruct.locations = &workingPath;  
    mediaListStruct.numLocations = 1;  
    mediaListStruct.locationType = SQLU_SNAPSHOT_MEDIA;  
  
    db2BackupStruct backupStruct = { 0 };  
  
    backupStruct.piDBAlias = dbAlias;  
    backupStruct.piUsername = user;  
    backupStruct.piPassword = pswd;  
    backupStruct.piVendorOptions = NULL;  
    backupStruct.piMediaList = &mediaListStruct;
```

```
    db2Backup(db2Version950, &backupStruct, &sqlca);  
    return 0;  
}
```

Using a split mirror as a backup image

Use the following procedure to create a split mirror of a database in a different location on the same system for use as a backup image outside of a DB2 pureScale environment. This procedure can be used instead of performing backup database operations on the database.

Procedure

To use a split mirror as a backup image:

1. Connect to the primary database by using the following command:

```
db2 connect to db_name
```

2. Suspend the I/O write operations on the primary database by using the following command:

```
db2 set write suspend for database
```

While the database is in suspended state, you should not be running other utilities or tools. You should be only making a copy of the database. You can optionally flush all buffer pools before you issue **SET WRITE SUSPEND** to minimize the recovery window. This can be achieved by using the FLUSH BUFFERPOOLS ALL statement.

3. Create one or multiple split mirrors from the primary database by using the appropriate operating system-level and storage-level commands.

Note:

- Ensure that you copy the entire database directory, including the volume directory. You must also copy the log directory and any container directories that exist outside the database directory. To gather this information, refer to the DBPATHS administrative view, which shows all the files and directories of the database that need to be split.
 - If you specified EXCLUDE LOGS with the **SET WRITE** command, do not include the log files in the copy.
4. Resume the I/O write operations on the primary database by using the following command:

```
db2 set write resume for database
```

Assuming that a failure would occur on the system, perform the following steps to restore the database by using the split-mirror database as the backup:

- a. Stop the database instance by using the following command:

```
db2stop
```

- b. Copy the split-off data by using operating system-level commands.

Important: Do not copy the split-off log files, because the original logs are needed for rollforward recovery.

- c. Start the database instance by using the following command:

```
db2start
```

- d. Initialize the primary database:

```
db2inidb database_alias as mirror
```

- where *database_alias* represents the database alias.
- e. Roll forward the database to the end of the logs, or to a point-in-time, and stop.

Using a split mirror as a backup image in a DB2 pureScale environment

Use the following procedure to create a split mirror of a database in a different location on the same system for use as a backup image in a DB2 pureScale environment. This procedure can be used instead of performing backup database operations on the database.

Procedure

To use a split mirror as a backup image:

1. Connect to the primary database by using the following command:
`db2 connect to db_name`
2. Configure the General Parallel File System (GPFS) on the secondary cluster by extracting and importing the settings from the primary cluster. On the primary cluster, run the following GPFS command:
`mmfsctl filesystem syncFSconfig -n remotenodefile`

where *remotenodefile* is the list of hosts in the secondary cluster.

3. Suspend the I/O write operations on the primary database by using the following command:

`db2 set write suspend for database`

While the database is in suspended state, you should not be running other utilities or tools. You should be only making a copy of the database. You can optionally flush all buffer pools before you issue **SET WRITE SUSPEND** to minimize the recovery window. This can be achieved by using the FLUSH BUFFERPOOLS ALL statement.

4. Determine which file systems must be suspended and copied by using the following command:

`db2cluster -cfs -list -filesystem`

5. Suspend each GPFS file system that contains container data or log data by using the following command:

`/usr/lpp/mmfs/bin/mmfsctl filesystem suspend-write`

where *filesystem* represents a file system that contains data or log data.

Note: While the GPFS file systems are suspended, write operations are blocked. You should only be performing the split mirror operations during this period to minimize the amount of time that operations are blocked.

6. Create one or multiple split mirrors from the primary database by using the appropriate operating system-level and storage-level commands.

Note:

- Ensure that you copy the entire database directory, including the volume directory. You must also copy the log directory and any container directories that exist outside the database directory. To gather this information, refer to the DBPATHS administrative view, which shows all the files and directories of the database that need to be split.

- If you specified EXCLUDE LOGS with the **SET WRITE** command, do not include the log files in the copy.
7. Resume the GPFS file systems that were suspended by using the following command for each suspended file system:
`/usr/lpp/mmfs/bin/mmfsctl filesystem resume`

where *filesystem* represents a suspended file system that contains data or log data.

8. Resume the I/O write operations on the primary database by using the following command:

```
db2 set write resume for database
```

Assuming that a situation requires you to restore the database by using the split mirror as the backup image, perform the following steps:

- Stop the primary database instance by using the following command:
`db2stop`
- List the cluster manager domain by using the following command:
`db2cluster -cm -list -domain`
- Stop the cluster manager on each host in the cluster by using the following command:
`db2cluster -cm -stop -host host -force`

Note: The last host which you shut down must be the host from which you are issuing this command.

- Stop the GPFS cluster on the primary database instance by using the following command:
`db2cluster -cfs -stop -all`
- Copy the split-off data off the primary database by using appropriate operating system-level commands.

Important: Do not copy the split-off log files, because the original logs are needed for rollforward recovery.

- Start the GPFS cluster on the primary database instance by using the following command:
`db2cluster -cfs -start -all`
- Start the cluster manager by using the following command:
`db2cluster -cm -start -domain domain`
- Start the database instance by using the following command:
`db2start`
- Initialize the primary database by using the following command:
`db2inidb database_alias as mirror`
- Roll forward the primary database to the end of the logs, or to a point-in-time, and stop.

Backing up to tape

When you back up your database or table space, you must correctly set your block size and your buffer size. This is particularly true if you are using a variable block size (on AIX, for example, if the block size has been set to zero).

There is a restriction on the number of fixed block sizes that can be used when backing up. This restriction exists because DB2 database systems write out the

backup image header as a 4-KB block. The only fixed block sizes DB2 database systems support are 512, 1024, 2048, and 4096 bytes. If you are using a fixed block size, you can specify any backup buffer size. However, you might find that your backup operation will not complete successfully if the fixed block size is not one of the sizes that DB2 database systems support.

If your database is large, using a fixed block size means that your backup operations might take more time than expected to complete. To improve performance, you can use a variable block size.

Note: When using a variable block size, ensure that you have well tested procedures in place that enable you to recover successfully, including explicitly specified buffer sizes for the **BACKUP** and **RESTORE** commands, with backup images that are created using a variable block size.

When using a variable block size, you must specify a backup buffer size that is less than or equal to the maximum limit for the tape devices that you are using. For optimal performance, the buffer size must be equal to the maximum block size limit of the device being used.

Before a tape device can be used on a Windows operating system, the following command must be issued:

```
db2 initialize tape on device using blksize
```

Where:

device is a valid tape device name. The default on Windows operating systems is `\.\.\TAPE0`.

blksize is the blocking factor for the tape. It must be a factor or multiple of 4096. The default value is the default block size for the device.

Restoring from a backup image with variable block size might return an error. If this happens, you might need to rewrite the image using an appropriate block size. Following is an example on AIX:

```
tctl -b 0 -Bn -f /dev/rmt0 read > backup_filename.file
dd if=backup_filename.file of=/dev/rmt0 obs=4096 conv=sync
```

The backup image is dumped to a file called `backup_filename.file`. The **dd** command dumps the image back onto tape, using a block size of 4096.

There is a problem with this approach if the image is too large to dump to a file. One possible solution is to use the **dd** command to dump the image from one tape device to another. This will work as long as the image does not span more than one tape. When using two tape devices, the **dd** command is:

```
dd if=/dev/rmt1 of=/dev/rmt0 obs=4096
```

If using two tape devices is not possible, you might be able to dump the image to a raw device using the **dd** command, and then to dump the image from the raw device to tape. The problem with this approach is that the **dd** command *must* keep track of the number of blocks dumped to the raw device. This number must be specified when the image is moved back to tape. If the **dd** command is used to dump the image from the raw device to tape, the command dumps the entire contents of the raw device to tape. The **dd** utility cannot determine how much of the raw device is used to hold the image.

When using the backup utility, you will need to know the maximum block size limit for your tape devices. Here are some examples:

Device	Attachment	Block Size Limit	DB2 Buffer Size Limit (in 4-KB pages)
8 mm	scsi	131,072	32
3420	s370	65,536	16
3480	s370	61 440	15
3490	s370	61 440	15
3490E	s370	65,536	16
7332 (4 mm) ¹	scsi	262,144	64
3490e	scsi	262,144	64
3590 ²	scsi	2,097,152	512
3570 (magstar MP)		262,144	64

Note:

1. The 7332 does not implement a block size limit. 256 KB is simply a suggested value. Block size limit is imposed by the parent adapter.
2. While the 3590 does support a 2-MB block size, you could experiment with lower values (like 256 KB), provided the performance is adequate for your needs.
3. For information about your device limit, check your device documentation or consult with the device vendor.

Verifying the compatibility of your tape device

On UNIX, Linux, and AIX operating systems only, to determine whether your tape device is supported for backing up your DB2 databases, perform the following procedure:

As the database manager instance owner, run the operating system command **dd** to read from or write to your tape device. If the **dd** command succeeds, then you can back up your DB2 databases using your tape device.

Backing up to named pipes

Support is now available for database backup to (and database restore from) local named pipes on UNIX operating systems.

Before you begin

Both the writer and the reader of the named pipe must be on the same machine. The pipe must exist on a local file system. Because the named pipe is treated as a local device, there is no need to specify that the target is a named pipe.

Procedure

1. Create a named pipe. The following is an AIX example:

```
mkfifo /u/dmcinnis/mypipe
```

2. If this backup image is going to be used by the restore utility, the restore operation must be invoked *before* the backup operation, so that it does not miss any data:

```
db2 restore db sample from /u/dmcinnis/mypipe into mynewdb
```

3. Use this pipe as the target for a database backup operation:

```
db2 backup db sample to /u/dmcinnis/mypipe
```

Backing up partitioned databases

Backing up a database in a partitioned database environment can pose difficulties such as tracking the success of the backup of each database partition, managing the multiple log files and backup images, and ensuring the log files and backup images for all the database partitions span the minimum recovery time that is required to restore the database.

Using a single system view (SSV) backup is the easiest way to back up a partitioned database.

About this task

There are four ways to back up a database in a partitioned database environment:

- Back up each database partition one at a time by using the **BACKUP DATABASE** command, the **BACKUP DATABASE** command with the ADMIN_CMD procedure, or the db2Backup API.
- Use the **db2_a11** command with the **BACKUP DATABASE** command to first back up the catalog partition and then to back up a specified list of database partitions.
- Run a single system view (SSV) backup to back up some or all of the database partitions simultaneously, including the catalog partition.
- Use a task assistant in IBM Data Studio to guide you through the process of backing up the database.

Backing up each database partition one at a time is time-consuming and error-prone. Backing up all the partitions by using the **db2_a11** command is easier than backing up each database partition individually because you generally only must make one command call. However, when you use **db2_a11** to back up a partitioned database, you sometimes still must make multiple calls to **db2_a11** because the database partition that contains the catalog cannot be backed up simultaneously with non-catalog database partitions. Whether you back up each database partition one at a time or use **db2_a11**, managing backup images that were created using either of these methods is difficult because the time stamp for each database partition's backup image is different, and coordinating the minimum recovery time across the database partitions' backup images is difficult as well.

For the previously mentioned reasons, the recommended way to back up a database in a partitioned database environment is to use an SSV backup because you can decide to back up all database partitions simultaneously, including the catalog partition, and get the same time stamp for each database partition backup. Alternatively, you can split your backup, specifying some database partitions for which you get the same time stamp, and later take additional backups on the other database partitions to complete the database backup. The catalog partition can be backed up at any time with any other database partitions.

Note: For restore operations, you still must restore the catalog partition before you restore some or all of the other database partitions.

Procedure

To back up some or all of the database partitions of a partitioned database simultaneously by using an SSV backup:

1. Optional: Allow the database to remain online, or take the database offline.
You can back up a partitioned database while the database is online or offline. If the database is online, the backup utility acquires shared connections to the other database partitions, so user applications are able to connect to database partitions while they are being backed up.
2. On the database partition that contains the database catalog, perform the backup with appropriate parameters for partitioned databases, using one of the following methods:
 - Run the **BACKUP DATABASE** command with the **ON DBPARTITIONNUMS** parameter.
 - Run the **BACKUP DATABASE** command with the **ON DBPARTITIONNUMS** parameter by using the **ADMIN_CMD** procedure.
 - Call the db2Backup API with the **iAllNodeFlag** parameter.
 - Open the task assistant for the **BACKUP DATABASE** command in IBM Data Studio.
3. Optional: Include the log files that are required for recovery with the backup images.
By default, log files are included with backup images if you are performing an SSV backup (that is, if you specify the **ON DBPARTITIONNUM** parameter). If you do not want log files to be included with the backup images, use the **EXCLUDE LOGS** command parameter when you run the backup. Log files are excluded from the backup image by default for non-SSV backups.
For more information, see “Including log files with a backup image” on page 293.
4. Optional: Delete previous backup images. The method that you use to delete old backup images depends on how you store the backup images. For example, if you store the backup images to disk, you can delete the files; if you store the backup images using Tivoli Storage Manager, you can use the **db2adut1** utility to delete the backup images. If you are using DB2 Advanced Copy Services (ACS), you can use the **db2acsutil** to delete snapshot backup objects.

Backup and restore operations in a DB2 pureScale environment

In a DB2 pureScale environment, issuing a single **BACKUP DATABASE** or **RESTORE DATABASE** command on any member initiates a backup or restore operation on behalf of all members.

Because a DB2 pureScale environment can have only one database partition, a backup operation has only one set of data to process and produces only one backup image for the entire group. In the case of the other members, only the database metadata and transaction logs must be processed, and those are included in the single backup image.

A backup image includes data from the specified table spaces and any required metadata and configuration information for all currently defined members. You do not have to perform additional backup operations on any other member in the DB2 pureScale instance. Moreover, you require only a single **RESTORE DATABASE** command to restore the database and the member-specific metadata for all members. You do not have to perform additional restore operations on any other

member to restore the cluster. The time stamps of consecutive backup images are unique, increasing values, regardless of which member produced them.

All members must be consistent before an offline backup operation can be attempted. Only one offline backup operation can run at one time, because the backup utility acquires super-exclusive access to the database across all members. Although concurrent online backup operations are supported, different backup operations cannot copy the same table spaces simultaneously, and must wait their turn.

All of the reading of data and metadata from the database and all of the writing to a backup image takes place on a single member. Interactions between the backup or restore operation and other members are limited to copying or updating database metadata (such as table space definitions, the log file header, and the database configuration).

Note: Before taking a backup, you need to ensure that the log archiving path is set to a shared directory so that all the members are able to access the logs for subsequent rollforward operations. If the archive path is not accessible from the member on which the rollforward is being executed, SQL1273N is returned. The following command is an example of how to set the log path to the shared directory:

```
db2 update db cfg using logarchmeth1  
DISK:/db2fs/gpfs1/svtdbm5/svtdbm5/ArchiveLOGS
```

(where *gpfs1* is the shared directory for the members and *ArchiveLOGS* is the actual directory that archives the logs.)

Online backup operations can proceed successfully if another member is offline, goes offline, or comes back online while the operation is executing (Table 24). Although database restore operations are not affected by the state of other members, backup operations might have to wait for a short duration while member crash recovery is completed on an offline and inconsistent member.

Table 24. Effect of the state of other members in a DB2 pureScale instance on database backup and restore operations

Operation	State of other members	
	Offline and consistent	Offline and inconsistent
Online backup	The backup operation succeeds. The other member cannot become active while the backup utility is accessing the log file header (LFH) near the beginning of the backup operation or while the backup utility is accessing the log stream near the end of the backup operation.	The backup operation succeeds, but it must wait for member crash recovery to be completed and for the other member to become either active or consistent. The other member cannot become active while the backup utility is accessing the LFH near the beginning of the backup operation or while the backup utility is accessing the log stream near the end of the backup operation.
Restore	The restore operation is completed normally.	The restore operation is completed normally.

Image and archive naming

File names for backup images that you create on disk consist of a concatenation of several elements, separated by periods:

DB_alias.Type.Inst_name.DBPARTnnn.Timestamp.Seq_num

DB_alias

The database alias name that you specified when you invoked the backup utility.

Type The type of backup operation, where 0 represents a full database backup, 3 represents a table space backup, and 4 represents a backup image generated by the **LOAD** command with the **COPY NO** option.

Inst_name

The name of the current instance, which is the value of the **DB2INSTANCE** environment variable.

nnn The database partition number. In a DB2 pureScale environment, the number is always 000.

Timestamp

A 14-character representation of the date and time when you performed the backup operation. The time stamp is in the form *yyyymmddhhnnss*, where:

- *yyyy* represents the year.
- *mm* represents the month (01 to 12).
- *dd* represents the day of the month (01 to 31).
- *hh* represents the hour (00 to 23).
- *nn* represents the minutes (00 to 59).
- *ss* represents the seconds (00 to 59).

Seq_num

A 3-digit number used as a file extension.

For example:

SAMPLE.0.kroger.DBPART000.200802241234.001

Online backup with INCLUDE LOGS

An online backup operation with the **INCLUDE LOGS** option (the default) produces a backup image that includes the range of log files required to restore and roll the database forward to its minimum recovery time. If this backup image is then used to restore to a new database (perhaps during disaster recovery), and only the logs from the backup image are available during a subsequent roll-forward operation, a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** parameter often returns an error message about a missing log file (SQL1273N). This is expected in some situations, because the database manager might have detected that additional logs were written after the backup operation, but that those logs are not available for the current roll-forward operation. It might also be the case that one or more of the logs that are necessary to roll the database forward to a consistent point in time are missing. In either case, verify that the end point of the roll-forward operation is acceptable and then issue a **ROLLFORWARD DATABASE** with the **AND STOP** parameter. If the roll-forward operation has reached its minimum recovery time despite the missing log file, the **ROLLFORWARD DATABASE** with the **AND STOP** parameter should complete successfully; otherwise, it returns SQL1276N (the roll-forward operation did not reach its minimum recovery time using this backup image).

Disaster recovery and high availability through log shipping in a DB2 pureScale environment

Log shipping is the process of copying whole log files to a standby machine, either from an archive device, or through a user exit program running against the primary database. You can choose to keep a standby database up-to-date by applying the logs to it as they are archived, or you can keep the database or table space backup images and log archives on the standby site, and perform restore and roll-forward operations only after a disaster has occurred. In either case, the roll-forward operation on the standby site might detect that one or more log files are missing and return SQL1273N. Verify that the roll-forward operation reached an acceptable time stamp, or take appropriate action to correct the problem.

If, during a log stream merge operation, the DB2 database manager determines that there is a missing log file in one of the log streams, an error is returned. The roll-forward utility returns SQL1273N; the db2ReadLog API returns SQL2657N. If you choose to keep a standby database up-to-date by applying logs to it as they are archived, roll-forward operations might frequently detect that some logs are missing.

Figure 55 shows an example of how two members could write log records to the log files in their active log stream. Each log file is represented by a box. Consider a scenario where both a primary and standby site have been set up for high availability. A **ROLLFORWARD DATABASE** command with the **END OF LOGS** option is attempted on the standby site at time points A, B and C. For any particular point in time, any log files that have been closed before that time have been archived and are accessible on the standby. Otherwise, the log file is still active on the primary and is not available to the standby yet (as shown for log file 4 on log stream 1 at time B).

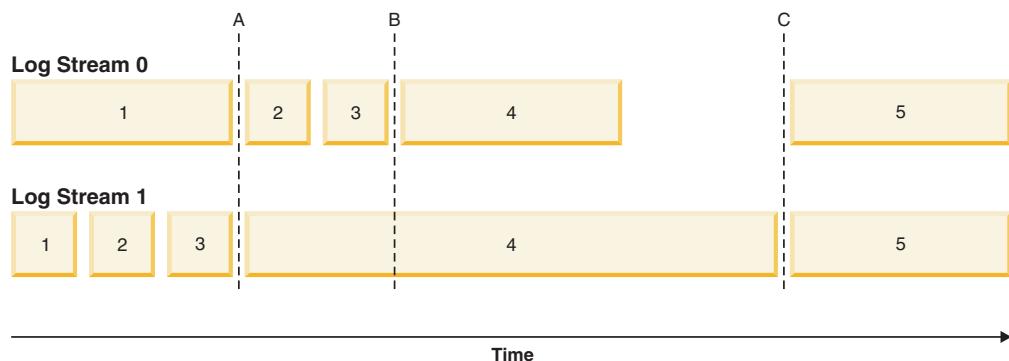


Figure 55. Log files in a DB2 pureScale environment

At time A, the **ROLLFORWARD DATABASE** command will complete successfully as log file 1 from log stream 0 was closed and archived at the same time as log file 3 from log stream 1. At time B however, the **ROLLFORWARD DATABASE** command will return v. This happens because at the time that the command is issued on the standby site, the standby site has access to log files 2 and 3 from log stream 0, but not to log file 4 from log stream 1 because the log file is still open and active on the primary site. Furthermore, since the log records in files 2 and 3 on log stream 0 were written during the same time period as the beginning of log file 4 on log stream 1, the roll-forward operation cannot process log files 2 and 3 until log file 4 from log stream 1 is made available. At time C, when log file 4 is finally closed and archived on log stream 1, a **ROLLFORWARD DATABASE** command will complete successfully. It is possible to force the truncation and archiving of files across all

the log streams using the **ARCHIVE LOG** command, or by deactivating the database across all members. In the case of the **ARCHIVE LOG** command, the current log file on each log stream is truncated independently and there is no guarantee that it will happen at the exact same point in time across all members. Therefore, even if the **ARCHIVE LOG** command is issued, it is still possible to get an SQL1273N error when executing the **ROLLFORWARD DATABASE** command.

While missing log conditions are common and expected when using log shipping in a DB2 pureScale environment, in most cases, each roll-forward operation on the standby will make additional progress over the last **ROLLFORWARD DATABASE** command (even when SQL1273N is returned) and therefore the error itself should often be expected. It is possible, however, for the primary site to have trouble archiving a file for one log stream while successfully archiving logs for the other log streams. This could be the result of a temporary problem accessing the archive storage for one log stream. Such problems can cause the log merge and replay on the standby to be held up, increasing the number of transactions that could be lost in the event of a disaster. To ensure that your standby system is up-to-date, issue a **ROLLFORWARD DATABASE** command with the **QUERY STATUS** parameter after each roll-forward operation that returns SQL1273N and verify that progress is being made over time. If a roll-forward operation on the standby is not making progress over an extended period of time, determine why the log file reported as missing is not available on the standby system and correct the problem. The **ARCHIVE LOG** command can be used to truncate the log files that are currently being updated on each member, making them eligible for archiving and subsequent replay on the standby system.

In the event of a disaster (for example, fire, earthquake, vandalism, or other catastrophic events) your plan for recovery might be to execute a roll-forward operation through all remaining logs, or a restore and roll-forward operation through all available logs. As mentioned previously, the roll-forward operation might detect that one or more log file is missing, because log files were written on the primary but not yet archived at the time of the disaster (SQL1273N). It is also possible that a log that was archived cannot be found by the roll-forward utility for some unexpected reason; this can also cause the roll-forward utility to return SQL1273N. It is important to validate the end point of a roll-forward operation by using the **ROLLFORWARD DATABASE** command with the **QUERY STATUS** parameter, and to decide whether or not the missing log condition is expected. If the missing log condition is expected, or the end point is acceptable, you can issue a **ROLLFORWARD DATABASE** command with the **STOP** parameter to complete the roll-forward recovery process.

Restrictions

Backup and restore operations between an environment where the DB2 pureScale Feature is installed and an environment where the DB2 pureScale Feature is not installed are not supported.

After a change in topology that involves adding or dropping a member, you cannot perform roll-forward recovery operations through the point where the topology change occurred. If you add or drop a member, the database is placed in backup pending state, and you must perform a full database backup operation before a connection to the database can be made. To recover, restore this backup image and roll forward to the end of the logs. If you must restore a backup image from before the topology change, you will only be able to roll forward to the point at which the topology change occurred. This can be accomplished by issuing a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** parameter (which

returns SQL1546N) followed by a **ROLLFORWARD DATABASE** command with the **STOP** parameter. This operation will not recover any transactions that changed the database after the topology change.

In a DB2 pureScale environment, the **ON ALL DBPARTITIONNUMS** parameter and the **ON DBPARTITION (0)** parameter of the **BACKUP DATABASE** command are valid. If you specify a database partition number other than 0, however, an error (SQL0270N) is returned because no other database partitions exist.

The following restriction applies to this release:

- A database, which resides outside of a DB2 pureScale environment, can be migrated to a DB2 pureScale environment. You cannot use database restore operations to migrate such database to a DB2 pureScale environment.
- Delta and incremental backup operations are not supported.
- In Version 10 GA and Fix Pack 1, snapshot backup operations using DB2 Advanced Copy Services (ACS) are not supported. In Version 10 Fix Pack 2, this restriction is removed.

Examples

- Back up a 4-member database named SAMPLE from any member:
`BACKUP DB SAMPLE`
- Restore a 1-member database named SAMPLE:
`RESTORE DB SAMPLE`
- Use the **RECOVER DATABASE** command to restore and roll forward a database named SAMPLE from any member:
`RECOVER DB SAMPLE TO END OF LOGS`

If the database does not exist, use the **RESTORE DATABASE** and **ROLLFORWARD DATABASE** commands instead of the **RECOVER DATABASE** command because an existing database with a complete database history is required for the successful completion of the **RECOVER DATABASE** command.

Enabling automatic backup

A database can become unusable due to a wide variety of hardware or software failures. Ensuring that you have a recent, full backup of your database is an integral part of planning and implementing a disaster recovery strategy for your system.

Use automatic database backup as part of your disaster recovery strategy to enable DB2 to back up your database both properly and regularly.

About this task

You can configure automatic backup using the command line interface, or the **AUTOMAINT_SET_POLICY** system stored procedure. You also need to enable the health indicator `db.db_backup_req`, which by default is enabled. Note that only an active database is considered for the evaluation.

Procedure

- To configure automatic backup using the command line interface, set each of the following database configuration parameters to ON:
 - `AUTO_MAINT`

- **AUTO_DB_BACKUP**
 - To configure automatic backup using IBM Data Studio, right-click the database and select the task assistant to configure automatic backup.
 - To configure automatic backup using the AUTOMAINT_SET_POLICY system stored procedure:
 1. Create configuration XML input specifying details like backup media, whether the backup should be online or offline, and frequency of the backup. You can copy the contents of the sample file called DB2DefaultAutoBackupPolicy.xml in the SQLLIB/samples/automaintcfg directory and modify the XML to satisfy your configuration requirements.
 2. Optional: Create an XML input file containing your configuration XML input.
 3. Call AUTOMAINT_SET_POLICY with the following parameters:
 - maintenance type: AutoBackup
 - configuration XML input: either a BLOB containing your configuration XML input text; or the name of the file containing your configuration XML input.
- See the topic “Configuring an automated maintenance policy using SYSPROC.AUTOMAINT_SET_POLICY or SYSPROC.AUTOMAINT_SET_POLICYFILE” for more information about using the AUTOMAINT_SET_POLICY system stored procedure.

Configuring an automated maintenance policy using SYSPROC.AUTOMAINT_SET_POLICY or SYSPROC.AUTOMAINT_SET_POLICYFILE

You can use the system stored procedures AUTOMAINT_SET_POLICY and AUTOMAINT_SET_POLICYFILE to configure the automated maintenance policy for a database.

Procedure

To configure the automated maintenance policy for a database, perform the following steps:

1. Connect to the database
2. Call AUTOMAINT_SET_POLICY or AUTOMAINT_SET_POLICYFILE
 - The parameters required for AUTOMAINT_SET_POLICY are:
 - a. Maintenance type, specifying the type of automated maintenance activity to configure.
 - b. Pointer to a BLOB that specifies the automated maintenance policy in XML format.
 - The parameters required for AUTOMAINT_SET_POLICYFILE are:
 - a. Maintenance type, specifying the type of automated maintenance activity to configure.
 - b. The name of an XML file that specifies the automated maintenance policy.

Valid maintenance type values are:

- AUTO_BACKUP - automatic backup
- AUTO_REORG - automatic table and index reorganization
- AUTO_RUNSTATS - automatic table **RUNSTATS** operations
- MAINTENANCE_WINDOW - maintenance window

What to do next

You can use the system stored procedures AUTOMAINT_GET_POLICY and AUTOMAINT_GET_POLICYFILE to retrieve the automated maintenance policy configured for a database.

Monitoring backup operations

You can use the **LIST UTILITIES** command to monitor the progress of backup operations on a database.

Procedure

Issue the **LIST UTILITIES** command and specify the **SHOW DETAIL** parameter:

```
list utilities show detail
```

Results

For backup operations, an initial estimate of the number of bytes to be processed will be specified. As the backup operation progresses the number of bytes to be processed will be updated. The bytes shown does not correspond to the size of the image and should not be used as an estimate for backup image size. The actual image might be much smaller depending on whether it is an incremental or compressed backup.

Example

The following is an example of the output for monitoring the performance of an offline database backup operation:

ID	= 3
Type	= BACKUP
Database Name	= SAMPLE
Partition Number	= 0
Description	= offline db
Start Time	= 08/04/2011 12:16:23.248367
State	= Executing
Invocation Type	= User
Throttling:	
Priority	= Unthrottled
Progress Monitoring:	
Estimated Percentage Complete	= 31
Total Work	= 123147277 bytes
Completed Work	= 37857269 bytes
Start Time	= 08/04/2011 12:16:23.248377

Optimizing backup performance

When you perform a backup operation, the DB2 database manager automatically chooses an optimal value for the number of buffers, the buffer size, and the parallelism settings. The values are based on the amount of utility heap memory available, the number of processors available, and the database configuration.

Therefore, depending on the amount of storage available on your system, consider allocating more memory by increasing the **util_heap_sz** configuration parameter.

The objective is to minimize the time it takes to complete a backup operation. Unless you explicitly enter a value for the following **B_{AC}KUP DATABASE** command parameters, the DB2 database manager selects one for them:

- **WITH** *num-buffers* **BUFFERS**
- **PARALLELISM** *n*
- **BUFFER** *buffer-size*

If the number of buffers and the buffer size are not specified, resulting in the DB2 database manager setting the values, it should have minimal effect on large databases. However, for small databases, it can cause a large percentage increase in backup image size. Even if the last data buffer written to disk contains little data, the full buffer is written to the image anyway. In a small database, this means that a considerable percentage of the image size might be empty.

You can also choose to do any of the following to reduce the amount of time required to complete a backup operation:

- Specify table space backup.

You can back up (and subsequently recover) part of a database by using the **TABLESPACE** option on the **BACKUP DATABASE** command. This facilitates the management of table data, indexes, and long field or large object (LOB) data in separate table spaces.

- Increase the value of the **PARALLELISM** parameter on the **BACKUP DATABASE** command so that it reflects the number of table spaces being backed up.

The **PARALLELISM** parameter defines the number of processes or threads that are started to read data from the database and to compress data during a compressed backup operation. Each process or thread is assigned to a specific table space, so there is no benefit to specifying a value for the **PARALLELISM** parameter that is larger than the number of table spaces being backed up. When it finishes backing up this table space, it requests another. Note, however, that each process or thread requires both memory and CPU overhead.

- Increase the backup buffer size.

The ideal backup buffer size is a multiple of the table space extent size plus one page. If you have multiple table spaces with different extent sizes, specify a value that is a common multiple of the extent sizes plus one page.

- Increase the number of buffers.

Use at least twice as many buffers as backup targets (or sessions) to ensure that the backup target devices do not have to wait for data.

- Use multiple target devices.

Compatibility of online backup and other utilities

Some utilities can be run at the same time as an online backup, but others cannot.

The following utilities are compatible with online backup:

- **EXPORT**
- **INSPECT**

The following SQL statements and utilities are compatible with online backup only under certain circumstances:

- **CREATE INDEX**

In SMS mode, online index create and online backup do not run concurrently due to the ALTER TABLE lock. Online index create acquires it in exclusive mode while online backup acquires it in share.

In DMS mode, online index create and online backup can run concurrently in most cases. There is a possibility if you have a large number of tables in the

same tablespace as the one in which you are creating the index, that the online index create will internally acquire an online backup lock that will conflict with any concurrent online backup.

- REORG INDEX with the ONLINE option

As with online index create, in SMS mode, online index reorganization do not run concurrently with online backup due to the ALTER TABLE lock. Online index reorganization acquires it in exclusive mode while online backup acquires it in share. In addition, an online index reorganization operation, quiesces the table before the switch phase and acquires a Z lock, which prevents an online backup. However, the ALTER TABLE lock should prevent an online backup from running concurrently before the Z table lock is acquired.

In DMS mode, online index reorganization and online backup can run concurrently.

In addition, online index reorganization quiesces the table before the switch phase and gets a Z lock, which prevents an online backup.

- IMPORT

The import utility is compatible with online backup except when the **IMPORT** command is issued with the **REPLACE** parameter, in which case, import gets a Z lock on the table and prevents an online backup from running concurrently.

- TRUNCATE TABLE

The TRUNCATE statement is not compatible with online backup because it gets a Z lock on the table and prevents an online backup from running concurrently.

- ALLOW READ ACCESS LOAD

ALLOW READ ACCESS load operations are not compatible with online backup when the **LOAD** command is issued with the **COPY NO** parameter. In this mode the utilities both modify the table space state, causing one of the utilities to report an error.

ALLOW READ ACCESS load operations are compatible with online backup when the **LOAD** command is issued with the **COPY YES** option, although there might still be some compatibility issues. In SMS mode, the utilities can execute concurrently, but they will hold incompatible table lock modes and consequently might be subject to table lock waits. In DMS mode, the utilities both hold incompatible "Internal-B" (OLB) lock modes and might be subject to waits on that lock. If the utilities execute on the same table space concurrently, the load utility might be forced to wait for the backup utility to complete processing of the table space before the load utility can proceed.

- REORG TABLE with the ONLINE option

The cleanup phase of online table reorganization cannot start while an online backup is running. You can pause the table reorganization, if required, to allow the online backup to finish before resuming the online table reorganization.

You can start an online backup of a DMS table space when a table within the same table space is being reorganized online. There might be lock waits associated with the reorganization operation during the truncate phase.

You cannot start an online backup of an SMS table space when a table within the same table space is being reorganized online. Both operations require an exclusive lock.

- DDLs that require a Z lock (such as ALTER TABLE, DROP TABLE, and DROP INDEX)

Online DMS table space backup is compatible with DDLs that require a Z lock.

Online SMS table space backup must wait for the Z lock to be released.

- Storage group DDLs

If you are modifying the database storage groups by issuing one of the following statements, you should take care to coordinate this operation with your online backup schedule:

- CREATE STOGROUP
- ALTER STOGROUP
- DROP STOGROUP
- RENAME STOGROUP
- ALTER DATABASE

If there is an online backup in progress, the storage group DDL waits behind that operation until it can obtain the appropriate lock, which can potentially take a long time. Similarly, an online backup waits behind any in-progress storage group DDL, until that DDL is committed or rolled back.

- **RUNSTATS** with the ALLOW WRITE or ALLOW READ option

The **RUNSTATS** command is compatible with online backup except when the system catalog table space is an SMS table space. If the system catalog resides in an SMS table space, then the **RUNSTATS** command and the online backup hold incompatible table locks on the table causing lock waits.

- **ALTER TABLESPACE**

Operations that enable or disable autoresize, or alter autoresize containers, are not permitted during an online backup of a table space.

- **ALTER TABLESPACE** with the REBALANCE option

When online backup and rebalancer are running concurrently, online backup pauses the rebalancer and does not wait for it to complete.

The following utilities are not compatible with online backup:

- **REORG TABLE**
- **RESTORE DATABASE**
- **ROLLFORWARD DATABASE**
- **LOAD** with the **ALLOW NO ACCESS** option
- **SET WRITE**
- **BACKUP DATABASE** with the **ONLINE** option

This applies to database-level online backups and table-space-level online backups (if they involve the same table space or table spaces).

Chapter 13. Recovering databases

Recovering a database restores a database and all its storage groups to a specified time, by using information found in the recovery history file.

Before you begin

If you issue the **RECOVER DATABASE** command following an incomplete recover operation that ended during the rollforward phase, the recover utility attempts to continue the previous recover operation, without redoing the restore phase. If you want to force the recover utility to redo the restore phase, issue the **RECOVER DATABASE** command with the **RESTART** option to force the recover utility to ignore any prior recover operation that failed to complete. If you are using the application programming interface (API), specify the caller action **DB2RECOVER_RESTART** for the **iRecoverAction** field to force the recover utility to redo the restore phase.

If the **RECOVER DATABASE** command is interrupted during the restore phase, it cannot be continued. You must reissue the **RECOVER DATABASE** command.

You should not be connected to the database that is to be recovered: the recover database utility automatically establishes a connection to the specified database, and this connection is terminated at the completion of the recover operation.

About this task

The database can be local or remote.

Note: In a partitioned database environment, the recover utility must be invoked from the catalog partition of the database.

Procedure

To invoke the recover utility, use the:

- **RECOVER DATABASE** command, or
- db2Recover application programming interface (API).

Example

The following example shows how to use the **RECOVER DATABASE** command through the CLP:

```
db2 recover db sample
```

Optimizing recovery performance

There are strategies that you can use to improve DB2 performance during database recovery and decrease the time that is required to recover from a DB2 service outage.

The following should be considered when thinking about recovery performance:

- You can improve performance for databases that are frequently updated by placing the logs on a separate device. In the case of an online transaction processing (OLTP) environment, often more I/O is needed to write data to the

logs than to store a row of data. Placing the logs on a separate device will minimize the disk arm movement that is required to move between a log and the database files.

You should also consider what other files are on the disk. For example, moving the logs to the disk used for system paging in a system that has insufficient real memory will defeat your tuning efforts.

DB2 database products automatically attempt to minimize the time it takes to complete a backup or restore operation by choosing an optimal value for the number of buffers, the buffer size and the parallelism settings. The values are based on the amount of utility heap memory available, the number of processors available and the database configuration.

- To reduce the amount of time required to complete a restore operation, use multiple source devices.
- If a table contains large amounts of long field and LOB data, restoring it could be very time consuming. If the database is enabled for rollforward recovery, the **RESTORE** command provides the capability to restore selected table spaces. If the long field and LOB data is critical to your business, restoring these table spaces should be considered against the time required to complete the backup task for these table spaces. By storing long field and LOB data in separate table spaces, the time required to complete the restore operation can be reduced by choosing not to restore the table spaces containing the long field and LOB data. If the LOB data can be reproduced from a separate source, choose the NOT LOGGED option when creating or altering a table to include LOB columns. If you choose not to restore the table spaces that contain long field and LOB data, but you need to restore the table spaces that contain the table, you must roll forward to the end of the logs so that all table spaces that contain table data are consistent.

Note: If you back up a table space that contains table data without the associated long or LOB fields, you cannot perform point-in-time rollforward recovery on that table space. All the table spaces for a table must be rolled forward simultaneously to the same point in time.

- The following apply for both backup and restore operations:
 - Multiple devices should be used.
 - Do not overload the I/O device controller bandwidth.
- DB2 database products use multiple agents to perform both crash recovery and database rollforward recovery. You can expect better performance during these operations, particularly on symmetric multi-processor (SMP) machines; using multiple agents during database recovery takes advantage of the extra CPUs that are available on SMP machines.

The agent type introduced by parallel recovery is **db2agnsc**. DB2 database managers choose the number of agents to be used for database recovery based on the number of CPUs on the machine.

DB2 database managers distribute log records to these agents so that they can be reapplied concurrently, where appropriate. For example, the processing of log records associated with insert, delete, update, add key, and delete key operations can be parallelized in this way. Because the log records are parallelized at the page level (log records on the same data page are processed by the same agent), performance is enhanced, even if all the work was done on one table.

- When you perform a recover operation, DB2 database managers will automatically choose an optimal value for the number of buffers, the buffer size and the parallelism settings. The values will be based on the amount of utility heap memory available, the number of processors available and the database

configuration. Therefore, depending on the amount of storage available on your system, you should consider allocating more memory by increasing the `util_heap_sz` configuration parameter.

Recovering data using db2adutl

You can perform cross-node recovery using the `db2adutl` command, `logarchopt1` and `vendoropt` database configuration parameters. This recovery is demonstrated in examples from a few different Tivoli Storage Manager (TSM) environments.

For the following examples, computer 1 is called bar and is running the AIX operating system. The user on this machine is roecken. The database on bar is called zample. Computer 2 is called dps. This computer is also running the AIX operating system, and the user is regress9.

Example 1: TSM server manages passwords automatically (PASSWORDACCESS option set to GENERATE)

This cross-node recovery example shows how to set up two computers so that you can recover data from one computer to another when log archives and backups are stored on a TSM server and where passwords are managed using the `PASSWORDACCESS=GENERATE` option.

Note: After updating the database configuration, you might have to take an offline backup of the database.

1. To enable the database for log archiving for the bar computer to the TSM server, update the database configuration parameter `logarchmeth1` for the zample database using the following command:

```
bar:/home/roecken> db2 update db cfg for zample using LOGARCHMETH1 tsm
```

The following information is returned:

```
DB20000I The UPDATE DATABASE CONFIGURATION command completed successfully.
```

2. Disconnect all users and applications from the database using the following command:

```
db2 force applications all
```

3. Verify that there are no applications connected to the database using the following command:

```
db2 list applications
```

You should receive a message that says that no data was returned.

Note: In a partitioned database environment, you must perform this step on all database partitions.

4. Create a backup of the database on the TSM server using the following command:

```
db2 backup db zample use tsm
```

Information similar to the following is returned:

```
Backup successful. The timestamp for this backup image is : 20090216151025
```

Note: In a partitioned database environment, you must perform this step on all database partitions. The order in which you perform this step on the database partitions differs depending on whether you are performing an online backup or an offline backup. For more information, see Chapter 12, “Backing up databases,” on page 301.

5. Connect to the zample database using the following command:

```
db2 connect to zample
```

6. Generate new transaction logs for the database by creating a table and loading data into the TSM server using the following command:

```
bar:/home/roecken> db2 load from mr of del modified by noheader replace  
into employee copy yes use tsm
```

where in this example, the table is called employee, and the data is being loaded from a delimited ASCII file called mr. The **COPY YES** option is specified to make a copy of the data that is loaded, and the **USE TSM** option specifies that the copy of the data is stored on the TSM server.

Note: You can specify the **COPY YES** option only if the database is enabled for roll-forward recovery; that is, the **logarchmeth1** database configuration parameter must be set to **USEREXIT**, **LOGRETAIN**, **DISK**, or **TSM**.

To indicate its progress, the load utility returns a series of messages:

```
SQL3109N The utility is beginning to load data from file "/home/roecken/mr".
```

```
SQL3500W The utility is beginning the "LOAD" phase at time "02/16/2009  
15:12:13.392633".
```

```
SQL3519W Begin Load Consistency Point. Input record count = "0".
```

```
SQL3520W Load Consistency Point was successful.
```

```
SQL3110N The utility has completed processing. "1" rows were read from the  
input file.
```

```
SQL3519W Begin Load Consistency Point. Input record count = "1".
```

```
SQL3520W Load Consistency Point was successful.
```

```
SQL3515W The utility has finished the "LOAD" phase at time "02/16/2009  
15:12:13.445718".
```

```
Number of rows read      = 1  
Number of rows skipped   = 0  
Number of rows loaded    = 1  
Number of rows rejected  = 0  
Number of rows deleted   = 0  
Number of rows committed = 1
```

7. After the data has been loaded into the table, confirm that there is one backup image, one load copy image, and one log file on the TSM server by running the following query on the zample database:

```
bar:/home/roecken/sql1lib/adsm> db2adutl query db zample
```

The following information is returned:

```
Retrieving FULL DATABASE BACKUP information.  
1 Time: 20090216151025 Oldest log: S0000000.LOG Log stream: 0  
Sessions: 1
```

```
Retrieving INCREMENTAL DATABASE BACKUP information.  
No INCREMENTAL DATABASE BACKUP images found for ZAMPLE
```

```
Retrieving DELTA DATABASE BACKUP information.  
No DELTA DATABASE BACKUP images found for ZAMPLE
```

```
Retrieving TABLESPACE BACKUP information.  
No TABLESPACE BACKUP images found for ZAMPLE
```

```
Retrieving INCREMENTAL TABLESPACE BACKUP information.  
No INCREMENTAL TABLESPACE BACKUP images found for ZAMPLE
```

```
Retrieving DELTA TABLESPACE BACKUP information.  
No DELTA TABLESPACE BACKUP images found for ZAMPLE
```

```
Retrieving LOAD COPY information.  
1 Time: 20090216151213
```

```
Retrieving LOG ARCHIVE information.  
Log file: S0000000.LOG, Chain Num: 0, Log stream: 0,  
Taken at: 2009-02-16-15.10.38
```

8. To enable cross-node recovery, you must give access to the objects associated with the bar computer to another computer and account. In this example, give access to the computer dps and the user regress9 using the following command:

```
bar:/home/roecken/sql1lib/adsm> db2adut1 grant user regress9  
on nodename dps for db zample
```

The following information is returned:

```
Successfully added permissions for regress9 to access ZAMPLE on node dps.
```

Note: You can confirm the results of the **db2adut1** grant operation by issuing the following command to retrieve the current access list for the current node:

```
bar:/home/roecken/sql1lib/adsm> db2adut1 queryaccess
```

The following information is returned:

Node	Username	Database Name	Type
DPS	regress9	ZAMPLE	A

Access Types: B - backup images L - logs A - both

9. In this example, computer 2, dps, is not yet set up for cross-node recovery of the zample database. Verify that there is no data associated with this user and computer on the TSM server using the following command:

```
dps:/home/regress9/sql1lib/adsm> db2adut1 query db zample
```

The following information is returned:

```
--- Database directory is empty ---  
Warning: There are no file spaces created by DB2 on the ADSM server  
Warning: No DB2 backup images found in ADSM for any alias.
```

10. Query the TSM server for a list of objects for the zample database associated with user roecken and computer bar using the following command:

```
dps:/home/regress9/sql1lib/adsm> db2adut1 query db zample nodename  
bar owner roecken
```

The following information is returned:

```
--- Database directory is empty ---
```

```
Query for database ZAMPLE
```

```
Retrieving FULL DATABASE BACKUP information.  
1 Time: 20090216151025 Oldest log: S0000000.LOG Log stream: 0  
Sessions: 1
```

```
Retrieving INCREMENTAL DATABASE BACKUP information.  
No INCREMENTAL DATABASE BACKUP images found for ZAMPLE
```

```
Retrieving DELTA DATABASE BACKUP information.  
No DELTA DATABASE BACKUP images found for ZAMPLE
```

```
Retrieving TABLESPACE BACKUP information.  
No TABLESPACE BACKUP images found for ZAMPLE
```

```
Retrieving INCREMENTAL TABLESPACE BACKUP information.  
No INCREMENTAL TABLESPACE BACKUP images found for ZAMPLE
```

```
Retrieving DELTA TABLESPACE BACKUP information.  
No DELTA TABLESPACE BACKUP images found for ZAMPLE
```

```
Retrieving LOAD COPY information.  
1 Time: 20090216151213
```

```

Retrieving LOG ARCHIVE information.
Log file: S0000000.LOG, Chain Num: 0, Log stream: 0,
Taken at: 2009-02-16-15.10.38

```

This information matches the TSM information that was generated previously and confirms that you can restore this image onto the dps computer.

11. Restore the zample database from the TSM server to the dps computer using the following command:

```
dps:/home/regress9> db2 restore db zample use tsm options
"!'-fromnode=bar -fromowner=roecken!" without prompting
```

The following information is returned:

```
DB20000I The RESTORE DATABASE command completed successfully.
```

Note: If the zample database already existed on dps, the **OPTIONS** parameter would be omitted, and the database configuration parameter **vendoropt** would be used. This configuration parameter overrides the **OPTIONS** parameter for a backup or restore operation.

12. Perform a roll-forward operation to apply the transactions recorded in the zample database log file when a new table was created and new data loaded. In this example, the following attempt for the roll-forward operation will fail because the roll-forward utility cannot find the log files because the user and computer information is not specified:

```
dps:/home/regress9> db2 rollforward db zample to end of logs and stop
```

The command returns the following error:

```
SQL4970N Roll-forward recovery on database "ZAMPLE" cannot reach the
specified stop point (end-of-log or point-in-time) because of missing log
file(s) on node(s) "0".
```

Force the roll-forward utility to look for log files associated with another computer using the proper **logarchopt** value. In this example, use the following command to set the **logarchopt1** database configuration parameter and search for log files associated with user roecken and computer bar:

```
dps:/home/regress9> db2 update db cfg for zample using logarchopt1
"!'-fromnode=bar -fromowner=roecken!"
```

13. Enable the roll-forward utility to use the backup and load copy images by setting the **vendoropt** database configuration parameter using the following command:

```
dps:/home/regress9> db2 update db cfg for zample using VENDOROPT
"!'-fromnode=bar -fromowner=roecken!"
```

14. You can finish the cross-node data recovery by applying the transactions recorded in the zample database log file using the following command:

```
dps:/home/regress9> db2 rollforward db zample to end of logs and stop
```

The following information is returned:

Rollforward Status				
Input database alias = zample Number of members have returned status = 1				
Member number	Rollforward status	Next log to be read	Log files processed	Last committed transaction
0	not pending		S0000000.LOG-S0000000.LOG	2009-05-06-15.28.11.000000 UTC

```
DB20000I The ROLLFORWARD command completed successfully.
```

The database zample on computer dps under user regress9 has been recovered to the same point as the database on computerbar under user roecken.

Example 2: Passwords are user-managed (PASSWORDACCESS option set to PROMPT)

This cross-node recovery example shows how to set up two computers so that you can recover data from one computer to another when log archives and backups are stored on a TSM server and where passwords are managed by the users. In these environments, extra information is required, specifically the TSM nodename and password of the computer where the objects were created.

1. Update the client dsm.sys file by adding the following line because computer bar is the name of the source computer

```
NODENAME bar
```

Note: On Windows operating systems, this file is called the dsm.opt file. When you update this file, you must reboot your system for the changes to take effect.

2. Query the TSM server for the list of objects associated with user roecken and computer bar using the following command:

```
dps:/home/regress9/sql1lib/adsm> db2adut1 query db zample nodename bar  
owner roecken password *****
```

The following information is returned:

```
Query for database ZAMPLE

Retrieving FULL DATABASE BACKUP information.  
1 Time: 20090216151025 Oldest log: S0000000.LOG Log stream: 0  
Sessions: 1

Retrieving INCREMENTAL DATABASE BACKUP information.  
No INCREMENTAL DATABASE BACKUP images found for ZAMPLE

Retrieving DELTA DATABASE BACKUP information.  
No DELTA DATABASE BACKUP images found for ZAMPLE

Retrieving TABLESPACE BACKUP information.  
No TABLESPACE BACKUP images found for ZAMPLE

Retrieving INCREMENTAL TABLESPACE BACKUP information.  
No INCREMENTAL TABLESPACE BACKUP images found for ZAMPLE

Retrieving DELTA TABLESPACE BACKUP information.  
No DELTA TABLESPACE BACKUP images found for ZAMPLE

Retrieving LOAD COPY information.  
1 Time: 20090216151213

Retrieving LOG ARCHIVE information.  
Log file: S0000000.LOG, Chain Num: 0, Log stream: 0,  
Taken at: 2009-02-16-15.10.38
```

3. If the zample database does not exist on computer dps, perform the following steps:

- a. Create an empty zample database using the following command:

```
dps:/home/regress9> db2 create db zample
```

- b. Update the database configuration parameter **tsm_nodename** using the following command:

```
dps:/home/regress9> db2 update db cfg for zample using tsm_nodename bar
```

- c. Update the database configuration parameter **tsm_password** using the following command:

```
dps:/home/regress9> db2 update db cfg for zample using  
tsm_password *****
```

4. Attempt to restore the zample database using the following command:

```
dps:/home/regress9> db2 restore db zample use tsm options  
"'-fromnode=bar -fromowner=roecken'" without prompting
```

The restore operation completes successfully, but a warning is issued:

```
SQL2540W Restore is successful, however a warning "2523" was  
encountered during Database Restore while processing in No  
Interrupt mode.
```

5. Perform a roll-forward operation using the following command:

```
dps:/home/regress9> db2 rollforward db zample to end of logs and stop
```

In this example, because the restore operation replaced the database configuration file, the roll-forward utility cannot find the correct log files and the following error message is returned:

```
SQL1268N Roll-forward recovery stopped due to error "-2112880618"  
while retrieving log file "S0000000.LOG" for database "ZAMPLE" on node "0".
```

Reset the following TSM database configuration values to the correct values:

- a. Set the **tsm_nodename** configuration parameter using the following command:

```
dps:/home/regress9> db2 update db cfg for zample using tsm_nodename bar
```

- b. Set the **tsm_password** database configuration parameter using the following command:

```
dps:/home/regress9> db2 update db cfg for zample using tsm_password *****
```

- c. Set the **logarchopt1** database configuration parameter so that the roll-forward utility can find the correct log files using the following command:

```
dps:/home/regress9> db2 update db cfg for zample using logarchopt1  
"'-fromnode=bar -fromowner=roecken'"
```

- d. Set the **vendoropt** database configuration parameter so that the load recovery file can also be used during the roll-forward operation using the following command:

```
dps:/home/regress9> db2 update db cfg for zample using VENDOROPT  
"'-fromnode=bar -fromowner=roecken'"
```

6. You can finish the cross-node recovery by performing the roll-forward operation using the following command:

```
dps:/home/regress9> db2 rollforward db zample to end of logs and stop
```

The following information is returned:

```
Rollforward Status  
  
Input database alias      = zample  
Number of members have returned status = 1  
  
Member number  Rollforward   Next log to    Log files processed      Last committed transaction  
           status        be read  
-----  -----  
       0     not pending          S0000000.LOG-S0000000.LOG  2009-05-06-15.28.11.000000 UTC
```

```
DB20000I The ROLLFORWARD command completed successfully.
```

The database zample on computer dps under user regress9 has been recovered to the same point as the database on computerbar under user roecken

Example 3: TSM server is configured to use client proxy nodes

This cross-node recovery example shows how to set up two computers as proxy nodes so that you can recover data from one computer to another when log archives and backups are stored on a TSM server and where passwords are managed using the PASSWORDACCESS=GENERATE option.

Note: After updating the database configuration, you might have to take an offline backup of the database.

In this example, the computers bar and dps are registered under the proxy name of clusternode. The computers are already setup as proxy nodes.

1. Register the computers bar and dps on the TSM server as proxy nodes using the following commands:

```
REGISTER NODE clusternode mypassword  
GRANT PROXYNODE TARGET=clusternode AGENT=bar,dps
```

2. To enable the database for log archiving to the TSM server, update the database configuration parameter **logarchmeth1** for the zample database using the following command:

```
bar:/home/roecken> db2 update db cfg for zample using  
LOGARCHMETH1 tsm logarchopt1 "-asnodenname=clusternode"
```

The following information is returned:

```
DB20000I The UPDATE DATABASE CONFIGURATION command completed successfully.
```

3. Disconnect all users and applications from the database using the following command:

```
db2 force applications all
```

4. Verify that there are no applications connected to the database using the following command:

```
db2 list applications
```

You should receive a message that says that no data was returned.

Note: In a partitioned database environment, you must perform this step on all database partitions.

5. Create a backup of the database on the TSM server using the following command:

```
db2 backup db zample use tsm options "-asnodenname=clusternode"
```

Information similar to the following is returned:

```
Backup successful. The timestamp for this backup image is : 20090216151025
```

Instead of specifying the **-asnodenname** option on the **BACKUP DATABASE** command, you can update the **vendoropt** database configuration parameter instead.

Note: In a partitioned database environment, you must perform this step on all database partitions. The order in which you perform this step on the database partitions differs depending on whether you are performing an online backup or an offline backup. For more information, see Chapter 12, “Backing up databases,” on page 301.

6. Connect to the zample database using the following command:

```
db2 connect to zample
```

7. Generate new transaction logs for the database by creating a table and loading data into the TSM server using the following command:

```
bar:/home/roecken> db2 load from mr of del modified by noheader  
replace into employee copy yes use tsmwhere
```

where in this example, the table is called employee, and the data is being loaded from a delimited ASCII file called mr. The **COPY YES** option is specified to make a copy of the data that is loaded, and the **USE TSM** option specifies that the copy of the data is stored on the TSM server.

Note: You can specify the **COPY YES** option only if the database is enabled for roll-forward recovery; that is, the **logarchmeth1** database configuration parameter must be set to USEREXIT, LOGRETAIN, DISK, or TSM.

To indicate its progress, the load utility returns a series of messages:

```
SQL3109N The utility is beginning to load data from file "/home/roecken/mr".
```

```
SQL3500W The utility is beginning the "LOAD" phase at time "02/16/2009  
15:12:13.392633".
```

```
SQL3519W Begin Load Consistency Point. Input record count = "0".
```

```
SQL3520W Load Consistency Point was successful.
```

```
SQL3110N The utility has completed processing. "1" rows were read from the  
input file.
```

```
SQL3519W Begin Load Consistency Point. Input record count = "1".
```

```
SQL3520W Load Consistency Point was successful.
```

```
SQL3515W The utility has finished the "LOAD" phase at time "02/16/2009  
15:12:13.445718".
```

```
Number of rows read      = 1  
Number of rows skipped   = 0  
Number of rows loaded    = 1  
Number of rows rejected  = 0  
Number of rows deleted   = 0  
Number of rows committed = 1
```

8. After the data has been loaded into the table, confirm that there is one backup image, one load copy image, and one log file on the TSM server by running the following query on the zample database:

```
bar:/home/roecken/sql1lib/adsm> db2adut1 query db zample  
options "-asnodenname=clusternode"
```

The following information is returned:

```
Retrieving FULL DATABASE BACKUP information.  
1 Time: 20090216151025 Oldest log: S0000000.LOG Log stream: 0  
Sessions: 1
```

```
Retrieving INCREMENTAL DATABASE BACKUP information.  
No INCREMENTAL DATABASE BACKUP images found for ZAMPLE
```

```
Retrieving DELTA DATABASE BACKUP information.  
No DELTA DATABASE BACKUP images found for ZAMPLE
```

```
Retrieving TABLESPACE BACKUP information.  
No TABLESPACE BACKUP images found for ZAMPLE
```

```
Retrieving INCREMENTAL TABLESPACE BACKUP information.  
No INCREMENTAL TABLESPACE BACKUP images found for ZAMPLE
```

```
Retrieving DELTA TABLESPACE BACKUP information.  
No DELTA TABLESPACE BACKUP images found for ZAMPLE
```

```
Retrieving LOAD COPY information.  
1 Time: 20090216151213
```

```
Retrieving LOG ARCHIVE information.  
Log file: S0000000.LOG, Chain Num: 0, Log stream: 0,  
Taken at: 2009-02-16-15.10.38
```

9. In this example, computer 2, dps, is not yet set up for cross-node recovery of the zample database. Verify that there is no data associated with this user and computer using the following command:

```
dps:/home/regress9/sql1lib/adsm> db2adut1 query db zample
```

The following information is returned:

```
--- Database directory is empty ---  
Warning: There are no file spaces created by DB2 on the ADSM server  
Warning: No DB2 backup images found in ADSM for any alias.
```

10. Query the TSM server for a list of objects for the zample database associated with the proxy node clusternode using the following command:

```
dps:/home/regress9/sql1lib/adsm> db2adut1 query db zample  
options="-asnodenname=clusternode"
```

The following information is returned:

```
--- Database directory is empty ---  
  
Query for database ZAMPLE  
  
Retrieving FULL DATABASE BACKUP information.  
1 Time: 20090216151025 Oldest log: S0000000.LOG Log stream: 0  
Sessions: 1  
  
Retrieving INCREMENTAL DATABASE BACKUP information.  
No INCREMENTAL DATABASE BACKUP images found for ZAMPLE  
  
Retrieving DELTA DATABASE BACKUP information.  
No DELTA DATABASE BACKUP images found for ZAMPLE  
  
Retrieving TABLESPACE BACKUP information.  
No TABLESPACE BACKUP images found for ZAMPLE  
  
Retrieving INCREMENTAL TABLESPACE BACKUP information.  
No INCREMENTAL TABLESPACE BACKUP images found for ZAMPLE  
  
Retrieving DELTA TABLESPACE BACKUP information.  
No DELTA TABLESPACE BACKUP images found for ZAMPLE  
  
Retrieving LOAD COPY information.  
1 Time: 20090216151213  
  
Retrieving LOG ARCHIVE information.  
Log file: S0000000.LOG, Chain Num: 0, Log stream: 0,  
Taken at: 2009-02-16-15.10.38
```

This information matches the TSM information that was generated previously and confirms that you can restore this image onto the dps computer.

11. Restore the zample database from the TSM server to the dps computer using the following command:

```
dps:/home/regress9> db2 restore db zample use tsm options  
"-asnodenname=clusternode" without prompting
```

The following information is returned:

```
DB20000I The RESTORE DATABASE command completed successfully.
```

Note: If the zample database already existed on dps, the **OPTIONS** parameter would be omitted, and the database configuration parameter **vendoropt** would be used. This configuration parameter overrides the **OPTIONS** parameter for a backup or restore operation.

12. Perform a roll-forward operation to apply the transactions recorded in the zample database log file when a new table was created and new data loaded. In this example, the following attempt for the roll-forward operation will fail because the roll-forward utility cannot find the log files because the user and computer information is not specified:

```
dps:/home/regress9> db2 rollforward db zample to end of logs and stop
```

The command returns the following error:

```
SQL4970N Roll-forward recovery on database "ZAMPLE" cannot reach the  
specified stop point (end-of-log or point-in-time) because of missing log  
file(s) on node(s) "0".
```

Force the roll-forward utility to look for log files on another computer using the proper **logarchopt** value. In this example, use the following command to set the **logarchopt1** database configuration parameter and search for log files associated with user roecken and computer bar:

```
dps:/home/regress9> db2 update db cfg for zample using logarchopt1
          "'-asnodename=clusternode'"
```

13. Enable the roll-forward utility to use the backup and load copy images by setting the **vendoropt** database configuration parameter using the following command:

```
dps:/home/regress9> db2 update db cfg for zample using VENDOROPT
          "'-asnodename=clusternode'"
```

14. You can finish the cross-node data recovery by applying the transactions recorded in the zample database log file using the following command:

```
dps:/home/regress9> db2 rollforward db zample to end of logs and stop
```

The following information is returned:

```
Rollforward Status

Input database alias           = zample
Number of members have returned status = 1

Member number  Rollforward      Next log to    Log files processed      Last committed transaction
          status        be read
-----  -----  -----
          0            not pending           S0000000.LOG-S0000000.LOG  2009-05-06-15.28.11.000000 UTC

DB20000I  The ROLLFORWARD command completed successfully.
```

The database zample on computer dps under user regress9 has been recovered to the same point as the database on computer bar under user roecken.

Example 4: TSM server is configured to use client proxy nodes in a DB2 pureScale environment

This example shows how to set up two members as proxy nodes so that you can recover data from one member to the other when log archives and backups are stored on a TSM server and where passwords are managed using the **PASSWORDACCESS=GENERATE** option.

Note: After updating the database configuration, you might have to take an offline backup of the database.

In this example, the members **member1** and **member2** are registered under the proxy name of **clusternode**. In DB2 pureScale environments, you can perform backup or data recovery operations from any member. In this example, data will be recovered from **member2**.

1. Register the members **member1** and **member2** on the TSM server as proxy nodes using the following commands:

```
REGISTER NODE clusternode mypassword
GRANT PROXYNODE TARGET=clusternode AGENT=member1,member2
```

2. To enable the database for log archiving to the TSM server, update the database configuration parameter **logarchmeth1** for the zample database using the following command:

```
member1:/home/roecken> db2 update db cfg for zample using
          LOGARCHMETH1 tsm logarchopt1 "'-asnodename=clusternode'"
```

Note: In DB2 pureScale environments, you can set the global **logarchmeth1** database configuration parameters once from any member.

The following information is returned:

DB20000I The UPDATE DATABASE CONFIGURATION command completed successfully.

3. Disconnect all users and applications from the database using the following command:

```
db2 force applications all
```

4. Verify that there are no applications connected to the database using the following command:

```
db2 list applications global
```

You should receive a message that says that no data was returned.

5. Create a backup of the database on the TSM server using the following command:

```
db2 backup db zample use tsm options '-asnodename=clusternode'
```

Information similar to the following is returned:

```
Backup successful. The timestamp for this backup image is : 20090216151025
```

Instead of specifying the **-asnodename** option on the **BACKUP DATABASE** command, you can update the **vendoropt** database configuration parameter instead.

Note: In DB2 pureScale environments, you can run this command from any member to back up all data for the database.

6. Connect to the zample database using the following command:

```
db2 connect to zample
```

7. Generate new transaction logs for the database by creating a table and loading data into the TSM server using the following command:

```
member1:/home/roecken> db2 load from mr of del modified by noheader replace  
into employee copy yes use tsmwhere
```

where in this example, the table is called employee, and the data is being loaded from a delimited ASCII file called mr. The **COPY YES** option is specified to make a copy of the data that is loaded, and the **USE TSM** option specifies that the copy of the data is stored on the TSM server.

Note: You can specify the **COPY YES** option only if the database is enabled for roll-forward recovery; that is, the **logarchmeth1** database configuration parameter must be set to **USEREXIT**, **LOGRETAIN**, **DISK**, or **TSM**.

To indicate its progress, the load utility returns a series of messages:

```
SQL3109N The utility is beginning to load data from file "/home/roecken/mr".
```

```
SQL3500W The utility is beginning the "LOAD" phase at time "02/16/2009  
15:12:13.392633".
```

```
SQL3519W Begin Load Consistency Point. Input record count = "0".
```

```
SQL3520W Load Consistency Point was successful.
```

```
SQL3110N The utility has completed processing. "1" rows were read from the  
input file.
```

```
SQL3519W Begin Load Consistency Point. Input record count = "1".
```

```
SQL3520W Load Consistency Point was successful.
```

```
SQL3515W The utility has finished the "LOAD" phase at time "02/16/2009  
15:12:13.445718".
```

Number of rows read = 1

```

Number of rows skipped      = 0
Number of rows loaded      = 1
Number of rows rejected    = 0
Number of rows deleted     = 0
Number of rows committed   = 1

```

- After the data has been loaded into the table, confirm that there is one backup image, one load copy image, and one log file on the TSM server by running the following query on the zample database:

```
member1:/home/roecken/sqllib/adsm> db2adutl query db zample
options "-asnodename=clusternode"
```

The following information is returned:

```

Retrieving FULL DATABASE BACKUP information.
  1 Time: 20090216151025 Oldest log: S0000000.LOG Log stream: 0
  Sessions: 1

```

```

Retrieving INCREMENTAL DATABASE BACKUP information.
  No INCREMENTAL DATABASE BACKUP images found for ZAMPLE

```

```

Retrieving DELTA DATABASE BACKUP information.
  No DELTA DATABASE BACKUP images found for ZAMPLE

```

```

Retrieving TABLESPACE BACKUP information.
  No TABLESPACE BACKUP images found for ZAMPLE

```

```

Retrieving INCREMENTAL TABLESPACE BACKUP information.
  No INCREMENTAL TABLESPACE BACKUP images found for ZAMPLE

```

```

Retrieving DELTA TABLESPACE BACKUP information.
  No DELTA TABLESPACE BACKUP images found for ZAMPLE

```

```

Retrieving LOAD COPY information.
  1 Time: 20090216151213

```

Retrieving LOG ARCHIVE information.

```
Log file: S0000000.LOG, Chain Num: 1, Log stream: 1, Taken at: 2009-02-16-13.01.10
```

```
Log file: S0000000.LOG, Chain Num: 1, Log stream: 0, Taken at: 2009-02-16-13.01.11
```

```
Log file: S0000000.LOG, Chain Num: 1, Log stream: 2, Taken at: 2009-02-16-13.01.19
```

```
Log file: S0000001.LOG, Chain Num: 1, Log stream: 0, Taken at: 2009-02-16-13.02.49
```

```
Log file: S0000001.LOG, Chain Num: 1, Log stream: 1, Taken at: 2009-02-16-13.02.49
```

```
Log file: S0000001.LOG, Chain Num: 1, Log stream: 2, Taken at: 2009-02-16-13.02.49
```

```
Log file: S0000002.LOG, Chain Num: 1, Log stream: 1, Taken at: 2009-02-16-13.03.15
```

```
Log file: S0000002.LOG, Chain Num: 1, Log stream: 2, Taken at: 2009-02-16-13.03.15
```

```
Log file: S0000002.LOG, Chain Num: 1, Log stream: 0, Taken at: 2009-02-16-13.03.16
```

- Query the TSM server for a list of objects for the zample database associated with the proxy node clusternode using the following command:

```
member2:/home/regress9/sqllib/adsm> db2adutl query db zample
options="-asnodename=clusternode"
```

The following information is returned:

```
--- Database directory is empty ---
```

```
Query for database ZAMPLE
```

```

Retrieving FULL DATABASE BACKUP information.
  1 Time: 20090216151025 Oldest log: S0000000.LOG Log stream: 0
  Sessions: 1

```

```

Retrieving INCREMENTAL DATABASE BACKUP information.
  No INCREMENTAL DATABASE BACKUP images found for ZAMPLE

```

```

Retrieving DELTA DATABASE BACKUP information.

```

```

        No DELTA DATABASE BACKUP images found for ZAMPLE

Retrieving TABLESPACE BACKUP information.
        No TABLESPACE BACKUP images found for ZAMPLE

Retrieving INCREMENTAL TABLESPACE BACKUP information.
        No INCREMENTAL TABLESPACE BACKUP images found for ZAMPLE

Retrieving DELTA TABLESPACE BACKUP information.
        No DELTA TABLESPACE BACKUP images found for ZAMPLE

Retrieving LOAD COPY information.
        1 Time: 20090216151213

Retrieving LOG ARCHIVE information.

        Log file: S0000000.LOG, Chain Num: 1, Log stream: 1, Taken at: 2009-02-16-13.01.10
        Log file: S0000000.LOG, Chain Num: 1, Log stream: 0, Taken at: 2009-02-16-13.01.11
        Log file: S0000000.LOG, Chain Num: 1, Log stream: 2, Taken at: 2009-02-16-13.01.19
        Log file: S0000001.LOG, Chain Num: 1, Log stream: 0, Taken at: 2009-02-16-13.02.49
        Log file: S0000001.LOG, Chain Num: 1, Log stream: 1, Taken at: 2009-02-16-13.02.49
        Log file: S0000001.LOG, Chain Num: 1, Log stream: 2, Taken at: 2009-02-16-13.02.49
        Log file: S0000002.LOG, Chain Num: 1, Log stream: 1, Taken at: 2009-02-16-13.03.15
        Log file: S0000002.LOG, Chain Num: 1, Log stream: 2, Taken at: 2009-02-16-13.03.15
        Log file: S0000002.LOG, Chain Num: 1, Log stream: 0, Taken at: 2009-02-16-13.03.16

```

This information matches the TSM information that was generated previously and confirms that you can restore this image onto the member2 member.

10. Restore the zample database on the TSM server from the member2 member using the following command:

```
member2:/home/regress9> db2 restore db zample use tsm options
'-asnodename=clusternode' without prompting
```

The following information is returned:

```
DB20000I The RESTORE DATABASE command completed successfully.
```

Note: If the zample database already existed on member2, the **OPTIONS** parameter would be omitted, and the database configuration parameter **vendoropt** would be used. This configuration parameter overrides the **OPTIONS** parameter for a backup or restore operation.

11. Enable the roll-forward utility to use the backup and load copy images by setting the **vendoropt** database configuration parameter using the following command:

```
member2:/home/regress9> db2 update db cfg for zample using VENDOROPT
"-asnodename=clusternode"
```

Note: In DB2 pureScale environments, you can set the global **vendoropt** database configuration parameters once from any member.

12. You can finish the cross-member data recovery by applying the transactions recorded in the zample database log file using the following command:

```
member2:/home/regress9> db2 rollforward db zample to end of logs and stop
```

The following information is returned:

```

Rollforward Status

Input database alias      = zample
Number of members have returned status = 3

```

Member number	Rollforward status	Next log to be read	Log files processed	Last committed transaction
0	not pending		S0000001.LOG-S0000012.LOG	2009-05-06-15.28.11.000000 UTC
1	not pending		S0000001.LOG-S0000012.LOG	2009-05-06-15.28.11.000000 UTC
2	not pending		S0000001.LOG-S0000012.LOG	2009-05-06-15.28.11.000000 UTC

DB20000I The ROLLFORWARD command completed successfully.

The database zample on member member2 under user regress9 has been recovered to the same point as the database on member member1 under user roecken.

Recovering a dropped table

You might occasionally drop a table that contains data you still need. If so, you should consider making your critical tables recoverable following a drop table operation.

You could recover the table data by invoking a database restore operation, followed by a database rollforward operation to a point in time before the table was dropped. The restore and rollforward operations can be time-consuming if the database is large, and your data is unavailable during the recovery.

The dropped table recovery feature lets you recover your dropped table data by using table space-level restore and rollforward operations.

This table space-level recovery is faster than database-level recovery, and your database remains available to users.

Before you begin

For a dropped table to be recoverable, the table space in which the table resides must have the DROPPED TABLE RECOVERY option turned on. This option can be enabled during table space creation, or by invoking the ALTER TABLESPACE statement. The DROPPED TABLE RECOVERY option is table space-specific and limited to regular table spaces. To determine if a table space is enabled for dropped table recovery, you can query the DROP_RECOVERY column in the SYSCAT.TABLESPACES catalog table.

The dropped table recovery option is on by default when you create a table space. If you do not want to enable a table space for dropped table recovery, you can either explicitly set the DROPPED TABLE RECOVERY option to OFF when you issue the CREATE TABLESPACE statement, or you can use the ALTER TABLESPACE statement to disable dropped table recovery for an existing table space. If there are many drop table operations to recover, or if the history file is large, the dropped table recovery feature might have a performance impact on forward recovery.

When a DROP TABLE statement is run against a table whose table space is enabled for dropped table recovery, an additional entry (identifying the dropped table) is made in the log files. An entry is also made in the recovery history file, containing information that can be used to re-create the table.

For partitioned tables, dropped table recovery is always on even if the dropped table recovery is turned off for non-partitioned tables in one or more table spaces. Only one dropped table log record is written for a partitioned table. This log record is sufficient to recover all the data partitions of the table.

About this task

If the table was in reorg pending state when it was dropped, the CREATE TABLE DDL in the history file does not match exactly that of the import file. The import file is in the format of the table before the first **REORG**-recommended ALTER was performed, but the CREATE TABLE statement in the history file matches the state of the table including the results of any ALTER TABLE statements.

File type modifiers to use with **LOAD** or **IMPORT**

To recover the table by loading or importing, specify the following file type modifiers:

- The file type modifier **usegraphiccodepage** should be used in the **IMPORT** or **LOAD** command if the data being recovered is of the GRAPHIC or VARGRAPHIC data type. The reason is that it might include more than one code page.
- The file type modifier **delprioritychar** should be used in the **IMPORT** or **LOAD** commands. It allows **LOAD** and **IMPORT** to parse rows which contains newline characters within character or graphic column data.

Restrictions

Only one dropped table can be recovered at a time.

There are some restrictions on the type of data that is recoverable from a dropped table. It is not possible to recover:

- The DROPPED TABLE RECOVERY option cannot be used for temporary table.
- The metadata associated with row types. (The data is recovered, but not the metadata.) The data in the hierarchy table of the typed table is recovered. This data might contain more information than appeared in the typed table that was dropped.
- XML data. If you attempt to recover a dropped table that contains XML data, the corresponding column data is empty.

Procedure

You can recover a dropped table by doing the following:

1. Identify the dropped table by invoking the **LIST HISTORY DROPPED TABLE** command. The dropped table ID is listed in the Backup ID column.
2. Restore a database- or table space-level backup image taken before the table was dropped.
3. Create an export directory to which files containing the table data are to be written. This directory must either be accessible to all database partitions, or exist on each database partition. Subdirectories under this export directory are created automatically by each database partition. These subdirectories are named **NODEnnnn**, where *nnnn* represents the database partition or node number. Data files containing the dropped table data as it existed on each database partition are exported to a lower subdirectory called **data**. For example:
`\export_directory\NODE0000\data`.
4. Roll forward to a point in time after the table was dropped, by using the **RECOVER DROPPED TABLE** parameter on the **ROLLFORWARD DATABASE** command. Alternatively, roll forward to the end of the logs, so that updates to other tables in the table space or database are not lost.
5. Re-create the table by using the CREATE TABLE statement from the recovery history file.

6. Import the table data that was exported during the rollforward operation into the table. If the table was in reorg pending state when the drop took place, the contents of the CREATE TABLE DDL might need to be changed to match the contents of the data file.

Chapter 14. Restoring databases

Restoring a database backup recovers a database or table space after a problem such as media or storage failure, or application failure. If you back up your database or individual table spaces, you can recreate them if they become damaged or corrupted in some way.

Before you begin

When restoring to an *existing* database, you should not be connected to the database that is to be restored: the restore utility automatically establishes a connection to the specified database, and this connection is terminated at the completion of the restore operation. When restoring to a *new* database, an instance attachment is required to create the database. When restoring to a *new remote* database, you must first attach to the instance where you want the new database to reside. Then, create the new database, specifying the code page and the territory of the server. Restore overwrites the code page of the destination database with the code page of the backup image.

About this task

The database can be local or remote.

The following restrictions apply to the restore utility:

- You can only use the restore utility if the database has been previously backed up using the DB2 backup utility.
- If users other than the instance owner (on UNIX), or members of the DB2ADMNS or Administrators group (on Windows) attempt to restore a backup image, they will get an error (SQL2061N). If other users need access to the backup image, the file permissions need to be changed after the backup is generated.
- A database restore operation cannot be started while the rollforward process is running.
- If you do not specify the **TRANSPORT** option, then you can restore a table space into an existing database only if the table space currently exists, and if it is the same table space. In this situation, “same” means that the table space was not dropped and then re-created between the backup and the restore operation. The database on disk and in the backup image must be the same.
- You cannot issue a table space-level restore of a table space-level backup to a new database.
- You cannot perform an online table space-level restore operation involving the system catalog tables.
- You cannot restore a backup taken in a single database partition environment into an existing partitioned database environment. Instead you must restore the backup to a single database partition environment and then add database partitions as required.
- When restoring a backup image with one code page into a system with a different codepage, the system code page will be overwritten by the code page of the backup image.
- You cannot use the **RESTORE DATABASE** command to convert nonautomatic storage enabled table spaces to automatic storage enabled table space.

- The following restrictions apply when the **TRANSPORT** option is specified:
 - If the backup image can be restored by a restore operation, and is supported for upgrades, then it can be transported.
 - If an online backup is used, then both source and target data servers must be running the same DB2 version.
 - The **RESTORE DATABASE** command must be issued against the target database. If the remote client is of the same platform as the server, then schema transport can be executed locally on the server or through remote instance attachment. If a target database is a remote database cataloged in the instance where transport runs locally, then schema transport against that remote target database is not supported.
 - You can only transport tables spaces and schemas into an existing database. The transport operation will not create a new database. To restore a database into a new database, you can use the **RESTORE DATABASE** command without specifying the **TRANSPORT** option.
 - If the schemas in the source database are protected by any DB2 security settings or authorizations, then the transported schemas in the target database will retain these same settings or authorizations.
 - Transport is not supported for partitioned database environments.
 - If any of the tables within the schema contains an XML column, the transport fails.
 - The **TRANSPORT** option is incompatible with the **REBUILD** option.
 - The **TRANSPORT** option is not supported for restore from a snapshot backup image.
 - The staging database is created for transport. It cannot be used for other operations.
 - The database configuration parameters on the staging table and the target table need to be the same, or the transport operation fails with an incompatibility error.
 - The **auto_reval** configuration parameter must be set to **deferred_force** on the target database to transport objects listed as invalid. Otherwise, the transport fails.
 - If an online backup image is used, and the active logs are not included, then the transport operation fails.
 - If an online backup is used, then the backup image must have been created with the **INCLUDE LOGS** option.
 - If the backup image is from a previous release, it must be a full offline database level backup image.
 - If an error occurs on either the staging or target database, the entire restore operation must be reissued. All failures that occur are logged in the **db2diag** log file on the target server and should be reviewed before reissuing the **RESTORE** command.
 - If the transport client fails, then the staging database might not be properly cleaned up. In this case, you need to drop the staging database. Before re-issuing the **RESTORE** command, drop all staging databases to prevent containers of staging database from blocking subsequent transport.
 - Concurrent transport running against the same target database is not supported.
 - Generating a redirected restore script is not supported with table space transport.

- You can restore a table space if the storage group has been updated. The target storage group during the table space restore is the storage group the table space is currently associated with when **RESTORE** is executed.
- You cannot perform a point-in-time recovery to an earlier storage group association.

Procedure

To invoke the restore utility:

- Issue the **RESTORE DATABASE** command.
- Call the db2Restore application programming interface (API).
- Open the task assistant in IBM Data Studio for the **RESTORE DATABASE** command.

Example

Following is an example of the **RESTORE DATABASE** command issued through the CLP:

```
db2 restore db sample from D:\DB2Backups taken at 20010320122644
```

Restoring from a snapshot backup image

Restoring from a snapshot backup uses the fast copying technology of a storage device to perform the data copying portion of the restore.

Before you begin

To perform snapshot backup and restore operations, you need a DB2 ACS API driver for your storage device. For a list of supported storage hardware for the integrated driver, refer to this tech note.

You must perform a snapshot backup before you can restore from a snapshot backup. See: "Performing a snapshot backup" on page 303.

Procedure

You can restore from a snapshot backup using the **RESTORE DATABASE** command with the **USE SNAPSHOT** parameter, or the db2Restore API with the **SQLU_SNAPSHOT_MEDIA** media type:

- **RESTORE DATABASE** command:

```
db2 restore db sample use snapshot
```
- db2Restore API:

```
int sampleRestoreFunction( char dbAlias[],  
                           char restoredDbAlias[],  
                           char user[],  
                           char pswd[],  
                           char workingPath[] )  
{  
    db2MediaListStruct mediaListStruct = { 0 };  
  
    rmediaListStruct.locations = &workingPath;  
    rmediaListStruct.numLocations = 1;  
    rmediaListStruct.locationType = SQLU_SNAPSHOT_MEDIA;
```

```

db2RestoreStruct restoreStruct = { 0 };

restoreStruct.piSourceDBAlias = dbAlias;
restoreStruct.piTargetDBAlias = restoredDbAlias;
restoreStruct.piMediaList = &mediaListStruct;
restoreStruct.piUsername = user;
restoreStruct.piPassword = pswd;
restoreStruct.iCallerAction = DB2RESTORE_STORDEF_NOINTERRUPT;

struct sqlca sqlca = { 0 };

db2Restore(db2Version900, &restoreStruct, &sqlca);

return 0;
}

```

Using incremental restore in a test and production environment

Once a production database is enabled for incremental backup and recovery, you can use an incremental or delta backup image to create or refresh a test database.

You can do this by using either manual or automatic incremental restore.

To restore the backup image from the production database to the test database, use the `INTO target-database-alias` option on the **RESTORE DATABASE** command. For example, in a production database with the following backup images:

```

backup db prod
Backup successful. The timestamp for this backup image is : ts1

backup db prod incremental
Backup successful. The timestamp for this backup image is : ts2

```

an example of a manual incremental restore would be:

```

restore db prod incremental taken at ts2 into test without
prompting
DB20000I The RESTORE DATABASE command completed successfully.

restore db prod incremental taken at ts1 into test without
prompting
DB20000I The RESTORE DATABASE command completed successfully.

restore db prod incremental taken at ts2 into test without
prompting
DB20000I The RESTORE DATABASE command completed successfully.

```

If the database TEST already exists, the restore operation overwrites any data that is already there. If the database TEST does not exist, the restore utility creates it and then populates it with the data from the backup images.

Since automatic incremental restore operations are dependent on the database history, the restore steps change slightly based on whether the test database exists. To perform an automatic incremental restore to the database TEST, its history must contain the backup image history for database PROD. The database history for the backup image replaces any database history that already exists for database TEST if either of the following are true:

- The database TEST does not exist when the **RESTORE DATABASE** command is issued.

- The database TEST exists when the **RESTORE DATABASE** command is issued, and the database TEST history contains no records.

The following example shows an automatic incremental restore to database TEST which does not exist:

```
restore db prod incremental automatic taken at ts2 into test without
prompting
DB20000I The RESTORE DATABASE command completed successfully.
```

The restore utility creates the TEST database and populates it.

If the database TEST does exist and the database history is not empty, you must drop the database before the automatic incremental restore operation as follows:

```
drop db test
DB20000I The DROP DATABASE command completed successfully.
```

```
restore db prod incremental automatic taken at ts2 into test without
prompting
DB20000I The RESTORE DATABASE command completed successfully.
```

If you do not want to drop the database, you can issue the **PRUNE HISTORY** command with a timestamp far into the future and the **WITH FORCE OPTION** parameter before issuing the **RESTORE DATABASE** command:

```
connect to test
Database Connection Information

Database server      = server_id
SQL authorization ID = id
Local database alias = TEST

prune history 9999 with force option
DB20000I The PRUNE command completed successfully.

connect reset
DB20000I The SQL command completed successfully.
restore db prod incremental automatic taken at ts2 into test without
prompting
SQL2540W Restore is successful, however a warning "2539" was
encountered during Database Restore while processing in No
Interrupt mode.
```

In this case, the **RESTORE DATABASE** command acts in the same manner as when the database TEST did not exist.

If the database TEST does exist and the database history is empty, you do not have to drop the database TEST before the automatic incremental restore operation:

```
restore db prod incremental automatic taken at ts2 into test without
prompting
SQL2540W Restore is successful, however a warning "2539" was
encountered during Database Restore while processing in No
Interrupt mode.
```

You can continue taking incremental or delta backups of the test database without first taking a full database backup. However, if you ever need to restore one of the incremental or delta images you have to perform a manual incremental restore. This requirement is because automatic incremental restore operations require that each of the backup images restored during an automatic incremental restore are created from the same database alias.

If you make a full database backup of the test database after you complete the restore operation using the production backup image, you can take incremental or delta backups and can restore them using either manual or automatic mode.

Performing a redirected restore operation

A database restore operation uses a database backup image to recreate a database.

Use a redirected restore operation in any of the following situations:

- If you want to restore a backup image to a target machine that is different from the source machine
- If you want to restore your table space containers into a different physical location
- If your restore operation failed because one or more containers are inaccessible
- If you want to redefine the paths of a defined storage group

Restrictions:

You cannot use a redirected restore to move data from one operating system to another.

You cannot create or drop a storage group during the restore process.

You cannot modify storage group paths during a table space restore process even if you are restoring all table spaces that are associated with the storage group.

The process for performing a redirected restore by using an incremental backup image is similar to the process of performing a redirected restore by using a non-incremental backup image. Use one of the following approaches:

- Issue the **RESTORE DATABASE** command with the **REDIRECT** parameter, and specify the backup image to use for the incremental restore of the database.
- Generate a redirected restore script from a backup image, and then modify the script as required.

Using the **RESTORE DATABASE** command approach is a two-step database restore process with an intervening step for defining a table space container or storage group path. To perform a redirected restore:

1. Issue the **RESTORE DATABASE** command with the **REDIRECT** parameter.
2. Take one of the following steps:
 - Define table space containers by issuing the **SET TABLESPACE CONTAINERS** command.
 - Define storage group paths for the database to be restored by issuing the **SET STOGROUP PATHS** command.
3. Issue the **RESTORE DATABASE** command again, this time specifying the **CONTINUE** parameter.

After you issue the **RESTORE CONTINUE** command, the new path takes effect as the table space container path for all associated table spaces. If you issue a **LIST TABLESPACE CONTAINERS** command or a **GET SNAPSHOT FOR TABLESPACES** command after the **SET STOGROUP PATHS** command and before the **RESTORE CONTINUE** command, the output for the table space container paths does not reflect the new paths that you specified by using the **SET STOGROUP PATHS** command.

During a redirected restore operation, directory and file containers are automatically created if they do not exist. The database manager does not automatically create device containers.

DB2 database products provide SQL statements for adding, changing, or removing table space containers non-automatic-storage DMS table spaces, and storage group paths of automatic storage table spaces. A redirected restore is the only way to modify a non-automatic-storage SMS table space container configuration.

You can redefine table space containers or modify storage group paths by issuing the **RESTORE DATABASE** command with the **REDIRECT** parameter.

Table space container redirection provides considerable flexibility for managing table space containers. You can alter the storage group configuration of a database before restoring any data pages from the backup image, similar to the way that you can redirect table space container paths. If you renamed a storage group since you produced the backup image, the storage group name that is specified by the **SET STOGROUP PATHS** command refers to the storage group name from the backup image, not the more recent name.

Performing a redirected restore operation in a partitioned database environment

In a partitioned database environment, during a redirected database restore, you can redirect the storage group paths to new storage group paths only from the catalog database partition. For all other database partitions you must have their storage group paths synchronized with those of the catalog partition.

Modifying any storage group paths on the catalog partition places all non-catalog partitions into a RESTORE_PENDING state. If you redirect storage group paths, you must restore the catalog partition before any other database partition. After you restore the catalog database partition, you can restore the non-catalog database partitions in parallel, without any storage group path redirection. The non-catalog database partitions automatically acquire the new storage group paths that you specified for the catalog database partition. New storage group paths are also automatically acquired when the storage group paths are implicitly changed during a database restore when you are restoring a different database (one with a different name, instance, or seed).

If you modified the storage group paths since taking the last backup, you can still use that backup image (with different storage group paths) for a restore on any database partition. This restore is not considered a redirected restore. Restoring from that backup image temporarily causes the database partition to use the storage group paths that you defined at the time that you created the backup. Perform a rollforward recovery to reapply the storage group path modifications and resynchronize all of the database partitions.

Examples

Example 1

You can perform a table space container redirected restore on database SAMPLE by using the **SET TABLESPACE CONTAINERS** command to define table space containers:

```
db2 restore db sample redirect without prompting
SQL1277W A redirected restore operation is being performed.
During a table space restore, only table spaces being restored can
```

have their paths reconfigured. During a database restore, storage group storage paths and DMS table space containers can be reconfigured.

DB20000I The RESTORE DATABASE command completed successfully.

```
db2 set tablespace containers for 2 using (path 'userspace1.0', path  
'userspace1.1')  
DB20000I The SET TABLESPACE CONTAINERS command completed successfully.
```

```
db2 restore db sample continue  
DB20000I The RESTORE DATABASE command completed successfully.
```

Example 2

You can redefine the paths of the defined storage group by using the **SET STOGROUP PATHS** command:

```
RESTORE DB SAMPLE REDIRECT
```

```
SET STOGROUP PATHS FOR sg_hot ON '/ssd/fs1', '/ssd/fs2'  
SET STOGROUP PATHS FOR sg_cold ON '/hdd/path1', '/hdd/path2'
```

```
RESTORE DB SAMPLE CONTINUE
```

Example 3

Following is a typical non-incremental redirected restore scenario for a database whose alias is MYDB:

1. Issue a RESTORE DATABASE command with the REDIRECT option.

```
db2 restore db mydb replace existing redirect
```

2. Issue a SET TABLESPACE CONTAINERS command for each table space whose containers you want to redefine. For example, in a Windows environment:

```
db2 set tablespace containers for 5 using  
(file 'f:\ts3con1'20000, file 'f:\ts3con2'20000)
```

To verify that the containers of the restored database are the ones specified in this step, issue the LIST TABLESPACE CONTAINERS command for every table space whose container locations are being redefined.

3. After successful completion of steps 1 and 2, issue:

```
db2 restore db mydb continue
```

This is the final step of the redirected restore operation.

4. If step 3 fails, or if the restore operation has been aborted, the redirected restore can be restarted, beginning at step 1.

Note:

1. After successful completion of step 1, and before completing step 3, the restore operation can be aborted by issuing:

```
db2 restore db mydb abort
```

2. If step 3 fails, or if the restore operation has been aborted, the redirected restore can be restarted, beginning at step 1.

Example 4

Following is a typical manual incremental redirected restore scenario for a database whose alias is MYDB and has the following backup images:

```
backup db mydb  
Backup successful. The timestamp for this backup image is : <ts1>
```

```
backup db mydb incremental  
Backup successful. The timestamp for this backup image is : <ts2>
```

1. Issue a RESTORE DATABASE command with the INCREMENTAL and REDIRECT options.

```
db2 restore db mydb incremental taken at <ts2> replace existing redirect
```

2. Issue a SET TABLESPACE CONTAINERS command for each table space whose containers must be redefined. For example, in a Windows environment:

```
db2 set tablespace containers for 5 using  
(file 'f:\ts3con1'20000, file 'f:\ts3con2'20000)
```

To verify that the containers of the restored database are the ones specified in this step, issue the LIST TABLESPACE CONTAINERS command.

3. After successful completion of steps 1 and 2, issue:

```
db2 restore db mydb continue
```

4. The remaining incremental restore commands can now be issued as follows:

```
db2 restore db mydb incremental taken at <ts1>  
db2 restore db mydb incremental taken at <ts2>
```

This is the final step of the redirected restore operation.

Note:

1. After successful completion of step 1, and before completing step 3, the restore operation can be aborted by issuing:

```
db2 restore db mydb abort
```
2. After successful completion of step 3, and before issuing all the required commands in step 4, the restore operation can be aborted by issuing:

```
db2 restore db mydb incremental abort
```
3. If step 3 fails, or if the restore operation has been aborted, the redirected restore can be restarted, beginning at step 1.
4. If either restore command fails in step 4, the failing command can be reissued to continue the restore process.

Example 5

Following is a typical automatic incremental redirected restore scenario for the same database:

1. Issue a RESTORE DATABASE command with the INCREMENTAL AUTOMATIC and REDIRECT options.

```
db2 restore db mydb incremental automatic taken at <ts2>  
replace existing redirect
```

2. Issue a SET TABLESPACE CONTAINERS command for each table space whose containers must be redefined. For example, in a Windows environment:

```
db2 set tablespace containers for 5 using  
(file 'f:\ts3con1'20000, file 'f:\ts3con2'20000)
```

To verify that the containers of the restored database are the ones specified in this step, issue the LIST TABLESPACE CONTAINERS command.

3. After successful completion of steps 1 and 2, issue:

```
db2 restore db mydb continue
```

This is the final step of the redirected restore operation.

Note:

1. After successful completion of step 1, and before completing step 3, the restore operation can be aborted by issuing:

```
db2 restore db mydb abort
```

2. If step 3 fails, or if the restore operation has been aborted, the redirected restore can be restarted, beginning at step 1 after issuing:

```
db2 restore db mydb incremental abort
```

Redefine table space containers by restoring a database using an automatically generated script

When you restore a database, the restore utility assumes that the physical container layout will be identical to that of the database when it was backed up.

If you need to change the location or size of any of the physical containers, you must issue the **RESTORE DATABASE** command with the **REDIRECT** option. Using this option requires that you specify the locations of physical containers stored in the backup image and provide the complete set of containers for each non-automatic table space that will be altered. You can capture the container information at the time of the backup, but this can be cumbersome.

To make it easier to perform a redirected restore, the restore utility allows you to generate a redirected restore script from an existing backup image by issuing the **RESTORE DATABASE** command with the **REDIRECT** parameter and the **GENERATE SCRIPT** parameter. The restore utility examines the backup image, extracts container information from the backup image, and generates a CLP script that includes all of the detailed container information. You can then modify any of the paths or container sizes in the script, then run the CLP script to recreate the database with the new set of containers. The script you generate can be used to restore a database even if you only have a backup image and you do not know the layout of the containers. The script is created on the client. Using the script as your basis, you can decide where the restored database will require space for log files and containers and you can change the log file and container paths accordingly.

The generated script consists of four sections:

Initialization

The first section sets command options and specifies the database partitions on which the command will run. The following is an example of the first section:

```
UPDATE COMMAND OPTIONS USING S ON Z ON SAMPLE_NODE0000.out V ON;  
SET CLIENT ATTACH_DBPARTITIONNUM 0;  
SET CLIENT CONNECT_DBPARTITIONNUM 0;
```

where

- S ON specifies that execution of the command should stop if a command error occurs

- Z ON SAMPLE_NODE0000.out specifies that output should be directed to a file named *dbalias_NODEdbpartitionnum.out*
- V ON specifies that the current command should be printed to standard output.

When running the script on a partitioned database environment, it is important to specify the database partition on which the script commands will run.

RESTORE DATABASE command with the REDIRECT parameter

The second section starts the **RESTORE DATABASE** command and uses the **REDIRECT** parameter. This section can use all of the **RESTORE DATABASE** command parameters, except any parameters that cannot be used with the **REDIRECT** parameter. The following is an example of the second section:

```
RESTORE DATABASE SAMPLE
-- USER 'username'
-- USING 'password'
FROM '/home/jseifert/backups'
TAKEN AT 20050906194027
-- DBPATH ON 'target-directory'
INTO SAMPLE
-- NEWLOGPATH '/home/jseifert/jseifert/NODE0000/SQL00001/LOGSTREAM0000/'
-- WITH num-buff BUFFERS
-- BUFFER buffer-size
-- REPLACE HISTORY FILE
-- REPLACE EXISTING
REDIRECT
-- PARALLELISM n
-- WITHOUT ROLLING FORWARD
-- WITHOUT PROMPTING
;
```

Table space definitions

This section contains table space definitions for each table space in the backup image or specified on the command line. There is a section for each table space, consisting of a comment block that contains information about the name, type and size of the table space. The information is provided in the same format as a table space snapshot. You can use the information provided to determine the required size for the table space. In cases where you are viewing output of a table space created using automatic storage, you will not see a SET TABLESPACE CONTAINERS clause. The following is an example of the table space definition section:

```
-- ****
-- ** Tablespace name          = SYSCATSPACE
-- ** Tablespace ID            = 0
-- ** Tablespace Type          = System managed space
-- ** Tablespace Content Type = Any data
-- ** Tablespace Page size (bytes) = 4096
-- ** Tablespace Extent size (pages) = 32
-- ** Using automatic storage = No
-- ** Total number of pages   = 5572
-- ****
SET TABLESPACE CONTAINERS FOR 0
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  PATH    'SQLT0000.0'
);
-- ****
-- ** Tablespace name          = TEMPSPACE1
-- ** Tablespace ID            = 1
-- ** Tablespace Type          = System managed space
-- ** Tablespace Content Type = System Temporary data
-- ** Tablespace Page size (bytes) = 4096
```

```

-- ** Tablespace Extent size (pages)      = 32
-- ** Using automatic storage            = No
-- ** Total number of pages              = 0
-- ****
SET TABLESPACE CONTAINERS FOR 1
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
    PATH    'SQLT0001.0'
);
-- ****
-- ** Tablespace name                  = DMS
-- ** Tablespace ID                   = 2
-- ** Tablespace Type                = Database managed space
-- ** Tablespace Content Type       = Any data
-- ** Tablespace Page size (bytes)   = 4096
-- ** Tablespace Extent size (pages) = 32
-- ** Using automatic storage        = No
-- ** Auto-resize enabled           = No
-- ** Total number of pages         = 2000
-- ** Number of usable pages       = 1960
-- ** High water mark (pages)      = 96
-- ****
SET TABLESPACE CONTAINERS FOR 2
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
    FILE    '/tmp/dms1'          1000
    , FILE   '/tmp/dms2'          1000
);

```

RESTORE DATABASE command with the CONTINUE parameter

The final section issues the **RESTORE DATABASE** command with the **CONTINUE** parameter, to complete the redirected restore. The following is an example of the final section:

```
RESTORE DATABASE SAMPLE CONTINUE;
```

Performing a redirected restore using an automatically generated script

When you perform a redirected restore operation, you must specify the locations of physical containers that are stored in the backup image and provide the complete set of containers for each table space that you are altering.

Before you begin

You can perform a redirected restore only if the database was previously backed up using the DB2 backup utility.

About this task

- If the database exists, you must be able to connect to it in order to generate the script. Therefore, if the database requires an upgrade or crash recovery, this must be done before you attempt to generate a redirected restore script.
- If you are working in a partitioned database environment, and the target database does not exist, you cannot run the command to generate the redirected restore script concurrently on all database partitions. Instead, the command to generate the redirected restore script must be run one database partition at a time, starting from the catalog partition.

Alternatively, you can first create a dummy database with the same name as your target database. After the dummy database is created, you can then generate the redirected restore script concurrently on all database partitions.

- Even if you specify the **REPLACE EXISTING** parameter when you issue the **RESTORE DATABASE** command to generate the script, the **REPLACE EXISTING** parameter is commented out in the script.
- For security reasons, your password does not appear in the generated script. You need to enter the password manually.
- The restore script includes the storage group associations for every table space that you restore.

Procedure

To perform a redirected restore using a script:

1. Use the restore utility to generate a redirected restore script. The restore utility can be invoked through the command line processor (CLP) or the db2Restore application programming interface (API). The following is an example of the **RESTORE DATABASE** command with the **REDIRECT** parameter and the **GENERATE SCRIPT** parameter:

```
db2 restore db test from /home/jseifert/backups taken at 20050304090733
    redirect generate script test_node0000.clp
```

This creates a redirected restore script on the client called `test_node0000.clp`.

2. Open the redirected restore script in a text editor to make any modifications that are required. You can modify:
 - Restore options
 - Automatic storage paths
 - Container layout and paths
3. Run the modified redirected restore script. For example:

```
db2 -tvf test_node0000.clp
```

Cloning a production database using different storage group paths

You might have to clone a production database onto a test database that uses a different machine. The test machine and production server are likely to have different storage group paths. The test system might not have paths backed by the newest and fastest storage disks.

About this task

Suppose you have a production database `proddb`, where some data is in storage group `sg_hot`, which has paths on an SSD device. You want to restore the data into the less expensive and lower-performance test database `testdb`. The test system does not have the SSD device, but the other paths are equivalent. Performing a redirected restore can change the paths for `sg_hot` on the test system without changing the other storage groups.

Procedure

To restore data from a production database to a test database:

1. Back up the production database. Issue the following command:

```
BACKUP DATABASE production_db TO /backup
```

where `production_db` is the production database.

2. Set up a redirected restore to the test database. Issue the following command:

```
RESTORE DATABASE testdb REDIRECT
```

where *testdb* is the test database.

3. Modify the storage paths for *sg_hot* because no hot storage is available on the test database. Issue the following command:

```
SET STOGROUP PATHS FOR sg_hot ON '/hdd/path1', '/hdd/path2'
```

where *sg_hot* is the *sg_hot* storage group.

4. Proceed with the test database restore. Issue the following command:

```
RESTORE DATABASE testdb CONTINUE
```

5. Update the storage group name to correspond with the new paths. Use the following commands:

```
CONNECT TO testdb
```

```
RENAME STOGROUP sg_hot TO sg_cold
```

Database rebuild

Rebuilding databases

The ability to rebuild a database from table space backup images means that you no longer have to take as many full database backups. As databases grow in size, opportunities for taking a full database backup are becoming limited. With table space backup as an alternative, you no longer need to take full database backups as frequently. Instead, you can take more frequent table space backups and plan to use them, along with log files, in case of a disaster.

In a recovery situation, if you need to bring a subset of table spaces online faster than others, you can use rebuild to accomplish this. The ability to bring only a subset of table spaces online is especially useful in a test and production environment.

Rebuilding a database involves a series of potentially many restore operations. A rebuild operation can use a database image, or table space images, or both. It can use full backups, or incremental backups, or both. The initial restore operation restores the target image, which defines the structure of the database that can be restored (such as the table space set, the storage groups and the database configuration). For recoverable databases, rebuilding allows you to build a database that is connectable and that contains the subset of table spaces that you need to have online, while keeping table spaces that can be recovered at a later time offline.

The method you use to rebuild your database depends on whether it is recoverable or non-recoverable.

- If the database is recoverable, use one of the following methods:
 - Using a full or incremental database or table space backup image as your target, rebuild your database by restoring SYSCATSPACE and any other table spaces from the target image only using the **REBUILD** option. You can then roll your database forward to a point in time.
 - Using a full or incremental database or table space backup image as your target, rebuild your database by specifying the set of table spaces defined in the database at the time of the target image to be restored using the **REBUILD** option. SYSCATSPACE must be part of this set. This operation will restore those table spaces specified that are defined in the target image and then use the recovery history file to find and restore any other required backup images

for the remaining table spaces not in the target image automatically. Once the restores are complete, roll your database forward to a point in time.

- If the database is non-recoverable:
 - Using a full or incremental database backup image as your target, rebuild your database by restoring SYSCATSPACE and any other table spaces from the target image using the appropriate **REBUILD** syntax. When the restore completes you can connect to the database.

Specifying the target image

To perform a rebuild of a database, you start by issuing the **RESTORE** command, specifying the most recent backup image that you use as the target of the restore operation. This image is known as the target image of the rebuild operation, because it defines the structure of the database to be restored, including the table spaces that can be restored, the database configuration, and the log sequence. The rebuild target image is specified using the **TAKEN AT** parameter in the **RESTORE DATABASE** command. The target image can be any type of backup (full, table space, incremental, online or offline). The table spaces defined in the database at the time the target image was created will be the table spaces available to rebuild the database.

You must specify the table spaces you want restored using one of the following methods:

- Specify that you want all table spaces defined in the database to be restored and provide an exception list if there are table spaces you want to exclude
- Specify that you want all table spaces that have user data in the target image to be restored and provide an exception list if there are table spaces you want to exclude
- Specify the list of table spaces defined in the database that you want to restore

Once you know the table spaces you want the rebuilt database to contain, issue the **RESTORE** command with the appropriate **REBUILD** option and specify the target image to be used.

Rebuild phase

After you issue the **RESTORE** command with the appropriate **REBUILD** option and the target image has been successfully restored, the database is considered to be in the rebuild phase. After the target image is restored, any additional table space restores that occur will restore data into existing table spaces, as defined in the rebuilt database. These table spaces will then be rolled forward with the database at the completion of the rebuild operation.

If you issue the **RESTORE** command with the appropriate **REBUILD** option and the database does not exist, a new database is created based on the attributes in the target image. If the database does exist, you will receive a warning message notifying you that the rebuild phase is starting. You will be asked if you want to continue the rebuild operation or not.

The rebuild operation restores all initial metadata from the target image. This includes all data that belongs to the database and does not belong to the table space data or the log files. Examples of initial metadata are:

- Table spaces definitions
- The history file, which is a database file that records administrative operations

The rebuild operation also restores the database configuration. The target image sets the log chain that determines what images can be used for the remaining restores during the rebuild phase. Only images on the same log chain can be used.

If a database already exists on disk and you want the history file to come from the target image, then you should specify the **REPLACE HISTORY FILE** option. The history file on disk at this time is used by the automatic logic to find the remaining images needed to rebuild the database.

Once the target image is restored:

- if the database is recoverable, the database is put into rollforward pending state and all table spaces that you restore are also put into rollforward pending state. Any table spaces defined in the database but not restored are put in restore pending state.
- If the database is not recoverable, then the database and the table spaces restored will go into normal state. Any table spaces not restored are put in drop pending state, as they can no longer be recovered. For this type of database, the rebuild phase is complete.

For recoverable databases, the rebuild phase ends when the first **ROLLFORWARD DATABASE** command is issued and the rollforward utility begins processing log records. If a rollforward operation fails after starting to process log records and a restore operation is issued next, the restore is not considered to be part of the rebuild phase. Such restores should be considered as normal table space restores that are not part of the rebuild phase.

Automatic processing

After the target image is restored, the restore utility determines if there are remaining table spaces that need to be restored. If there are, they are restored using the same connection that was used for running the **RESTORE DATABASE** command with the **REBUILD** option. The utility uses the history file on disk to find the most recent backup images taken prior to the target image that contains each of the remaining table spaces that needs to be restored. The restore utility uses the backup image location data stored in the history file to restore each of these images automatically. These subsequent restores, which are table space level restores, can be performed only offline. If the image selected does not belong on the current log chain, an error is returned. Each table space that is restored from that image is placed in rollforward pending state.

The restore utility tries to restore all required table spaces automatically. In some cases, it will not be able to restore some table spaces due to problems with the history file, or an error will occur restoring one of the required images. In such a case, you can either finish the rebuild manually or correct the problem and reissue the rebuild.

If automatic rebuilding cannot complete successfully, the restore utility writes to the diagnostics log (**db2diag** log file) any information it gathered for the remaining restore steps. You can use this information to complete the rebuild manually.

If a database is being rebuilt, only containers belonging to table spaces that are part of the rebuild process will be acquired.

If any containers need to be redefined through redirected restore, you will need to set the new path and size of the new container for the remaining restores and the subsequent rollforward operation.

If the data for a table space restored from one of these remaining images cannot fit into the new container definitions, the table space is put into restore pending state and a warning message is returned at the end of the restore. You can find additional information about the problem in the diagnostic log.

Completing the rebuild phase

Once all the intended table spaces have been restored you have two options based on the configuration of the database. If the database is nonrecoverable, the database will be connectable and any table spaces restored will be online. Any table spaces that are in drop pending state can no longer be recovered and should be dropped if future backups will be performed on the database.

If the database is recoverable, you can issue the rollforward command to bring the table spaces that were restored online. If SYSCATSPACE has not been restored, the rollforward will fail and this table space will have to be restored before the rollforward operation can begin. This means that during the rebuild phase, SYSCATSPACE must be restored.

Note: In a partitioned database environment, SYSCATSPACE does not exist on non-catalog partitions so it cannot be rebuilt there. However, on the catalog partition, SYSCATSPACE must be one of the table spaces that is rebuilt, or the rollforward operation will not succeed.

Rolling the database forward brings the database out of rollforward pending state and rolls any table spaces in rollforward pending state forward. The rollforward utility will not operate on any table space in restore pending state.

The stop time for the rollforward operation must be a time that is later than the end time of the most recent backup image restored during the rebuild phase. An error will occur if any other time is given. If the rollforward operation is not able to reach the backup time of the oldest image that was restored, the rollforward utility will not be able to bring the database up to a consistent point, and the rollforward fails.

You must have all log files for the time frame between the earliest and most recent backup images available for the rollforward utility to use. The logs required are those logs which follow the log chain from the earliest backup image to the target backup image, as defined by the truncation array in the target image, otherwise the rollforward operation will fail. If any backup images more recent than the target image were restored during the rebuild phase, then the additional logs from the target image to the most recent backup image restored are required. If the logs are not made available, the rollforward operation will put those table spaces that were not reached by the logs into restore pending state. You can issue the **LIST HISTORY** command to show the restore rebuild entry with the log range that will be required by roll forward.

The correct log files must be available. If you rely on the rollforward utility to retrieve the logs, you must ensure that the DB2 Log Manager is configured to indicate the location from which log files can be retrieved. If the log path or archive path has changed, you need to use the **OVERFLOW LOG PATH** option of the **ROLLFORWARD DATABASE** command.

Use the **AND STOP** option of the **ROLLFORWARD DATABASE** command to make the database available when the rollforward command successfully completes. At this point, the database is no longer in rollforward pending state. If the rollforward operation begins, but an error occurs before it successfully completes, the rollforward operation stops at the point of the failure and an error is returned. The database remains in rollforward pending state. You must take steps to correct the problem (for example, fix the log file) and then issue another rollforward operation to continue processing.

If the error cannot be fixed, you will be able to bring the database up at the point of the failure by issuing the **ROLLFORWARD STOP** command. Any log data beyond that point in the logs will no longer be available once the **STOP** option is used. The database comes up at that point and any table spaces that have been recovered are online. Table spaces that have not yet been recovered are in restore pending state. The database is in the normal state.

You will have to decide what is the best way to recover the remaining table spaces in restore pending state. This could be by doing a new restore and roll forward of a table space or by reissuing the whole rebuild operation again. This will depend on the type of problems encountered. In the situation where SYSCATSPACE is one of the table spaces in restore pending state, the database will not be connectable.

Database rebuild and table space containers

During a database rebuild, only those table spaces that are part of the rebuild process have their containers acquired. The containers belonging to each table space are acquired at the time the table space user data is restored out of an image.

When the target image is restored, each table space known to the database at the time of the backup has its definitions restored. This means the database created by the rebuild has knowledge of the same table spaces it did at backup time. For those table spaces that should also have their user data restored from the target image, their containers are also be acquired at this time.

Any remaining table spaces that are restored through intermediate table space restores have their containers acquired at the time the image that contains the table space data is restored.

Rebuild with redirected restore

In the case of redirected restore, all table space containers must be defined during the restore of the target image. If you specify the **REDIRECT** option, control is given back to you to redefine your table space containers. If you have redefined table space containers using the **SET TABLESPACE CONTAINERS** command then those new containers are acquired at that time. Any table space containers that you have not redefined are acquired as normal, at the time the table space user data is restored out of an image.

If the data for a table space that is restored cannot fit into the new container definitions, the table space is put into restore-pending state and a warning (SQL2563W) is returned to you at the end of the restore. There will also be a message in the DB2 diagnostics log detailing the problem.

Database rebuild and temporary table spaces

Temporary table spaces are stored differently than other database components in a backup image. Because they are stored differently, temporary table spaces are rebuilt differently during a database restoration.

In general, a DB2 backup image is made up of the following components:

- Initial database metadata, such as the table space definitions, database configuration file, and history file.
- Data for non-temporary table spaces specified to the **BACKUP** utility
- Final database metadata such as the log file header
- Log files (if the **INCLUDE LOGS** option was specified)

In every backup image, whether it is a database or table space backup, a full or incremental (delta) backup, these core components can always be found.

A database backup image will contain all of the previously listed components, as well as data for every table space defined in the database at the time of the backup.

A table space backup image will always include the database metadata listed previously, but it will only contain data for those table spaces that are specified to the backup utility.

Temporary table spaces are treated differently than nontemporary table spaces. Temporary table space data is never backed up, but their existence is important to the framework of the database. Although temporary table space data is never backed up, the temporary table spaces are considered part of the database, so they are specially marked in the metadata that is stored with a backup image. This makes it look like they are in the backup image. In addition, the table space definitions hold information about the existence of any temporary table spaces.

Although no backup image ever contains data for a temporary table space, during a database rebuild operation when the target image is restored (regardless the type of image), temporary table spaces are also restored, only in the sense that their containers are acquired and allocated. The acquisition and allocation of containers is done automatically as part of the rebuild processing. As a result, when rebuilding a database, you cannot exclude temporary table spaces.

Choosing a target image for database rebuild

The rebuild target image should be the most recent backup image that you want to use as the starting point of your restore operation.

This image is known as the target image of the rebuild operation, because it defines the structure of the database to be restored, including the table spaces that can be restored, the database configuration, and the log sequence. It can be any type of backup (full, table space, incremental, online or offline).

The target image sets the log sequence (or log chain) that determines what images can be used for the remaining restores during the rebuild phase. Only images on the same log chain can be used.

The following examples illustrate how to choose the image you should use as the target image for a rebuild operation.

Suppose there is a database called SAMPLE that has the following table spaces in it:

- SYSCATSPACE (system catalogs)
- USERSP1 (user data table space)
- USERSP2 (user data table space)
- USERSP3 (user data table space)

Figure 56 on page 359 shows that the following database-level backups and table space-level backups that have been taken, in chronological order:

1. Full database backup DB1
2. Full table space backup TS1
3. Full table space backup TS2
4. Full table space backup TS3
5. Database restore and roll forward to a point between TS1 and TS2
6. Full table space backup TS4
7. Full table space backup TS5

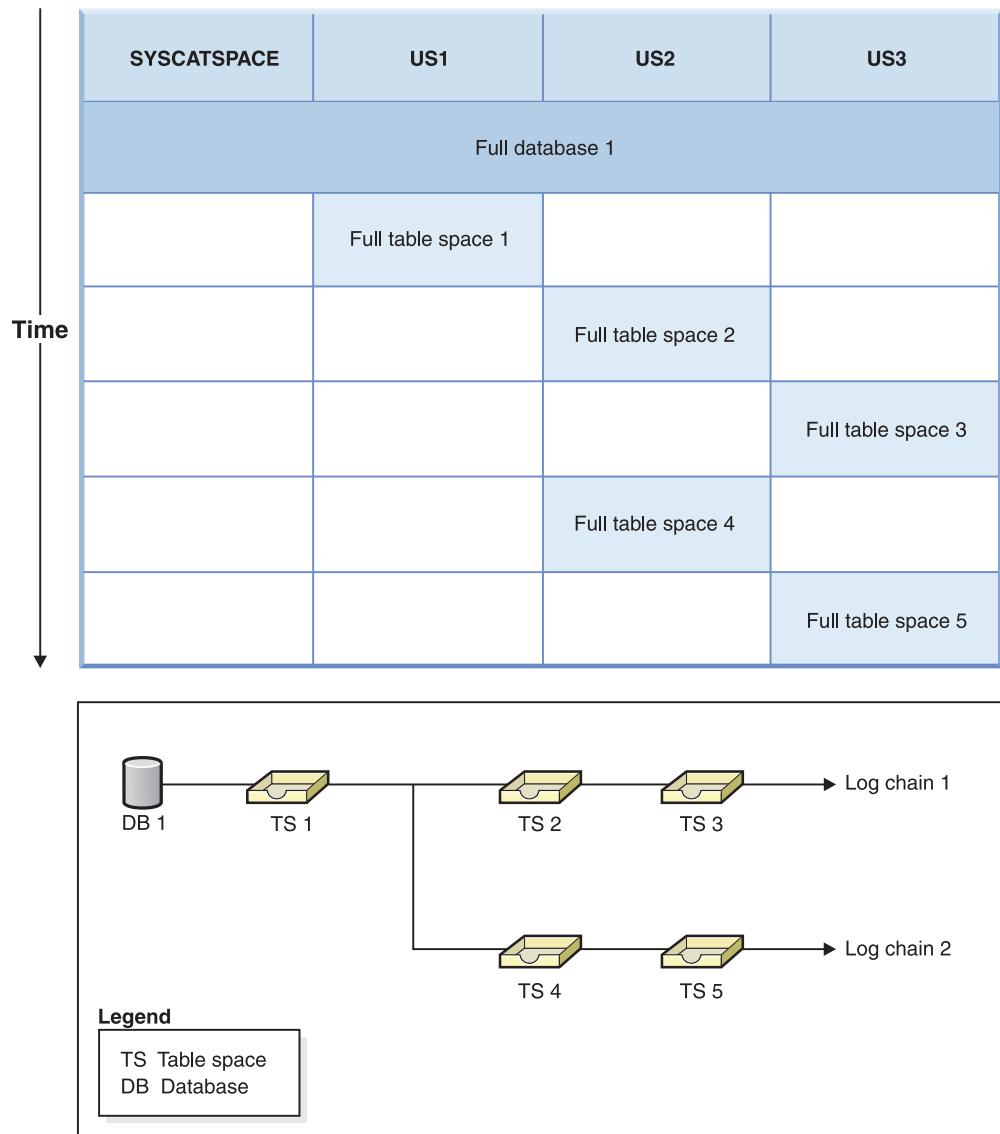


Figure 56. Database and table space-level backups of database SAMPLE

Example 1

The following example demonstrates the CLP commands you need to issue to rebuild database SAMPLE to the current point of time. First you need to choose the table spaces you want to rebuild. Since your goal is to rebuild the database to the current point of time you need to select the most recent backup image as your target image. The most recent backup image is image TS5, which is on log chain 2:

```

db2 restore db sample rebuild with all tablespaces in database taken at
      TS5 without prompting
db2 rollforward db sample to end of logs
db2 rollforward db sample stop
  
```

This restores backup images TS5, TS4, TS1 and DB1 automatically, then rolls the database forward to the end of log chain 2.

Note: All logs belonging to log chain 2 must be accessible for the rollforward operations to complete.

Example 2

This second example demonstrates the CLP commands you need to issue to rebuild database SAMPLE to the end of log chain 1. The target image you select should be the most recent backup image on log chain 1, which is TS3:

```
db2 restore db sample rebuild with all tablespaces in database  
    taken at TS3 without prompting  
db2 rollforward db sample to end of logs  
db2 rollforward db sample stop
```

This restores backup images TS3, TS2, TS1, and DB1 automatically, then rolls the database forward to the end of log chain 1.

Note:

- All logs belonging to log chain 1 must be accessible for the rollforward operations to complete.
- This command may fail because a log file is retrieved from a higher log chain (the rollforward utility always attempts to get log files from the highest log chain), you need to do the following steps:
 1. Extract the log files manually to the overflow log path.
 2. Run the **ROLLFORWARD** command. Include the parameters **-OVERFLOW LOG PATH**, to specify the location of the extracted log files, and **-NORETRIEVE**, to disable the retrieval of archived logs.

Choosing the wrong target image for rebuild

Suppose there is a database called SAMPLE2 that has the following table spaces in it:

- SYSCATSPACE (system catalogs)
- USERSP1 (user data table space)
- USERSP2 (user data table space)

Figure 57 on page 361 shows the backup log chain for SAMPLE2, which consists of the following backups:

1. BK1 is a full database backup, which includes all table spaces
2. BK2 is a full table space backup of USERSP1
3. BK3 is a full table space backup of USERSP2

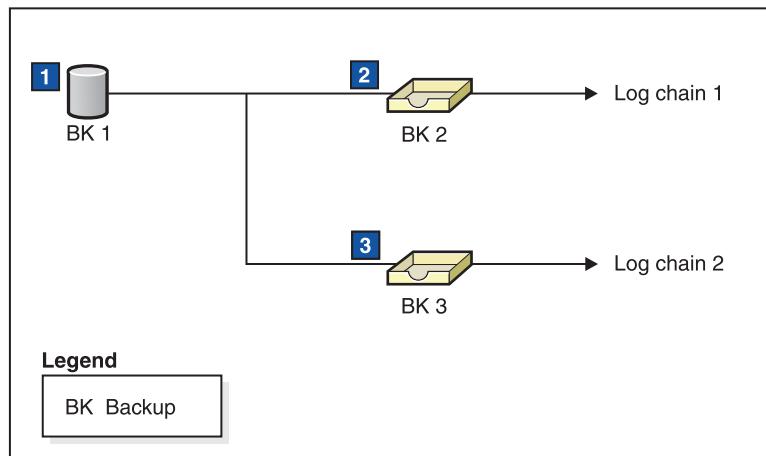


Figure 57. Backup log chain for database SAMPLE2

The following example demonstrates the CLP command you need to issue to rebuild the database from BK3 using table spaces SYSCATSPACE and USERSP2:

```
db2 restore db sample2 rebuild with tablespace (SYSCATSPACE,
USERSP2) taken at BK3 without prompting
```

Now suppose that after this restore completes, you decide that you also want to restore USERSP1, so you issue the following command:

```
db2 restore db sample2 tablespace (USERSP1) taken at BK2
```

This restore fails and provides a message that says BK2 is from the wrong log chain (SQL2154N). As you can see in Figure 57, the only image that can be used to restore USERSP1 is BK1. Therefore, you need to type the following command:

```
db2 restore db sample2 tablespace (USERSP1) taken at BK1
```

This succeeds so that database can be rolled forward accordingly.

Rebuilding selected table spaces

Rebuilding a database allows you to build a database that contains a subset of the table spaces that make up the original database.

About this task

Rebuilding only a subset of table spaces within a database can be useful in the following situations:

- In a test and development environment in which you want to work on only a subset of table spaces.
- In a recovery situation in which you need to bring table spaces that are more critical online faster than others, you can first restore a subset of table spaces then restore other table spaces at a later time.

To rebuild a database that contains a subset of the table spaces that make up the original database, consider the following example.

In this example, there is a database named SAMPLE that has the following table spaces:

- SYSCATSPACE (system catalogs)
- USERSP1 (user data table space)

- USERSP2 (user data table space)
- USERSP3 (user data table space)

Figure 58 shows that the following backups have been taken:

- BK1 is a backup of SYSCATSPACE and USERSP1
- BK2 is a backup of USERSP2 and USERSP3
- BK3 is a backup of USERSP3

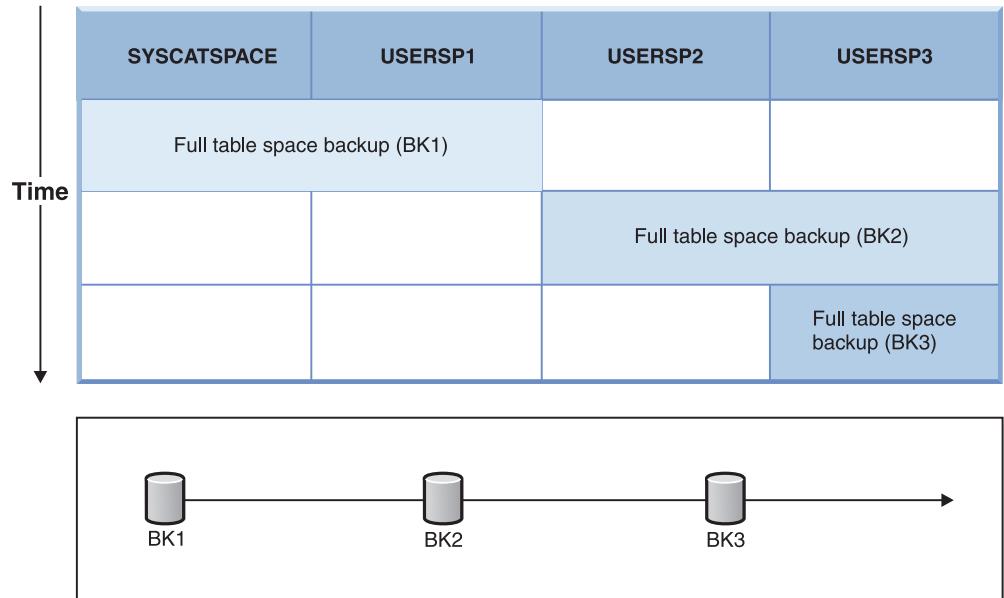


Figure 58. Backup images available for database SAMPLE

The following procedure demonstrates using the **RESTORE DATABASE** and **ROLLFORWARD DATABASE** commands, issued through the CLP, to rebuild just SYSCATSPACE and USERSP1 to end of logs:

```

db2 restore db mydb rebuild with all tablespaces in image
      taken at BK1 without prompting
db2 rollforward db mydb to end of logs
db2 rollforward db mydb stop
  
```

At this point the database is connectable and only SYSCATSPACE and USERSP1 are in NORMAL state. USERSP2 and USERSP3 are in restore-pending state. You can still restore USERSP2 and USERSP3 at a later time.

Rebuild and incremental backup images

You can rebuild a database using incremental images.

By default, the restore utility tries to use automatic incremental restore for all incremental images. This means that if you do not use the **INCREMENTAL** option of the **RESTORE DATABASE** command, but the target image is an incremental backup image, the restore utility will issue the rebuild operation using automatic incremental restore. If the target image is not an incremental image, but another required image is an incremental image then the restore utility will make sure those incremental images are restored using automatic incremental restore. The restore utility will behave in the same way whether you specify the **INCREMENTAL** option with the **AUTOMATIC** option or not.

If you specify the **INCREMENTAL** option but not the **AUTOMATIC** option, you will need to perform the entire rebuild process manually. The restore utility will just restore the initial metadata from the target image, as it would in a regular manual incremental restore. You will then need to complete the restore of the target image using the required incremental restore chain. Then you will need to restore the remaining images to rebuild the database.

It is recommended that you use automatic incremental restore to rebuild your database. Only in the event of a restore failure, should you attempt to rebuild a database using manual methods.

Rebuilding partitioned databases

To rebuild a partitioned database, rebuild each database partition separately. For each database partition, beginning with the catalog partition, first restore all the table spaces that you require. Any table spaces that are not restored are placed in restore pending state.

Once all the database partitions are restored, you then issue the **ROLLFORWARD DATABASE** command on the catalog partition to roll all of the database partitions forward.

About this task

Note: If, at a later date, you need to restore any table spaces that were not originally included in the rebuild phase, you need to make sure that when you subsequently roll the table space forward that the rollforward utility keeps all the data across the database partitions synchronized. If a table space is missed during the original restore and rollforward operation, it might not be detected until there is an attempt to access the data and a data access error occurs. You will then need to restore and roll the missing table space forward to get it back in sync with the rest of the partitions.

To rebuild a partitioned database using table space level backup images, consider the following example.

In this example, there is a recoverable database called SAMPLE with three database partitions:

- Database partition 1 contains table spaces SYSCATSPACE, USERSP1 and USERSP2, and is the catalog partition
- Database partition 2 contains table spaces USERSP1 and USERSP3
- Database partition 3 contains table spaces USERSP1, USERSP2 and USERSP3

The following backups have been taken, where BK xy represents backup number x on partition y :

- BK11 is a backup of SYSCATSPACE, USERSP1 and USERSP2
- BK12 is a backup of USERSP2 and USERSP3
- BK13 is a backup of USERSP1, USERSP2 and USERSP3
- BK21 is a backup of USERSP1
- BK22 is a backup of USERSP1
- BK23 is a backup of USERSP1
- BK31 is a backup of USERSP2
- BK33 is a backup of USERSP2

- BK42 is a backup of USERSP3
- BK43 is a backup of USERSP3

The following procedure demonstrates using the **RESTORE DATABASE** and **ROLLFORWARD DATABASE** commands, issued through the CLP, to rebuild the entire database to the end of logs.

Procedure

1. On database partition 1, issue a **RESTORE DATABASE** command with the **REBUILD** option:

```
db2 restore db sample rebuild with all tablespaces in database
taken at BK31 without prompting
```
2. On database partition 2, issue a **RESTORE DATABASE** command with the **REBUILD** option:

```
db2 restore db sample rebuild with tablespaces in database
taken at BK42 without prompting
```
3. On database partition 3, issue a **RESTORE DATABASE** command with the **REBUILD** option:

```
db2 restore db sample rebuild with all tablespaces in database
taken at BK43 without prompting
```
4. On the catalog partition, issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option:

```
db2 rollforward db sample to end of logs
```
5. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:

```
db2 rollforward db sample stop
```

What to do next

At this point the database is connectable on all database partitions and all table spaces are in NORMAL state.

Restrictions for database rebuild

You can use the **REBUILD** option to complete a set of restore commands, but it has restrictions that you need to be aware of.

The following list is a summary of database rebuild restrictions:

- One of the table spaces you rebuild must be SYSCATSPACE on the catalog partition.
- You must either issue commands using the command line processor (CLP) or use the corresponding application programming interfaces (APIs) to perform a rebuild operation.
- The **REBUILD** option cannot be used against a pre-Version 9.1 target image unless the image is that of an offline database backup. If the target image is an offline database backup, then only the table spaces in this image can be used for the rebuild. The database needs to be migrated after the rebuild operation successfully completes. Attempts to rebuild using any other type of pre-Version 9.1 target image result in an error.
- The **REBUILD** option cannot be issued against a target image from a different operating system than the one being restored on unless the target image is a full database backup. If the target image is a full database backup, then only the table spaces in this image can be used for the rebuild. Attempts to rebuild using any other type of target image from a different operating system than the one being restored on result in an error.

- The TRANSPORT option is incompatible with the REBUILD option.

Rebuild sessions - CLP examples

This topic provides a number of examples of rebuild operations.

Scenario 1

In the following examples, there is a recoverable database called MYDB with the following table spaces in it:

- SYSCATSPACE (system catalogs)
- USERSP1 (user data table space)
- USERSP2 (user data table space)
- USERSP3 (user data table space)

The following backups have been taken:

- BK1 is a backup of SYSCATSPACE and USERSP1
- BK2 is a backup of USERSP2 and USERSP3
- BK3 is a backup of USERSP3

Example 1

The following rebuilds the entire database to the most recent point in time:

1. Issue a **RESTORE DATABASE** command with the **REBUILD** option:

```
db2 restore db mydb rebuild with all tablespaces in database
taken at BK3 without prompting
```
2. Issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option (this assumes all logs have been saved and are accessible):

```
db2 rollforward db mydb to end of logs
```
3. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:

```
db2 rollforward db mydb stop
```

At this point the database is connectable and all table spaces are in NORMAL state.

Example 2

The following rebuilds just SYSCATSPACE and USERSP2 to a point in time (where end of BK3 is less recent than the point in time, which is less recent than end of logs):

1. Issue a **RESTORE DATABASE** command with the **REBUILD** option and specify the table spaces you want to include.

```
db2 restore db mydb rebuild with tablespace (SYSCATSPACE, USERSP2)
taken at BK2 without prompting
```
2. Issue a **ROLLFORWARD DATABASE** command with the **TO PIT** option (this assumes all logs have been saved and are accessible):

```
db2 rollforward db mydb to PIT
```
3. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:

```
db2 rollforward db mydb stop
```

At this point the database is connectable and only SYSCATSPACE and USERSP2 are in NORMAL state. USERSP1 and USERSP3 are in RESTORE_PENDING state.

To restore USERSP1 and USERSP3 at a later time, using normal table space restores (without the **REBUILD** option):

1. Issue the **RESTORE DATABASE** command *without* the **REBUILD** option and specify the table space you want to restore. First restore USERSP1:

```
db2 restore db mydb tablespace (USERSP1) taken at BK1 without prompting
```

2. Then restore USERSP3:

```
db2 restore db mydb tablespace taken at BK3 without prompting
```

3. Issue a **ROLLFORWARD DATABASE** command with the **END OF LOGS** option and specify the table spaces to be restored (this assumes all logs have been saved and are accessible):

```
db2 rollforward db mydb to end of logs tablespace (USERSP1, USERSP3)
```

The rollforward will replay all logs up to the PIT and then stop for these two table spaces since no work has been done on them since the first rollforward.

4. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:

```
db2 rollforward db mydb stop
```

Example 3

The following rebuilds just SYSCATSPACE and USERSP1 to end of logs:

1. Issue a **RESTORE DATABASE** command with the **REBUILD** option:

```
db2 restore db mydb rebuild with all tablespaces in image  
taken at BK1 without prompting
```

2. Issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option (this assumes all logs have been saved and are accessible):

```
db2 rollforward db mydb to end of logs
```

3. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:

```
db2 rollforward db mydb stop
```

At this point the database is connectable and only SYSCATSPACE and USERSP1 are in NORMAL state. USERSP2 and USERSP3 are in RESTORE_PENDING state.

Example 4

In the following example, the backups BK1 and BK2 are no longer in the same location as stated in the history file, but this is not known when the rebuild is issued.

1. Issue a **RESTORE DATABASE** command with the **REBUILD** option, specifying that you want to rebuild the entire database to the most recent point in time:

```
db2 restore db mydb rebuild with all tablespaces in database  
taken at BK3 without prompting
```

At this point, the target image is restored successfully, but an error is returned from the restore utility stating it could not find a required image.

2. You must now complete the rebuild manually. Since the database is in the rebuild phase this can be done as follows:

- a. Issue a **RESTORE DATABASE** command and specify the location of the BK1 backup image:

```
db2 restore db mydb tablespace taken at BK1 from location  
without prompting
```

- b. Issue a **RESTORE DATABASE** command and specify the location of the BK2 backup image:

```
db2 restore db mydb tablespace (USERSP2) taken at BK2 from  
location without prompting
```

- c. Issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option (this assumes all logs have been saved and are accessible):


```
db2 rollforward db mydb to end of logs
```
- d. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:


```
db2 rollforward db mydb stop
```

At this point the database is connectable and all table spaces are in NORMAL state.

Example 5

In this example, table space USERSP3 contains independent data that is needed for generating a specific report, but you do not want the report generation to interfere with the original database. In order to gain access to the data but not affect the original database, you can use **REBUILD** to generate a new database with just this table space and SYSCATSPACE. SYSCATSPACE is also required so that the database will be connectable after the restore and roll forward operations.

To build a new database with the most recent data in SYSCATSPACE and USERSP3:

1. Issue a **RESTORE DATABASE** command with the **REBUILD** option, and specify that table spaces SYSCATSPACE and USERSP3 are to be restored to a new database, NEWDB:


```
db2 restore db mydb rebuild with tablespace (SYSCATSPACE, USERSP3)
      taken at BK3 into newdb without prompting
```
2. Issue a **ROLLFORWARD DATABASE** command on NEWDB with the **TO END OF LOGS** option (this assumes all logs have been saved and are accessible):


```
db2 rollforward db newdb to end of logs
```
3. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:


```
db2 rollforward db newdb stop
```

At this point the new database is connectable and only SYSCATSPACE and USERSP3 are in NORMAL state. USERSP1 and USERSP2 are in RESTORE_PENDING state.

Note: If container paths are an issue between the current database and the new database (for example, if the containers for the original database need to be altered because the file system does not exist or if the containers are already in use by the original database) then you will need to perform a redirected restore. This example assumes the default autostorage database paths are used for the table spaces.

Scenario 2

In the following example, there is a recoverable database called MYDB that has SYSCATSPACE and one thousand user table spaces named Txxxx, where xxxx stands for the table space number (for example, T0001). There is one full database backup image (BK1)

Example 6

The following restores all table spaces except T0999 and T1000:

1. Issue a **RESTORE DATABASE** command with the **REBUILD** option:

```
db2 restore db mydb rebuild with all tablespaces in image except
      tablespace (T0999, T1000) taken at BK1 without prompting
```

2. Issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option (this assumes all logs have been saved and are accessible):

```
db2 rollforward db mydb to end of logs
```

3. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:

```
db2 rollforward db mydb stop
```

At this point the database will be connectable and all table spaces except T0999 and T1000 will be in NORMAL state. T0999 and T1000 will be in RESTORE_PENDING state.

Scenario 3

The examples in this scenario demonstrate how to rebuild a recoverable database using incremental backups. In the following examples, there is a database called MYDB with the following table spaces in it:

- SYSCATSPACE (system catalogs)
- USERSP1 (data table space)
- USERSP2 (user data table space)
- USERSP3 (user data table space)

The following backups have been taken:

- FULL1 is a full backup of SYSCATSPACE, USERSP1, USERSP2 and USERSP3
- DELTA1 is a delta backup of SYSCATSPACE and USERSP1
- INCR1 is an incremental backup of USERSP2 and USERSP3
- DELTA2 is a delta backup of SYSCATSPACE, USERSP1, USERSP2 and USERSP3
- DELTA3 is a delta backup of USERSP2
- FULL2 is a full backup of USERSP1

Example 7

The following rebuilds just SYSCATSPACE and USERSP2 to the most recent point in time using incremental automatic restore.

1. Issue a **RESTORE DATABASE** command with the **REBUILD** option. The **INCREMENTAL AUTO** option is optional. The restore utility will detect what the granularity of the image is and use automatic incremental restore if it is required.

```
db2 restore db mydb rebuild with tablespace (SYSCATSPACE, USERSP2)
incremental auto taken at DELTA3 without prompting
```

2. Issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option (this assumes all logs have been saved and are accessible):

```
db2 rollforward db mydb to end of logs
```

3. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:

```
db2 rollforward db mydb stop
```

At this point the database is connectable and only SYSCATSPACE and USERSP2 are in NORMAL state. USERSP1 and USERSP3 are in RESTORE_PENDING state.

Example 8

The following rebuilds the entire database to the most recent point in time using incremental automatic restore.

- Issue a **RESTORE DATABASE** command with the **REBUILD** option. The **INCREMENTAL AUTO** option is optional. The restore utility will detect what the granularity of the image is and use automatic incremental restore if it is required.

```
db2 restore db mydb rebuild with all tablespaces in database
incremental auto taken at DELTA3 without prompting
```

- Issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option (this assumes all logs have been saved and are accessible):

```
db2 rollforward db mydb to end of logs
```

- Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:

```
db2 rollforward db mydb stop
```

At this point the database is connectable and all table spaces are in NORMAL state.

Example 9

The following rebuilds the entire database, except for USERSP3, to the most recent point in time.

- Issue a **RESTORE DATABASE** command with the **REBUILD** option. Although the target image is a non-incremental image, the restore utility will detect that the required rebuild chain includes incremental images and it will automatically restore those images incrementally.

```
db2 restore db mydb rebuild with all tablespaces in database except
tablespace (USERSP3) taken at FULL2 without prompting
```

- Issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option (this assumes all logs have been saved and are accessible):

```
db2 rollforward db mydb to end of logs
```

- Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:

```
db2 rollforward db mydb stop
```

Scenario 4

The examples in this scenario demonstrate how to rebuild a recoverable database using backup images that contain log files. In the following examples, there is a database called MYDB with the following table spaces in it:

- SYSCATSPACE (system catalogs)
- USERSP1 (user data table space)
- USERSP2 (user data table space)

Example 10

The following rebuilds the database with just SYSCATSPACE and USERSP2 to the most recent point in time. There is a full online database backup image (BK1), which includes log files.

- Issue a **RESTORE DATABASE** command with the **REBUILD** option:

```
db2 restore db mydb rebuild with tablespace (SYSCATSPACE, USERSP2)
taken at BK1 logtarget /logs without prompting
```

- Issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option (this assumes all logs after the end of BK1 have been saved and are accessible):

```
db2 rollforward db mydb to end of logs overflow log path (/logs)
```

- Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:

```
db2 rollforward db mydb stop
```

At this point the database is connectable and only SYSCATSPACE and USERSP2 are in NORMAL state. USERSP1 is in RESTORE_PENDING state.

Example 11

The following rebuilds the database to the most recent point in time. There are two full online table space backup images that include log files:

- BK1 is a backup of SYSCATSPACE, using log files 10-45
 - BK2 is a backup of USERSP1 and USERSP2, using log files 64-80
1. Issue a **RESTORE DATABASE** command with the **REBUILD** option:

```
db2 restore db mydb rebuild with all tablespaces in database  
taken at BK2 logtarget /logs without prompting
```

The rollforward operation will start at log file 10, which it will always find in the overflow log path if not in the primary log file path. The log range 46-63, since they are not contained in any backup image, will need to be made available for roll forward.

2. Issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option, using the overflow log path for log files 64-80:

```
db2 rollforward db mydb to end of logs overflow log path (/logs)
```

3. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:

```
db2 rollforward db mydb stop
```

At this point the database is connectable and all table spaces are in NORMAL state.

Scenario 5

In the following examples, there is a recoverable database called MYDB with the following table spaces in it:

- SYSCATSPACE (0), SMS system catalog (relative container)
- USERSP1 (1) DMS user data table space (absolute container /usersp2)
- USERSP2 (2) DMS user data table space (absolute container /usersp3)

The following backups have been taken:

- BK1 is a backup of SYSCATSPACE
- BK2 is a backup of USERSP1 and USERSP2
- BK3 is a backup of USERSP2

Example 12

The following rebuilds the entire database to the most recent point in time using redirected restore.

1. Issue a **RESTORE DATABASE** command with the **REBUILD** option:

```
db2 restore db mydb rebuild with all tablespaces in database  
taken at BK3 redirect without prompting
```
2. Issue a **SET TABLESPACE CONTAINERS** command for each table space whose containers you want to redefine. For example:

```
db2 set tablespace containers for 3 using (file '/newusersp1' 10000)
```
3.

```
db2 set tablespace containers for 4 using (file '/newusersp2' 15000)
```
4. Issue a **RESTORE DATABASE** command with the **CONTINUE** option:

```
db2 restore db mydb continue
```

5. Issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option (this assumes all logs have been saved and are accessible):

```
db2 rollforward db mydb to end of logs
```

6. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:

```
db2 rollforward db mydb stop
```

At this point the database is connectable and all table spaces are in NORMAL state.

Scenario 6

In the following examples, there is a database called MYDB with three database partitions:

- Database partition 1 contains table spaces SYSCATSPACE, USERSP1 and USERSP2, and is the catalog partition
- Database partition 2 contains table spaces USERSP1 and USERSP3
- Database partition 3 contains table spaces USERSP1, USERSP2 and USERSP3

The following backups have been taken, where BK xy represents backup number x on partition y :

- BK11 is a backup of SYSCATSPACE, USERSP1 and USERSP2
- BK12 is a backup of USERSP2 and USERSP3
- BK13 is a backup of USERSP1, USERSP2 and USERSP3
- BK21 is a backup of USERSP1
- BK22 is a backup of USERSP1
- BK23 is a backup of USERSP1
- BK31 is a backup of USERSP2
- BK33 is a backup of USERSP2
- BK42 is a backup of USERSP3
- BK43 is a backup of USERSP3

Example 13

The following rebuilds the entire database to the end of logs.

1. On database partition 1, issue a **RESTORE DATABASE** command with the **REBUILD** option:

```
db2 restore db mydb rebuild with all tablespaces in database  
taken at BK31 without prompting
```

2. On database partition 2, issue a **RESTORE DATABASE** command with the **REBUILD** option:

```
db2 restore db mydb rebuild with tablespaces in database taken at  
BK42 without prompting
```

3. On database partition 3, issue a **RESTORE DATABASE** command with the **REBUILD** option:

```
db2 restore db mydb rebuild with all tablespaces in database  
taken at BK43 without prompting
```

4. On the catalog partition, issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option (assumes all logs have been saved and are accessible on all database partitions):

```
db2 rollforward db mydb to end of logs
```

5. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:

```
db2 rollforward db mydb stop
```

At this point the database is connectable on all database partitions and all table spaces are in NORMAL state.

Example 14

The following rebuilds SYSCATSPACE, USERSP1 and USERSP2 to the most recent point in time.

1. On database partition 1, issue a **RESTORE DATABASE** command with the **REBUILD** option:

```
db2 restore db mydb rebuild with all tablespaces in database  
taken at BK31 without prompting
```

2. On database partition 2, issue a **RESTORE DATABASE** command with the **REBUILD** option:

```
db2 restore db mydb rebuild with all tablespaces in image taken at  
BK22 without prompting
```

3. On database partition 3, issue a **RESTORE DATABASE** command with the **REBUILD** option:

```
db2 restore db mydb rebuild with all tablespaces in image taken at  
BK33 without prompting
```

Note: this command omitted USERSP1, which is needed to complete the rebuild operation.

4. On the catalog partition, issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option:

```
db2 rollforward db mydb to end of logs
```

5. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:

```
db2 rollforward db mydb stop
```

The rollforward succeeds and the database is connectable on all database partitions. All table spaces are in NORMAL state, except USERSP3, which is in RESTORE PENDING state on all database partitions on which it exists, and USERSP1, which is in RESTORE PENDING state on database partition 3.

When an attempt is made to access data in USERSP1 on database partition 3, a data access error will occur. To fix this, USERSP1 will need to be recovered:

- a. On database partitions 3, issue a **RESTORE DATABASE** command, specifying a backup image that contains USERSP1:

```
db2 restore db mydb tablespace taken at BK23 without prompting
```

- b. On the catalog partition, issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option and the **AND STOP** option:

```
db2 rollforward db mydb to end of logs on dbpartitionnum (3) and stop
```

At this point USERSP1 on database partition 3 can have its data accessed since it is in NORMAL state.

Scenario 7

In the following examples, there is a *nonrecoverable* database called MYDB with the following table spaces:

- SYSCATSPACE (0), SMS system catalog
- USERSP1 (1) DMS user data table space
- USERSP2 (2) DMS user data table space

There is just one backup of the database, BK1:

Example 15

The following demonstrates using rebuild on a nonrecoverable database.

Rebuild the database using only SYSCATSPACE and USERSP1:

```
db2 restore db mydb rebuild with tablespace (SYSCATSPACE, USERSP1)
taken at BK1 without prompting
```

Following the restore, the database is connectable. If you issue the **LIST TABLESPACES** command or the MON_GET_TABLESPACE table function, you see that the SYSCATSPACE and USERSP1 are in NORMAL state, while USERSP2 is in DELETE_PENDING/OFFLINE state. You can now work with the two table spaces that are in NORMAL state.

If you want to do a database backup, you will first need to drop USERSP2 using the DROP TABLESPACE statement, otherwise, the backup will fail.

To restore USERSP2 at a later time, you need to reissue a database restore from BK1.

Transporting database schemas

Transporting a database schema involves taking a backup image of a database and restoring the database schema to a different, existing database.

When you transport a database schema, the database objects in the transported schema are re-created to reference the new database, and the data is restored to the new database.

A database schema must be transported in its entirety. If a table space contains both the schema you want to transport, as well as another schema, you must transport all data objects from both schemas. These sets of schemas that have no references to other database schemas are called *transportable sets*. The data in the table spaces and logical objects in the schemas in a transportable set reference only table spaces and schemas in the transportable set. For example, tables have table dependencies only on other tables in the transportable set.

The following diagram illustrates a database with several table spaces and schemas. In the diagram, the table spaces that are referenced by the schemas are above the schemas. Some schemas reference multiple table spaces and some table spaces are referenced by multiple schemas.

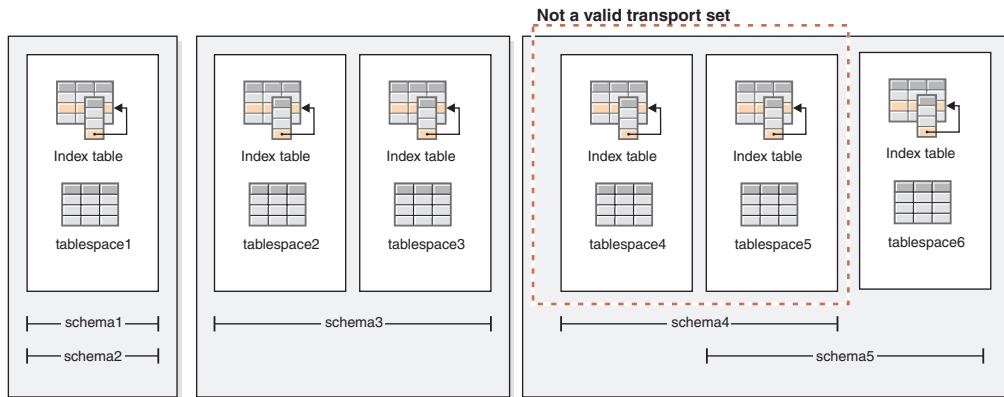


Figure 59. Sets of table spaces and schemas

The following combinations of table spaces and schemas are valid transportable sets:

- **tablespace1** with **schema1** and **schema2**
- **tablespace2** and **tablespace3** with **schema3**
- **tablespace4**, **tablespace5**, and **tablespace6**, with **schema4** and **schema5**
- A combination of valid transportable sets also constitutes a valid transportable set:
 - **tablespace1**, **tablespace2**, and **tablespace3**, with **schema1**, **schema2**, and **schema3**

The set **tablespace4** and **tablespace5** with **schema4** is not a valid transportable set because there are references between **tablespace5** and **schema5** and between **schema5** and **tablespace6**. The set requires **tablespace6** with **schema5** to be a valid transportable set.

You can transport database schemas by using the **RESTORE** command with the **TRANSPORT** parameter.

When you transport database schemas, a temporary database is created and named as a part of the transport operation. This *transport staging database* is used to extract logical objects from the backup image so that they can be re-created on the target database. If logs are included in the backup image, they are also used to bring the staging database to a point of transactional consistency. The ownership of the transported table spaces is then transferred to the target database.

Considerations about the database objects re-created when transporting database schemas

Review the following information related to the re-creation of database objects when you are transporting database schemas:

Table 25. Transport considerations for specific database objects

Database object	Consideration when transporting schemas
SQL routines (not external routines using SQL)	A new copy of the SQL routine is created in the target database. For SQL stored procedures, additional catalog space is consumed because an additional copy of the stored procedure byte code is created in the new database.

Table 25. Transport considerations for specific database objects (continued)

Database object	Consideration when transporting schemas
External routines	A new catalog entry is created for each routine. This catalog entry references the same binary file as the original source routine. The RESTORE command does not copy the external routine binary file from the source system.
Source tables in states causing access problems	For tables that are not in normal state at the time the backup image was generated, such as tables in check pending state or load pending state, the data from those tables might not be accessible in the target database. To avoid having this inaccessible data, you can move the tables to normal state in the source database before schema transport.
Tables containing the data capture attribute	Source tables with data capture enabled are transported to the target database with the data capture attribute and continue to log interdatabase data replication information. However, replicated tables do not extract information from this table. You have the option of registering the new target table to act as a replication source after the RESTORE command completes.
Tables using label-based access control (LBAC)	When transporting data that is protected by LBAC, the transport operation re-creates the LBAC objects on the target database. If LBAC objects of the same name exist on the target database, the transport operation fails. To ensure that restricted data access is not compromised, the transport operation does not use existing LBAC objects on the target database.
Temporary table spaces	If there are any system temporary table spaces that are defined with the source backup image and the transport operation excludes them from the table space list, these system temporary table spaces are still created in the staging database but not the final target database. As a result, you must issue the SET TABLESPACE CONTAINERS command for these system temporary table spaces in order to provide valid containers to complete the restore operation, just as you would for any table spaces that are specified within the table space list.

When you transport table spaces, a log record with a special format is created on the target database. This format cannot be read by previous DB2 versions. If you transport table spaces and then downgrade to a version earlier than DB2 Version 9.7 Fix Pack 2, then you cannot recover the target database that contains the table spaces that were transported. To ensure that the target database is compatible with earlier DB2 versions, you can roll forward the target database to a point in time before the transport operation.

Important: If database rollforward detects a table space schema transport log record, the corresponding transported table space is taken offline and moved into drop pending state. This is because database does not have complete logs of transported table spaces to rebuild transported table spaces and their contents. You can take a full backup of the target database after transport completes, so subsequent rollforward does not pass the point of schema transport in the log stream.

Transportable objects

When you transport data from a backup image to a target database, there are two main results. The physical and logical objects in the table spaces that you are restoring are re-created in the target database, and the table space definitions and containers are added to the target database.

The following logical objects are re-created:

- Tables, created global temporary tables, and materialized query tables
- Normal and statistical views
- The following types of generated columns:
 - Expression
 - Identity
 - Row change timestamp
 - Row change token
- User-defined functions and generated functions
- Functions and procedures except for external routine executables
- User-defined types
- The following types of constraints:
 - Check
 - Foreign key
 - Functional dependency
 - Primary
 - Unique
- Indexes
- Triggers
- Sequences
- Object authorizations, privileges, security, access control, and audit configuration
- Table statistics, profiles, and hints
- Packages

The following components of a schema are not created on the target database:

- Aliases
- Created global variables
- External routine executable files
- Functional mappings and templates
- Hierarchy tables
- Index extensions
- Jobs
- Methods
- Nicknames
- OLE DB external functions
- Range-partitioned tables
- Servers
- Sourced procedures
- Structured types
- System catalogs
- Typed tables and typed views
- Usage lists
- Wrappers

Transport examples

You can use the **RESTORE DATABASE** command with the **TRANSPORT** option to copy a set of table spaces and SQL schemas from one database to another database.

The following examples use a database named ORIGINALDB as source of the backup image and the target database TARGETDB.

The following illustration shows the ORIGINALDB table spaces and schemas:

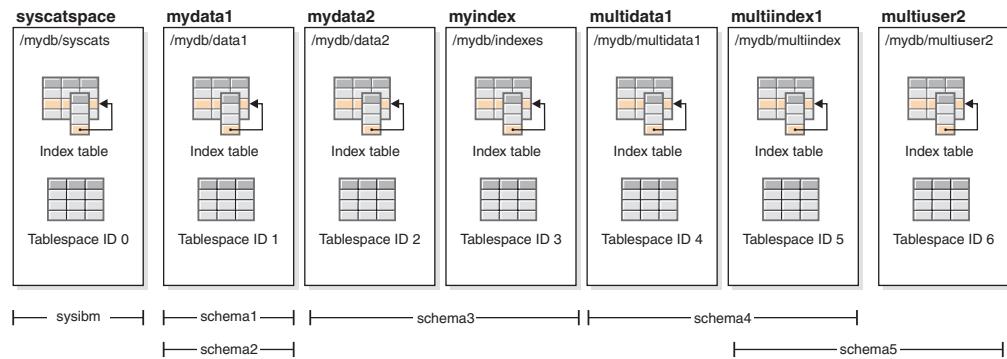


Figure 60. ORIGINALDB database

The originalDB database contains the following valid transportable sets:

- mydata1; schema1 + schema2
- mydata2 + myindex; schema3
- multidata1 + multiindex1 + multiuser2; schema4 + schema5
- A combination of valid transportable sets also constitutes a valid transportable set:
 - mydata1 + mydata2 + myindex; schema1 + schema + schema3

The following illustration shows the TARGETDB table spaces and schemas:

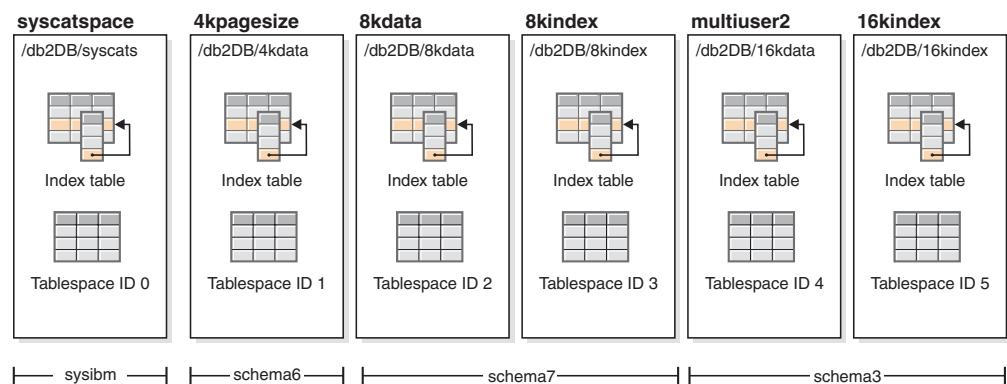


Figure 61. TARGETDB database

If the sources and target databases contain any schemas with the same schema name, or any table spaces of the table space name, then you cannot transport that schema or table space to the target database. Issuing a transport operation that contains a schema or a table space that has the same name as a schema or a table space on the target database will cause the transport operation to fail. For example,

even though the following grouping is a valid transportable set, it cannot be directly transported to the target database:

- mydata2 + myindex; schema3 (schema3 exists in both the source and target databases)

If there exists a single online backup image for ORIGINALDB that contains all of the table spaces in the database, then this will be the source for the transport. This also applies to table space level backup images.

You can redirect the container paths for the table spaces being transported. This is especially important if database relative paths were used.

Examples

Example 1: Successfully transport the schemas schema1 and schema2 in the mydata1 table space into TARGETDB.

```
db2 restore db originaldb tablespace (mydata1) schema(schema1,schema2)
  from <Media_Target_clause> taken at <date-time>
    transport into targetdb redirect
```

```
db2 list tablespaces
db2 set tablespace containers for <tablespace ID for mydata1>
  using (path '/db2DB/data1')
```

```
db2 restore db originaldb continue
```

The resulting TARGETDB will contain the mydata1 table space and schema1 and schema2.

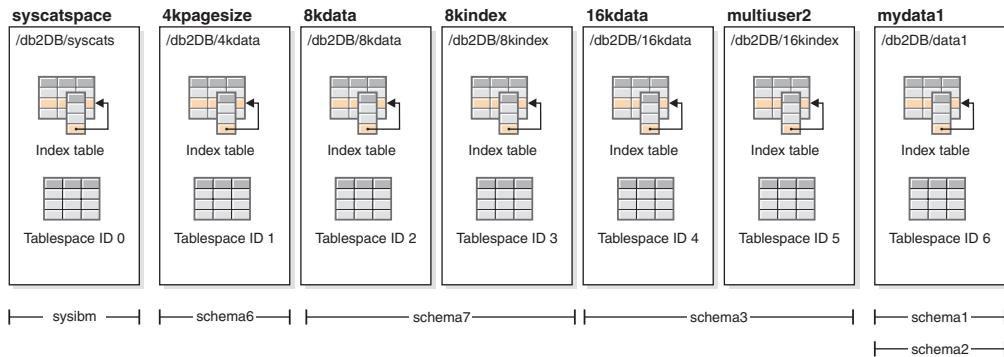


Figure 62. TARGETDB database after transport

Example 2: Transport the schema schema3 in the mydata2 and myindex table spaces into TARGETDB. You cannot transport a schema that already exists on the target database.

```
db2 restore db originaldb tablespace (mydata2,myindex) schema(schema3)
  transport into targetdb
```

The transport operation will fail because the schema schema3 already exists on the target database. TARGETDB will remain unchanged. SQLCODE=SQL2590N rc=3.

Example 3: Transport the schemas schema4 and schema5 in the multidata1, multiindex1, and multiuser2 table spaces into TARGETDB. You cannot transport a table space that already exists on the target database.

```
db2 restore db originaldb tablespace (multidata1,multiindex1,multiuser2)
  schema(schema4,schema5) transport into targetdb
```

The transport operation will fail and TARGETDB will remain unchanged because table space multiuser2 already exists on the target database. SQLCODE=SQL2590N rc=3.

Example 4: Transport the myindex table space into TARGETDB. You cannot transport partial schemas.

```
db2 restore db originaldb tablespace (myindex) schema(schema3)
      transport into targetdb
```

The list of table spaces and schemas being transported is not a valid transportable set. The transport operation will fail and TARGETDB will remain unchanged. SQLCODE=SQL2590N rc=1.

Example 5: Restore the syscatspace table space into TARGETDB. You cannot transport system catalogs.

```
db2 restore db originaldb tablespace (syscatspace) schema(sysibm)
      transport into targetdb
```

The transport operation will fail because the system catalogs can not be transported. SQLCODE=SQL2590N rc=4. You can transport user defined table spaces or restore the system catalogs with the RESTORE DATABASE command without specifying the transport option.

Example 6: You cannot restore into a target database that does not exist on the system.

```
db2 restore db originaldb tablespace (mydata1) schema(schema1,schema2)
      transport into notexists
```

The transport operation will fail. Table spaces cannot be transported to a target database that does not exist.

Troubleshooting: transporting schemas

If an error occurs on either the staging or target database, you must redo the entire restore operation. All failures that occur are logged in the db2diag log file on the target server. Review the **db2diag** log before reissuing the **RESTORE** command.

Dealing with errors

Errors occurring during restore are handled in various ways depending on the type of object being copied and the phase of transport. There might be circumstances, such as a power failure, in which not everything is cleaned up.

The transport operation consists of the following phases:

- Staging database creation
- Physical table space container restoration
- Rollforward processing
- Schema validation
- Transfer of ownership of the table space containers
- Schema re-creation in target database
- Dropping the staging database (if the **STAGE IN** parameter is not specified)

If any errors are logged at the end of the schema re-creation phase, about transporting physical objects, then the restore operation fails and an error is returned. All object creation on the target database is rolled back, and all internally

created tables are cleaned up on the staging database. The rollback occurs at the end of the re-create phase, to allow all possible errors to be recorded into the **db2diag** log file. You can investigate all errors returned before reissuing the command.

The staging database is dropped automatically after success or failure. However, it is not dropped in the event of failure if the **STAGE IN** parameter is specified. The staging database must be dropped before the staging database name can be reused.

Monitoring the progress of restore operations

You can use the **LIST UTILITIES** command to monitor restore operations on a database.

Procedure

Issue the **LIST UTILITIES** command and specify the **SHOW DETAIL** parameter

```
LIST UTILITIES SHOW DETAIL
```

Results

For restore operations, an initial estimate is not given. Instead, UNKNOWN is specified. As each buffer is read from the image, the actual number of bytes read is updated. For automatic incremental restore operations where multiple images might be restored, the progress is tracked by using phases. Each phase represents an image to be restored from the incremental chain. Initially, only one phase is indicated. After the first image is restored, the total number of phases will be indicated. As each image is restored the number of phases completed is updated, as is the number of bytes processed.

Example

The following is an example of the output for monitoring the performance of a restore operation:

ID	= 6
Type	= RESTORE
Database Name	= SAMPLE
Partition Number	= 0
Description	= db
Start Time	= 08/04/2011 12:24:47.494191
State	= Executing
Invocation Type	= User
Progress Monitoring:	
Completed Work	= 4096 bytes
Start Time	= 08/04/2011 12:24:47.494197

Optimizing restore performance

When you perform a restore operation, DB2 database products will automatically choose an optimal value for the number of buffers, the buffersize and the parallelism settings. The values will be based on the amount of utility heap memory available, the number of processors available and the database configuration.

Therefore, depending on the amount of storage available on your system, you should consider allocating more memory by increasing the **util_heap_sz** configuration parameter. The objective is to minimize the time it takes to complete

a restore operation. Unless you explicitly enter a value for the following **RESTORE DATABASE** command parameters, DB2 database products will select one for them:

- **WITH** *num-buffers* **BUFFERS**
- **PARALLELISM** *n*
- **BUFFER** *buffer-size*

For restore operations, a multiple of the buffer size used by the backup operation will always be used. You can specify a buffer size when you issue the **RESTORE DATABASE** command but you need to make sure that it is a multiple of the backup buffer size.

You can also choose to do any of the following to reduce the amount of time required to complete a restore operation:

- Increase the restore buffer size.

The restore buffer size must be a positive integer multiple of the backup buffer size specified during the backup operation. If an incorrect buffer size is specified, the buffers allocated will be the smallest acceptable size.

- Increase the number of buffers.

The value you specify must be a multiple of the buffersize that was used for the backup, otherwise it will be rounded down to the closest multiple of the backup buffersize.

- Increase the value of the **PARALLELISM** parameter.

This will increase the number of buffer manipulators (BM) that will be used to write to the database during the restore operation.

- Increase the utility heap size

This will increase the memory that can be used simultaneously by the other utilities.

Chapter 15. Rolling forward databases

Rolling forward a database recovers the transactions that were logged after the last backup command. Database logging must be enabled to issue a roll forward database.

Before you begin

You should not be connected to the database that is to be rollforward recovered. The rollforward utility automatically establishes a connection to the specified database, and this connection is terminated at the completion of the rollforward operation.

About this task

Do not restore table spaces without canceling a rollforward operation that is in progress. Otherwise, you might have a table space set in which some table spaces are in rollforward in progress state, and some table spaces are in rollforward pending state. A rollforward operation that is in progress only operates on the tables spaces that are in rollforward in progress state.

The database can be local or remote.

The following restrictions apply to the rollforward utility:

- You can invoke only one rollforward operation at a time. If there are many table spaces to recover, you can specify all of them in the same operation.
- If you have renamed a table space following the most recent backup operation, ensure that you use the new name when rolling the table space forward. The previous table space name is not recognized.
- You cannot cancel a rollforward operation that is running. You can only cancel a rollforward operation that has completed, but for which the **STOP** parameter has not been specified, or a rollforward operation that has failed before completing.
- You cannot *continue* a table space rollforward operation to a point in time, specifying a time stamp that is less than the previous one. If a point in time is not specified, the previous one is used. You can issue a rollforward operation that ends at a specified point in time by just specifying STOP, but this is only allowed if the table spaces involved were all restored from the same offline backup image. In this case, no log processing is required. If you start another rollforward operation with a different table space list before the in-progress rollforward operation is either completed or cancelled, an error message (SQL4908) is returned. Invoke the **LIST TABLESPACES** command on all database partitions (or use the **MON_GET_TABLESPACE** table function) to determine which table spaces are currently being rolled forward (rollforward in progress state), and which table spaces are ready to be rolled forward (rollforward pending state). You have three options:
 - Finish the in-progress rollforward operation on all table spaces.
 - Finish the in-progress rollforward operation on a subset of table spaces. (This might not be possible if the rollforward operation is to continue to a specific point in time, which requires the participation of all database partitions.)
 - Cancel the in-progress rollforward operation.

- In a partitioned database environment, the rollforward utility must be invoked from the catalog partition of the database.
- Point in time rollforward of a table space was introduced in DB2 Version 9.1 clients. You should upgrade to Version 10.1 any clients in order to roll a table space forward to a point in time.
- You cannot roll forward logs from a previous release version.

Procedure

To invoke the rollforward utility, use the:

- **ROLLFORWARD DATABASE** command, or
- db2Rollforward application programming interface (API).
- Open the task assistant inIBM Data Studio for the **ROLLFORWARD DATABASE** command.

Example

The following is an example of the **ROLLFORWARD DATABASE** command issued through the CLP:

```
db2 rollforward db sample to end of logs and stop
```

Rollforward sessions - CLP examples

You can issue rollforward commands from the Command Line Prompt. Before issuing a rollforward command, you might find it helpful to review some sample sessions.

Example 1

The ROLLFORWARD DATABASE command permits specification of multiple operations at once, each being separated with the keyword AND. For example, to roll forward to the end of logs, and complete, the separate commands are:

```
db2 rollforward db sample to end of logs
db2 rollforward db sample complete
```

can be combined as follows:

```
db2 rollforward db sample to end of logs and complete
```

Although the two are equivalent, it is recommended that such operations be done in two steps. It is important to verify that the rollforward operation has progressed as expected before you stop it, so that you do not miss any logs.

If the rollforward command encounters an error, the rollforward operation will not complete. The error will be returned, and you will then be able to fix the error and reissue the command. If, however, you are unable to fix the error, you can force the rollforward to complete by issuing the following:

```
db2 rollforward db sample complete
```

This command brings the database online at the point in the logs before the failure.

Example 2

Roll the database forward to the end of the logs (two table spaces have been restored):

```
db2 rollforward db sample to end of logs  
db2 rollforward db sample to end of logs and stop
```

These two statements are equivalent. Neither AND STOP or AND COMPLETE is needed for table space rollforward recovery to the end of the logs. Table space names are not required. If not specified, all table spaces requiring rollforward recovery will be included. If only a subset of these table spaces is to be rolled forward, their names must be specified.

Example 3

After three table spaces have been restored, roll one forward to the end of the logs, and the other two to a point in time, both to be done online:

```
db2 rollforward db sample to end of logs tablespace(TBS1) online  
db2 rollforward db sample to 1998-04-03-14.21.56 and stop  
tablespace(TBS2, TBS3) online
```

Note that two rollforward operations cannot be run concurrently. The second command can only be invoked after the first rollforward operation completes successfully.

Example 4

After restoring the database, roll forward to a point in time, using OVERFLOW LOG PATH to specify the directory where the user exit saves archived logs:

```
db2 rollforward db sample to 1998-04-03-14.21.56 and stop  
overflow log path (/logs)
```

Example 5

In the following example, there is a database called sample. The database is backed up and the recovery logs are included in the backup image; the database is restored; and the database is rolled forward to the end of backup timestamp.

Back up the database, including the recovery logs in the backup image:

```
db2 backup db sample online include logs
```

Restore the database using that backup image:

```
db2 restore db sample
```

Roll forward the database to the end of backup timestamp:

```
db2 rollforward db sample to end of backup
```

Example 6 (partitioned database environments)

There are three database partitions: 0, 1, and 2. Table space TBS1 is defined on all database partitions, and table space TBS2 is defined on database partitions 0 and 2. After restoring the database on database partition 1, and TBS1 on database partitions 0 and 2, roll the database forward on database partition 1:

```
db2 rollforward db sample to end of logs and stop
```

This returns warning SQL1271 ("Database is recovered but one or more table spaces are offline on database partitions 0 and 2.").

```
db2 rollforward db sample to end of logs
```

This rolls TBS1 forward on database partitions 0 and 2. The clause TABLESPACE(TBS1) is optional in this case.

Example 7 (partitioned database environments)

In the following example, there is a partitioned database called sample. All the database partitions are backed up with a single system view backup; the database is restored on all database partitions; and the database is rolled forward to the end of backup timestamp.

Perform a single system view (SSV) backup:

```
db2 backup db sample on all nodes online include logs
```

Restore the database on all database partitions:

```
db2_all "db2 restore db sample taken at 1998-04-03-14.21.56"
```

Roll forward the database to the end of backup timestamp:

```
db2 rollforward db sample to end of backup on all nodes
```

Example 8 (partitioned database environments)

In the following example, there is a partitioned database called sample. All the database partitions are backed up with one command using db2_all; the database is restored on all database partitions; and the database is rolled forward to the end of backup timestamp.

Back up all the database partitions with one command using db2_all:

```
db2_all "db2 backup db sample include logs to //dir/"
```

Restore the database on all database partitions:

```
db2_all "db2 restore db sample from //dir/"
```

Roll forward the database to the end of backup timestamp:

```
db2 rollforward db sample to end of backup on all nodes
```

Example 9 (partitioned database environments)

After restoring table space TBS1 on database partitions 0 and 2 only, roll TBS1 forward on database partitions 0 and 2:

```
db2 rollforward db sample to end of logs
```

Database partition 1 is ignored.

```
db2 rollforward db sample to end of logs tablespace(TBS1)
```

This fails, because TBS1 is not ready for rollforward recovery on database partition 1. Reports SQL4906N.

```
db2 rollforward db sample to end of logs on  
dbpartitionnums (0, 2) tablespace(TBS1)
```

This completes successfully.

```
db2 rollforward db sample to 1998-04-03-14.21.56 and stop  
tablespace(TBS1)
```

This fails, because TBS1 is not ready for rollforward recovery on database partition 1; all pieces must be rolled forward together.

Note: With table space rollforward to a point in time, the dbpartitionnum clause is not accepted. The rollforward operation must take place on all the database partitions on which the table space resides.

After restoring TBS1 on database partition 1:

```
db2 rollforward db sample to 1998-04-03-14.21.56 and stop  
tablespace(TBS1)
```

This completes successfully.

Example 10 (partitioned database environments)

After restoring a table space on all database partitions, roll forward to PIT2, but do not specify AND STOP. The rollforward operation is still in progress. Cancel and roll forward to PIT1:

```
db2 rollforward db sample to pit2 tablespace(TBS1)  
db2 rollforward db sample cancel tablespace(TBS1)  
  
** restore TBS1 on all dbpartitionnums **  
  
db2 rollforward db sample to pit1 tablespace(TBS1)  
db2 rollforward db sample stop tablespace(TBS1)
```

Example 11 (partitioned database environments)

Rollforward recover a table space that resides on eight database partitions (3 to 10) listed in the db2nodes.cfg file:

```
db2 rollforward database dwtest to end of logs tablespace (tssprod)
```

This operation to the end of logs (not point in time) completes successfully. The database partitions on which the table space resides do not have to be specified. The utility defaults to the db2nodes.cfg file.

Example 12 (partitioned database environments)

Rollforward recover six small table spaces that reside on a single database partition database partition group (on database partition 6):

```
db2 rollforward database dwtest to end of logs on dbpartitionnum (6)  
tablespace(tsstore, tssbuyer, tsstime, tsswhse, tsslscat, tssvendor)
```

This operation to the end of logs (not point in time) completes successfully.

Example 13 (Partitioned tables - Rollforward to end of log on all data partitions)

A partitioned table is created using table spaces tbsp1, tbsp2, tbsp3 with an index in tbsp0. Later on, a user adds data partitions to the table in tbsp4, and attaches data partitions from the table in tbsp5. All table spaces can be rolled forward to END OF LOGS.

```
db2 rollforward db PBARDB to END OF LOGS and stop  
tablespace(tbsp0, tbsp1, tbsp2, tbsp3, tbsp4, tbsp5)
```

This completes successfully.

Example 14 (Partitioned tables - Rollforward to end of logs on one table space)

A partitioned table is created initially using table spaces tbsp1, tbsp2, tbsp3 with an index in tbsp0. Later on, a user adds data partitions to the table in tbsp4, and attaches data partitions from the table in tbsp5. Table space tbsp4 becomes corrupt and requires a restore and rollforward to end of logs.

```
db2 rollforward db PBARDB to END OF LOGS and stop tablespace(tbsp4)
```

This completes successfully.

Example 15 (Partitioned tables - Rollforward to PIT of all data partitions including those added, attached, detached or with indexes)

A partitioned table is created using table spaces tbsp1, tbsp2, tbsp3 with an index in tbsp0. Later on, a user adds data partitions to the table in tbsp4, attaches data partitions from the table in tbsp5, and detaches data partitions from tbsp1. The user performs a rollforward to PIT with all the table spaces used by the partitioned table including those table spaces specified in the INDEX IN clause.

```
db2 rollforward db PBARDB to 2005-08-05-05.58.53 and stop  
tablespace(tbsp0, tbsp1, tbsp2, tbsp3, tbsp4, tbsp5)
```

This completes successfully.

Example 16 (Partitioned tables - Rollforward to PIT on a subset of the table spaces)

A partitioned table is created using three table spaces (tbsp1, tbsp2, tbsp3). Later, the user detaches all data partitions from tbsp3. The rollforward to PIT is only permitted on tbsp1 and tbsp2.

```
db2 rollforward db PBARDB to 2005-08-05-06.02.42 and stop  
tablespace( tbsp1, tbsp2)
```

This completes successfully.

Rolling forward changes in a table space

If the database is enabled for rollforward recovery, you have the option of backing up, restoring, and rolling forward table spaces instead of the entire database.

You can roll forward changes to a table space independently of other table spaces in your database, or you can roll forward changes to all table spaces at the same time.

Implementing a recovery strategy for individual table spaces can save time because it takes less time to recover a portion of the database than it does to recover the entire database. For example, if a disk is bad, and it contains only one table space, you can restore that table space and roll it forward without having to recover the entire database, and without impacting user access to the rest of the database, unless the damaged table space contains the system catalog tables; in this situation, you cannot connect to the database. (You can restore the system catalog table space independently if a table space-level backup image containing the system catalog table space is available.) Table space-level backups also allow you to back up critical parts of the database more frequently than other parts, and requires less time than backing up the entire database.

After a table space is restored, it is always in rollforward pending state. To make the table space usable, you must perform rollforward recovery on it. In most cases, you have the option of rolling forward to the end of the logs, or rolling forward to a point in time. You cannot, however, roll table spaces containing system catalog tables forward to a point in time. These table spaces must be rolled forward to the end of the logs to ensure that all table spaces in the database remain consistent.

Ensure that the **DB2_COLLECT_TS_REC_INFO** registry variable is set to ON (the default) if you want to skip the log files known not to contain any log records affecting the table space. This registry variable must be set before the log files are created and used so that the information required for skipping log files is collected. If **DB2_COLLECT_TS_REC_INFO** is set to OFF, all log files are processed even if they do not contain log records that affect that table space when that table space is rolled forward.

The table space change history file (DB2TSCHG.HIS), which is located in the database directory, tracks which logs to process for each table space. You can view the contents of this file with the **db2logsForRfwd** utility, and delete entries from it with the **PRUNE HISTORY** command. During a database restore operation, the DB2TSCHG.HIS file is restored from the backup image and then brought up to date during the database rollforward operation. If no information is available for a log file, it is treated as though it is required for the recovery of every table space.

Because information for each log file is flushed to disk after the log becomes inactive, this information can be lost as a result of a crash. To prevent this loss from occurring, if a recovery operation begins in the middle of a log file, the entire log is treated as though it contains modifications to every table space in the system. All active logs are processed and the information for them is rebuilt. If information for older or archived log files is lost in a crash situation and no information for them exists in the data file, they are treated as though they contain modifications for every table space during the table space recovery operation.

Before you roll a table space forward, use the MON_GET_TABLESPACE table function to determine the *minimum recovery time*, which is the earliest point in time to which the table space can be rolled forward. The minimum recovery time is updated when data definition language (DDL) statements are run against the table space, or against tables in the table space. The table space must be rolled forward to at least the minimum recovery time so that it becomes synchronized with the information in the system catalog tables. If you are recovering more than one table space, the table spaces must be rolled forward to at least the highest minimum recovery time of all the table spaces that are being recovered. You cannot roll forward a table space to a time that is earlier than the backup timestamp. In a partitioned database environment, you must roll forward the table spaces to at least the highest minimum recovery time of all the table spaces on all database partitions.

If you are rolling table spaces forward to a point in time, and a table is contained in multiple table spaces, all of these table spaces must be rolled forward simultaneously. If, for example, the table data is contained in one table space, and the index for the table is contained in another table space, you must roll both table spaces forward simultaneously to the same point in time.

If the data and the long objects in a table are in separate table spaces, and the long object data was reorganized, the table spaces for both the data and the long objects must be restored and rolled forward together. Take a backup of the affected table spaces after the table is reorganized.

If you want to roll forward a table space to a point in time, and a table in the table space is either:

- an underlying table for a materialized query or staging table that is in another table space
- a materialized query or staging table for a table in another table space

then roll both table spaces forward to the same point in time. If you do not, the materialized query or staging table is placed in set integrity pending state at the end of the rollforward operation. The materialized query table needs to be fully refreshed, and the staging table is marked as incomplete.

If you want to roll forward a table space to a point in time, and a table in the table space participates in a referential integrity relationship with another table that is contained in another table space, roll forward both table spaces simultaneously to the same point in time. If you do not roll forward both table spaces, the child table in the referential integrity relationship is placed in set integrity pending state at the end of the rollforward operation. When the child table is later checked for constraint violations, a check on the entire table is required. If any of the following tables exist, they are also placed in set integrity pending state with the child table:

- any descendant materialized query tables for the child table
- any descendant staging tables for the child table
- any descendant foreign key tables of the child table

These tables require full integrity processing to bring them out of the set integrity pending state. If you roll forward both table spaces simultaneously, the constraint remains active at the end of the point-in-time rollforward operation.

Ensure that a point-in-time table space rollforward operation does not cause a transaction to be rolled back in some table spaces, and committed in others. This inconsistency can happen in the following cases:

- A point-in-time rollforward operation is performed on a subset of the table spaces that were updated by a transaction, and that point in time precedes the time at which the transaction was committed.
- Any table that is contained in the table space being rolled forward to a point in time has an associated trigger, or is updated by a trigger that affects table spaces other than the one that is being rolled forward.

The solution is to find a suitable point in time that prevents this from happening.

You can issue the **QUIESCE TABLESPACES FOR TABLE** command to create a transaction-consistent point in time for rolling table spaces forward. The quiesce request (in share, intent to update, or exclusive mode) waits (through locking) for all running transactions against those table spaces to complete, and blocks new requests. When the quiesce request is granted, the table spaces are in a consistent state. To determine a suitable time to stop the rollforward operation, you can look in the recovery history file to find quiesce points, and check whether they occur after the minimum recovery time.

After a table space point-in-time rollforward operation completes, the table space is put in backup pending state. You must take a backup of the table space because all updates made to it between the point in time to which you rolled forward and the current time were removed. You can no longer roll forward the table space to the current time from a previous database- or table space-level backup image. The following example shows why the table space-level backup image is required, and how it is used. (To make the table space available, you can either back up the entire database, the table space that is in backup pending state, or a set of table

spaces that includes the table space that is in backup pending state.)

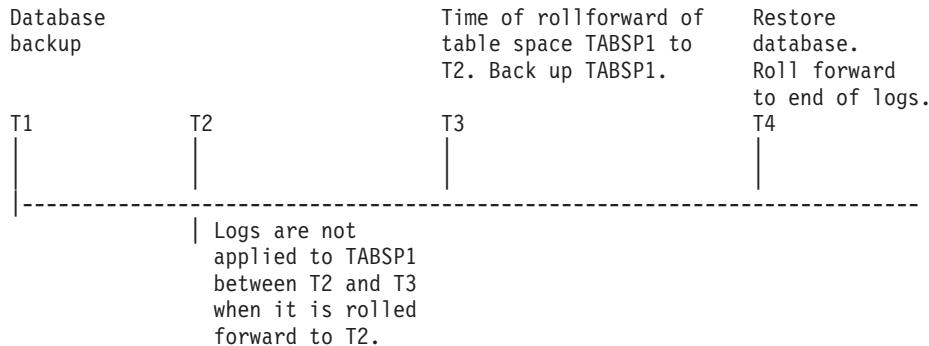


Figure 63. Table space backup requirement

In the preceding example, the database is backed up at time T1. Then, at time T3, table space TABSP1 is rolled forward to a specific point in time (T2). The table space is backed up after time T3. Because the table space is in backup pending state, this backup operation is mandatory. The timestamp of the table space backup image is after time T3, but the table space is at time T2. Log records from between T2 and T3 are not applied to TABSP1. At time T4, the database is restored, using the backup image that was created at T1, and rolled forward to the end of the logs. Table space TABSP1 is put in restore pending state at time T3, because the database manager assumes that operations were performed on TABSP1 between T3 and T4 without the log changes between T2 and T3 being applied to the table space. If these log changes were in fact applied as part of the rollforward operation against the database, this assumption would be incorrect. The table space-level backup that must be taken after the table space is rolled forward to a point in time allows you to roll forward that table space past a previous point-in-time rollforward operation (T3 in the example).

Assuming that you want to recover table space TABSP1 to T4, you would restore the table space from a backup image that was taken after T3 (either the required backup, or a later one), then roll forward TABSP1 to the end of the logs.

In the preceding example, the most efficient way of restoring the database to time T4 would be to perform the required steps in the following order:

1. Restore the database.
2. Restore the table space.
3. Roll forward the database.

Because you restore the table space before you roll forward the database, resources are not used to apply log records to the table space when the database is rolled forward.

If you cannot find the TABSP1 backup image that follows time T3, or you want to restore TABSP1 to T3 (or earlier), you can do one of the following actions:

- Roll forward the table space to T3. You do not need to restore the table space again because it was restored from the database backup image.
- Restore the table space again by restoring the database backup that was taken at time T1, and then roll forward the table space to a time that precedes time T3.
- Drop the table space.

In a partitioned database environment:

- You must simultaneously roll forward all parts of a table space to the same point in time at the same time. This ensures that the table space is consistent across database partitions.
- If some database partitions are in rollforward pending state, and on other database partitions, some table spaces are in rollforward pending state (but the database partitions are not), you must first roll forward the database partitions, and then roll forward the table spaces.
- If you intend to roll forward a table space to the end of the logs, you do not have to restore it at each database partition; you must restore it at the database partitions that require recovery. If you intend to roll forward a table space to a point in time, however, you must restore it at each database partition.

In a database with partitioned tables:

- If you are rolling a table space that contains any piece of a partitioned table forward to a point in time, you must also roll forward all of the other table spaces in which that table resides to the same point in time. However, rolling forward a single table space containing a piece of a partitioned table to the end of logs is allowed. If a partitioned table has any attached, detached, or dropped data partitions, then a point-in-time rollforward operation must also include all table spaces for these data partitions. In order to determine if a partitioned table has any attached, detached, or dropped data partitions, query the SYSCAT.DATAPARTITIONS catalog view.

Database rollforward operations in a DB2 pureScale environment

In a DB2 pureScale environment, each member has its own log stream; however, log streams from all members are required for successful execution of the **ROLLFORWARD DATABASE** command.

During a database rollforward operation, log records from all of the log streams are merged and replayed to make the database consistent. The point in time that you specify on the **ROLLFORWARD DATABASE** command is relative to the merged log stream. To restore the database to a consistent state, the specified time must be later than the *minimum recovery time* (MRT). The MRT is the earliest time during a rollforward operation when objects that are listed in the database catalog match the objects that physically exist on disk. For example, if you are restoring from an image that was created during an online backup operation, the specified point in time for the rollforward operation must be later than the time at which the online backup operation completed. This will ensure database consistency.

The specified point in time for the subsequent database rollforward operation must be greater than or equal to the MRT in the merged log stream; otherwise, the rollforward operation fails (SQL1276N), and the timestamp of the MRT is returned with the error message. Alternatively, you can use the END OF BACKUP option to automatically roll forward to the MRT.

It is recommended that the member clocks be synchronized; however, it might not be possible to synchronize them at all times. This can result in log records having the same time stamp, and merged log streams with log records that appear to be out of time stamp order. In a DB2 pureScale environment, a point-in-time database rollforward operation stops when it encounters the first log record whose time stamp is greater than the specified time stamp from any log stream, and it has processed the log record that corresponds to the MRT for the database.

An incomplete or interrupted rollforward operation leaves the database in rollforward pending state. In this case, issue another **ROLLFORWARD DATABASE** command. In a DB2 pureScale environment, subsequent **ROLLFORWARD DATABASE** commands can be run on the same or on a different member.

In a DB2 pureScale environment, if you want to perform a database restore operation into a new database using an online database backup image, the correct approach depends on whether all of the log files are available, or only log files from the backup image are available.

- If pre-existing log files or archived log files can be accessed, the following rollforward operation is appropriate:

```
db2 rollforward db dbname to end of logs and stop
```

Note: Before taking a backup, you need to ensure that the log archiving path is set to a shared directory so that all the members are able to access the logs for subsequent rollforward operations. If the archive path is not accessible from the member on which the rollforward is being executed, SQL1273N is returned. The following command is an example of how to set the log path to the shared directory:

```
db2 update db cfg using logarchmeth1  
DISK:/db2fs/gpfs1/svtdbm5/svtdbm5/ArchiveLOGS
```

(where *gpfs1* is the shared directory for the members and *ArchiveLOGS* is the actual directory that archives the logs.)

- If the only log files that can be accessed come from the backup image, the following rollforward operation is appropriate:

```
db2 rollforward db dbname to end of backup and stop
```

This command replays all required log records to achieve the consistent database state that was in effect when the backup operation ended. You can also use this command if pre-existing log files or archived log files can be accessed, but it will stop at the point at which the backup operation ended; it will not use any extra logs that were generated after the backup operation ended.

A **ROLLFORWARD DATABASE** command specifying the END OF LOGS option in this case would return SQL1273N. A subsequent **ROLLFORWARD DATABASE** command with the STOP option is successful, and the database will be available, if the missing log files are not needed. However, if the missing log files are needed (and it is not safe to stop), the rollforward operation will again return SQL1273N.

Example

Suppose that there are two members, M1 and M2. M2's clock is ahead of M1's clock by five seconds. M2's log stream contains the following log records:

```
A1 at 2010-04-03-14.21.56  
A2 at 2010-04-03-14.21.56  
B at 2010-04-03-14.21.58  
C at 2010-04-03-14.22.01
```

M1's log stream contains the following log records:

```
D at 2010-04-03-14.21.55  
E at 2010-04-03-14.21.56  
F at 2010-04-03-14.21.57
```

The minimum recovery time (MRT) for the database on M2 is at time 2010-04-03-14.21.55. Because M1's clock is five seconds slow, log records D, E, and F appear later in the merged log stream:

```
MRT: 2010-04-03-14.21.55 (M2)
A1: 2010-04-03-14.21.56 (M2)
A2: 2010-04-03-14.21.56 (M2)
B: 2010-04-03-14.21.58 (M2)
D: 2010-04-03-14.21.55 (M1) --> corresponding time on M2 is 14.22.00
C: 2010-04-03-14.22.01 (M2)
E: 2010-04-03-14.21.56 (M1) --> corresponding time on M2 is 14.22.01
F: 2010-04-03-14.21.57 (M1) --> corresponding time on M2 is 14.22.02
```

The alphabetic characters (A1, A2, B, and so on) represent the order in which the corresponding log records were actually written at run time (across members). Note that log records A1 and A2 from member M2 have the same time stamp; this can happen when the DB2 data server tries to optimize performance by including the commit log record from multiple transactions when data is written from the log buffer to a log file.

The following command returns SQL1276N (Database "test" cannot be brought out of rollforward pending state until rollforward has passed a point in time greater than or equal to "2010-04-03-14.21.55"):

```
db2 rollforward db test to 2010-04-03-14.21.54
```

But the following command rolls forward the database up to and including log record A2:

```
db2 rollforward db test to 2010-04-03-14.21.56
```

Because log records A1 and A2 both have a time stamp that is less than or equal to the time that was specified in the command, both are replayed. Log record B, whose time stamp (2010-04-03-14.21.58) is greater than the specified value (2010-04-03-14.21.56), stops the rollforward operation and is not replayed. Log record D is not replayed either, even though its time stamp is less than the specified value, because log record B's higher value (2010-04-03-14.21.58) was encountered first. The following command rolls forward the database up to and including log record D:

```
db2 rollforward db test to 2010-04-03-14.21.58
```

Log record C, whose time stamp (2010-04-03-14.22.01) is greater than the specified value (2010-04-03-14.21.58), stops the rollforward operation and is not replayed. Log record E is not replayed either, even though its time stamp is less than the specified value.

Monitoring a rollforward operation

You can use the **db2pd** or the **LIST UTILITIES** command to monitor the progress of rollforward operations on a database.

Procedure

- Issue the **LIST UTILITIES** command and specify the **SHOW DETAIL** parameter
LIST UTILITIES SHOW DETAIL
- Issue the **db2pd** command and specify the **-recovery** parameter:
db2pd -recovery

Results

For rollforward recovery, there are two phases of progress monitoring: FORWARD and BACKWARD. During the FORWARD phase, log files are read and the log records are applied to the database. For rollforward recovery, when this phase begins UNKNOWN is specified for the total work estimate. The amount of work processed in bytes is updated as the process continues.

During the BACKWARD phase, any uncommitted changes applied during the FORWARD phase are rolled back. An estimate for the amount of log data to be processed, in bytes, is provided. The amount of work processed, in bytes, is updated as the process continues.

Example

The following is an example of the output for monitoring the performance of a rollforward operation using the **db2pd** command:

```
Recovery:  
Recovery Status      0x00000401  
Current Log          S0000005.LOG  
Current LSN          0000001F07BC  
Current LSO          000002551BEA  
Job Type             ROLLFORWARD RECOVERY  
Job ID               7  
Job Start Time       (1107380474) Wed Feb 2 16:41:14 2005  
Job Description      Database Rollforward Recovery  
Invoker Type         User  
Total Phases         2  
Current Phase        1  
  
Progress:  
Address      PhaseNum Description StartTime           CompletedWork TotalWork  
0x0000000200667160 1       Forward    Wed Feb 2 16:41:14 2005 2268098 bytes Unknown  
0x0000000200667258 2       Backward   NotStarted          0 bytes      Unknown
```

The following is an example of the output for monitoring the performance of a database rollforward operation using the **LIST UTILITIES** command with the SHOW DETAIL option:

```
ID                  = 7  
Type                = ROLLFORWARD RECOVERY  
Database Name       = TESTDB  
Member Number       = 0  
Description         = Database Rollforward Recovery  
Start Time          = 01/11/2012 16:56:53.770404  
State               = Executing  
Invocation Type     = User  
  
Progress Monitoring:  
Estimated Percentage Complete = 50  
Phase Number        = 1  
Description          = Forward  
Total Work           = 928236 bytes  
Completed Work       = 928236 bytes  
Start Time          = 01/11/2012 16:56:53.770492  
  
Phase Number [Current] = 2  
Description          = Backward  
Total Work           = 928236 bytes  
Completed Work       = 0 bytes  
Start Time          = 01/11/2012 16:56:56.886036
```

The following is an example of the output for monitoring the performance of a table space rollforward operation using the **LIST UTILITIES** command with the SHOW DETAIL option:

ID = 17
Type = ROLLFORWARD RECOVERY
Database Name = TESTDB
Member Number = 0
Description = Offline Tablespace Rollforward Recovery: 3
Start Time = 01/11/2012 17:04:27.269171
State = Executing
Invocation Type = User
Progress Monitoring:
 Estimated Percentage Complete = 63
 Phase Number = 1
 Description = Forward
 Total Work = 142
 Completed Work = 90
 Start Time = 01/11/2012 17:04:27.269283

 Phase Number [Current] = 2
 Description = Backward
 Total Work = 0
 Completed Work = 0
 Start Time = Not Started

Chapter 16. DB2 Workload Manager (WLM)

DB2 Workload Manager (WLM) introduces significant capabilities for controlling and monitoring executing-workloads within DB2. This technology is directly incorporated into the DB2 engine infrastructure to allow handling higher volumes with minimal overhead. It is also enabled for tighter integration with external workload management products such as AIX WLM.

Workload management concepts

A good workload management system helps to efficiently meet goals in the environment where work occurs. You can see examples of the need for a good workload management system all around you.

For example, look at a grocery store. Different activities must be considered: serving customers, stocking shelves, maintaining inventories, and so on. And some simple goals must be set. Store owners want to maximize both the number of customers who move through the store, and the amount that customers purchase, achieving both goals in a way that customers leave both satisfied and wanting to come back. Store owners must also ensure that they have sufficient stock for their customers to buy (but not too much stock, because waste becomes an issue). Store owners also track what their customers purchase, and use this information to create advertisements that are designed to induce their customers to return. Monitoring mechanisms track inventory and send notifications when stocks run low. Security devices are in place to detect shoplifting. Special fast checkout lanes are created so that shoppers who only want to purchase a few items can do so without having to wait behind other customers who are purchasing many items. If all of these goals are met and all of these operational procedures work well, customers are satisfied, and are likely to return rather than to go to another store. These goals and operational procedures are all aspects of workload management.

In a data server environment, you can see even more of a need for effective management of work, especially now that data servers are being stressed like never before. Cash registers generate thousands of data inserts, reports are constantly being generated to determine whether sales targets are being met, batch applications run to load collected data, and administration tasks such as backups and reorganizations run to protect the data and make the server run optimally. All these operations are using the same database system and competing for the same resources.

To ensure the best chance of meeting goals for running a data server, an efficient workload management system is critical.

Phases of workload management

The phases of workload management are identification of the work entering the data server, management of the work when it is running, and monitoring to ensure that the data server is being used efficiently.

A number of aspects must be considered for successful workload management with DB2 workload manager, starting with understanding your goals. In the grocery store example described in “Workload management concepts,” goals might

include maximizing customer spending, minimizing shoplifting, and ensuring that customers leave the store satisfied so that they will return again.

In a data server environment, you must also define goals. Sometimes the goals are clear, especially when they originate from service level agreement (SLA) objectives. For example, queries from a particular application can consume no more than 10% of the total processor resource. Goals can also be tied to a particular time of day. For example, an overnight batch utility might have to complete loading data by 8 a.m. so that the daily sales reports are on time. In other situations, the goals can be difficult to quantify. A goal might be to keep the database users satisfied and to prevent aberrant database activity from hampering their day-to-day work. Whether the goals are quantifiable or not, understanding them is critical when considering the following stages of workload management:

Identification

If you want to achieve a goal for some kind of work, you first must be able to identify details about the work. In the grocery store, you can identify shopper information through credit cards and debit cards, or an unpaid-for item through an active security tag on the item. For the data server, you need to decide how you want to identify the work that enters the system. You can use the name of the application that submits the work, the authorization ID that submits the work, or a combination of elements that provide some form of identification.

Management

The management phase includes mechanisms for making steady progress towards your goal, and actions to take if a goal is not being met. An example of a mechanism is managing price checks in fast checkout lanes. Fast checkout lanes should result in faster throughput and satisfied customers, but if a carton of milk has the wrong price and a price check is required, the fast checkout lane could slow down. The problem is managed by performing a fast price check, possibly opening up another checkout lane, and trying to fix the pricing problem so that it does not occur again. On the data server, you might find that overall performance is suffering when a few poorly written SQL statements are running, a surge in volume occurs during peak times, or there is too much competition between different applications for the same resources. The management phase includes mechanisms for assigning resources to achieve your goals and actions to take if a goal is not being met. These workload management mechanisms, which indirectly and directly control CPU resources, include the following:

1. Concurrency thresholds, applied with a work action set defined on the workload, to control the concurrency of incoming work
2. The ability to move work from one service class to another, currently only effective when workload management dispatcher is being used on those service classes
3. Workload management dispatcher to specifically allocate CPU resources for workloads assigned to service classes for more granular control of CPU resources when the first two workload management mechanisms are not adequate

Monitoring

Monitoring is important for a couple of reasons. First, to determine whether you are achieving a goal, you must have a mechanism to track progress toward that goal. Also, monitoring helps to identify the problems that might be preventing you from achieving your goal. In a store, the store manager can watch the flow of customers, automatically be alerted to

problems such as shoplifting or dangerously low inventory of a particular sale item, or perform analysis on historical purchase patterns to determine optimal product placement in the store. For a data server, there are often explicit goals for response times of database activities and it is important to have a way to measure this metric, and watch for trends.

The following figure represents the workload management phases:

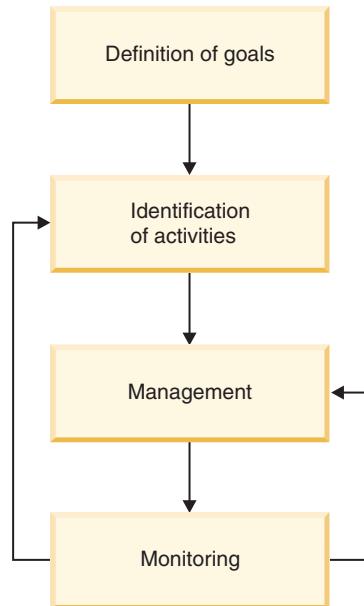


Figure 64. Phases of workload management

Frequently asked questions about DB2 workload management

This FAQ provides you with answers to common questions about DB2 workload management.

General

- On which DB2 platforms can I use DB2 workload management?
- I am not on AIX. Does this mean I do not have any control over processor resource or I/O activity?
- Now that Query Patroller is discontinued and DB2 Governor are deprecated, how do I migrate to DB2 workload manager?
- Is there a way for WebSphere® Application Server to pass the client information fields used by the DB2 workload?
- Why is my work not assigned to the correct workload?
- Why does DB2 workload manager affect REORGCHK, IMPORT, EXPORT and other CLP commands?
- Is there a way to change the service class to which an activity is assigned while it is executing?
- Much of my batch work is done using CLP scripts under the same ID, how can I go about uniquely identifying these so I can manage them differently from each other?
- When should I use the COLLECT AGGREGATE ACTIVITY DATA clause versus the COLLECT ACTIVITY DATA clause?

- How does DB2 workload management work with the new AIX WPAR feature?
- What is the relationship between the DB2_OPT_MAX_TEMP_SIZE registry variable and a DB2 threshold based on SQLTEMPSPACE?

Licensing

- What are the licensing requirements for DB2 workload manager?

Monitoring

- What information do you get from the different event monitors that are associated with workload management?

OS workload management (AIX WLM and Linux WLM)

- Why would I ever want to use AIX WLM or Linux WLM?
- I am not on AIX. Does this mean I do not have any control over processor resources or I/O activity?
- Can I use AIX WLM to manage I/O activity?
- Can I use AIX WLM to manage memory use?
- How does DB2 WLM work with the new AIX WPAR feature?

Platforms

- On which DB2 platforms can I use DB2 workload manager?
- I am not on AIX. Does this mean I do not have any control over processor resource or I/O activity?
- How does DB2 workload management work with the new AIX WPAR feature?
- Why would I ever want to use AIX WLM or Linux WLM?

Query Patroller and Governor

- How does this new functionality affect Query Patroller and DB2 Governor?
- Now that Query Patroller and DB2 Governor are deprecated, how do I migrate to DB2 workload manager?

Thresholds

- Can I create multiple CONCURRENTDBCOORDACTIVITIES concurrency thresholds for the same set of work?
- How do I determine which activities are queued by a workload management threshold and the order of the activities in the queue?

Workload management dispatcher

- Do I need to use workload management dispatcher?
- What changes in behavior might I see when I turn ON the workload management dispatcher?
- With the introduction of the workload management dispatcher, are concurrency thresholds such as CONCURRENTDBCOORDACTIVITIES no longer needed or useful?

On which DB2 platforms can I use DB2 workload management?

DB2 workload management is available on all platforms supported by DB2 9.5 for Linux, UNIX, and Windows or later. The optional tight integration-offered between DB2 workload management at the service class level and operating system workload management capabilities-is available on AIX platforms and any Linux platform based on the 2.6.26 kernel or higher.

Do I need to use workload management dispatcher?

Most workload management configurations begin with concurrency thresholds, which affect the consumption of all resources by controlling how much work can begin executing at any one time. In some situations, however, a concurrency threshold is not able to effectively limit the total amount of processing resource that is consumed, and high priority work is affected; for example, a scenario in which complex work is restricted to one running query that nevertheless consumes enough resource to disrupt higher priority work. In such cases, the workload management dispatcher is used to explicitly control CPU consumption and protect the higher priority work.

You can use the workload management dispatcher for any or all of the following situations:

- You want to manage the share of CPU resources among multiple users or applications and you are using an operating system that does not have an operating system (OS) workload manager that integrates with DB2 workload management through the `outbound_correlator` field on each service class.
- You want to manage the share of CPU resources among multiple users or applications and you do not have root privilege on the operating system.
- You want to manage the share of CPU resources among multiple users or applications in a multiple member environment across multiple systems and managing this through the OS WLM on each system requires too much administration.
- You want to manage the share of CPU resources among multiple users or applications using hard shares to limit certain service classes, even when the CPU is under-utilized, and this is not available in your OS WLM or does not produce the desired result.

How does this new functionality affect Query Patroller and DB2 Governor?

The DB2 workload manager introduces an independent approach to workload management and does not rely on or interact with Query Patroller or DB2 Governor in any way. Query Patroller has been discontinued starting with the Version 10.1 release. DB2 Governor was deprecated in the DB2 Version 9.7 release and, although still functional, it is no longer central to the DB2 workload management strategy. No further investment is planned for DB2 Governor in future releases.

When DB2 9.5 or later is first installed, the default user service class is automatically defined and all incoming work is sent to it for execution. Although DB2 Governor can watch agents in any service class, it is permitted to adjust the agent priority only for agents in the default user service class.

I am not on AIX. Does this mean I do not have any control over processor resources or I/O activity?

Users on all platforms have the same ability to control processor resources and I/O activity between service classes using SQL, such as the CREATE and ALTER SERVICE CLASS statements, for example.

To control CPU usage when the workload management dispatcher is enabled, use the CPU limit attribute of the DB2 service class to limit the amount of CPU resources a service class can consume. If the workload management dispatcher

CPU shares (**wlm_disp_cpu_shares**) database manager configuration parameter is also enabled, you can use the CPU shares attribute of the DB2 service class to specify the share of CPU resources that a service class can consume relative to the CPU consumption of other service classes. On AIX and some Linux platforms, you can supplement (or replace) these approaches by taking advantage of the workload management capabilities that are offered by those operating systems to control CPU consumption.

For I/O activity, users on all platforms can set the buffer pool or prefetcher priority attribute of a DB2 service class to a value of high, medium, or low. All service classes run with a medium priority by default.

Can I use AIX or Linux WLM or the DB2 workload management dispatcher to manage I/O activity?

Currently, neither AIX WLM nor Linux WLM support direct I/O activity controls at the thread level. However, it is possible to indirectly control I/O activity by means of concurrency thresholds, or to use the DB2 workload management dispatcher, AIX WLM, or Linux WLM to manipulate CPU resources. The more CPU resource that is available to an executing thread, the less frequently that thread will request I/O resources.

You can influence buffer pool behavior by using the BUFFERPOOL PRIORITY attribute of any DB2 service class. You can also control DB2 prefetcher I/O activity by using the PREFETCH PRIORITY attribute of any DB2 service class.

Can I use AIX or Linux WLM to manage memory use?

DB2 data server uses primarily shared memory, which is accessed by more than one agent from different service classes. For this reason, it is not possible to divide memory allocation between different service classes using either AIX or Linux WLM.

Memory (such as sortheap) that is consumed during the execution of an SQL statement can be indirectly influenced through the use of concurrency thresholds, because consumption does not begin until the statement is allowed to execute. However, unlike I/O activity, restricting CPU consumption does not affect the amount of memory that is consumed. In fact, restricting CPU consumption can exacerbate the memory situation, because queries will be running more slowly and holding onto their allocated memory longer.

Is there a way for WebSphere Application Server to pass the client information fields used by the DB2 workload?

WebSphere Application Server Version 6.0 and Version 6.1 can set or pass in the CLIENT INFO fields to DB2 data server, either explicitly by your applications (see: Passing client information to a database) or implicitly by having WebSphere Application Server do it for you (see: Implicitly set client information).

Can I create multiple CONCURRENTDBCOORDACTIVITIES concurrency thresholds for the same set of work?

You can create one or more CONCURRENTDBCOORDACTIVITIES concurrency thresholds that apply to the same set of activities by defining them at the level of the database, the service class in which the work executes, or within a work action set applied at the database or workload level. Be aware that each new concurrency

threshold that applies to an activity implies additional overhead to enforce that concurrency threshold. Verify that you really need more than one concurrency threshold level.

Why is my work not assigned to the correct workload?

There are a number of reasons why a connection may not be mapped to the desired workload. The most common ones are the failure to grant USAGE privilege on the workload, incorrect spelling of the case sensitive connection attributes, or the existence of a matching workload definition that is positioned earlier in the evaluation order.

Before a connection can be assigned to a workload, the connection attributes must match those of the workload definition, and the session authorization ID must have USAGE privilege on the workload. A common omission is to create the workload but not to grant USAGE privilege on the workload to users (See “GRANT (Workload Privileges) statement” in *SQL Reference*). Only users with ACCESSCTRL, SECADM, or WLMADM authority can grant workload usage privilege to other users. Users with ACCESSCTRL, DATAACCESS, DBADM, SECADM, or WLMADM authority have implicit usage privilege on all workloads.

Connection attributes for workloads are case sensitive. For example: If the system user ID is uppercased, then the SYSTEM_USER connection attribute you specify must be in uppercase as well.

To establish why a connection is not being mapped to the expected workload, you should gather some information. Which workload is the work being mapped to? Is that workload before or after the one that you thought would be used when you look at the workload definitions in the order of evaluation? (Hint: try selecting the workload definitions ordered in ascending order by the value of the EVALUATIONORDER column in SYSCAT.WORKLOADS).

If you do not know what the connection attributes are for the target connection, you can find out the values for the connection in a number of different ways:

- Issue a query against the system using the WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES table function while the connection is active
- Open a cursor on a connection and use the WLM_CAPTURE_ACTIVITY_IN_PROGRESS stored procedure against that cursor to have the activity information captured to the activities event monitor (Hint: do not forget to create and activate the activities information event monitor)
- Turn on the collection of detailed activity information for the workload being used by the connection, issue one statement in order to capture the activity information, and then turn off the collection.

Why does DB2 workload manager affect REORGCHK, IMPORT, EXPORT and other CLP commands?

These CLP commands are affected by DB2 workload management thresholds, because the database engine cannot distinguish system requests originating with these utilities from other requests directly initiated by users within the CLP interactive front-end.

Is there a way to change the service class to which an activity is assigned while it is executing?

Yes, you can change the service subclass an activity is executing in to another service subclass within the same parent service superclass by defining a CPUTIMEINSC, DATATAGINSC, or SQLROWSINSC threshold with the REMAP ACTIVITY action on the original service subclass. Initially, DB2 workload management maps an activity to a service class based on the relevant workload definition for the connection, modifies it as required if a work action set exists on that service class, and then sets up the DB2 agent to execute in the assigned service class. When an activity violates a threshold that has a REMAP ACTIVITY action defined, the agent remaps itself to the specified target service subclass (under the same superclass) once the threshold violation has been detected and the activity continues executing in the new service subclass.

Much of my batch work is done using CLP scripts under the same ID, how can I go about uniquely identifying these so I can manage them differently?

You have a couple of options:

An enhancement has been added to CLP so that the client application name is automatically set to the CLP script filename, with a **CLP** prefix preceding it (the value of this field at the server can be seen in the CURRENT CLIENT_APPLNAME special register). For example, if the CLP script filename is **batch.db2**, the CURRENT CLIENT_APPLNAME special register value is set to **CLP batch.db2** by CLP when that script is run. With this feature, it is possible for different CLP scripts to be associated with different workloads based on the client application name.

For example, to create a workload for CLP file **batch1.db2**, you can issue the following DDL statement:

```
CREATE WORKLOAD batch1 CURRENT CLIENT_APPLNAME ('CLP batch1.db2')
SERVICE CLASS class1
```

To create a workload for CLP file **batch2.db2**, you can issue the following DDL statement:

```
CREATE WORKLOAD batch2 CURRENT CLIENT_APPLNAME ('CLP batch2.db2')
SERVICE CLASS class2
```

Since these two batch files are associated with different workloads, they can be assigned to different service classes and managed differently.

Another option is the new stored procedure WLM_SET_CLIENT_INFO, which permits you to set the values of any of the client information fields at the server using a simple CALL SQL statement. By inserting a CALL statement into any of your existing CLP scripts, you can uniquely identify them using these fields and map them to different workload definitions.

For more information, see “WLM_SET_CLIENT_INFO procedure” in *Administrative Routines and Views*.

When should I use the COLLECT AGGREGATE ACTIVITY DATA clause versus the COLLECT ACTIVITY DATA clause?

The answer depends on why the monitoring is desired and what is to be done with the information.

Aggregate activity information spans the entire set of work that has executed within the scope covered by the clause, and it captures summary characteristics of this set; it does not capture specific details about individual activities. The COLLECT AGGREGATE ACTIVITY DATA clause can be specified on DB2 workloads, DB2 service classes, and DB2 work action sets. For normal operational monitoring, use the COLLECT AGGREGATE ACTIVITY DATA clause, because it is very lightweight, it can be gathered automatically by the statistics event monitor for a historical record, and it provides important information on overall response time patterns. If further insight is required on a specific type of work, use the COUNT ACTIVITY or COLLECT AGGREGATE ACTIVITY DATA actions within a DB2 work action set to gather more granular information (with minimal overhead) about different types of work executing in a workload, service class, or database.

In contrast, activity information contains detailed information about each and every activity that executes within the scope covered by the COLLECT ACTIVITY DATA clause. This clause can be specified on DB2 workloads, DB2 service classes, DB2 work action sets, and DB2 thresholds. It permits further in-depth analysis of the individual activities that are captured, in order to understand the flow and type of SQL statements submitted by a new application, for example, or to look into performance tuning opportunities with tools such as the Explain facility or the Design Advisor. Because it captures much more information for each activity affected by it, the impact of using this clause is higher on affected activities than other monitoring methods and it should be carefully controlled.

How does DB2 workload management work with the new AIX WPAR feature?

All aspects of DB2 workload management will work within an AIX WPAR but because AIX WPARs do not support the use of AIX WLM features, the option to tightly integrate DB2 service classes with AIX WLM service classes is of no benefit in this environment.

What is the relationship between the DB2_OPT_MAX_TEMP_SIZE registry variable and DB2 thresholds based on SQLTEMPSPACE?

There is no direct relationship between these two things. The DB2_OPT_MAX_TEMP_SIZE registry variable is a directive to the query compiler to limit the amount of temporary table space that a query can use. This can cause the optimizer to choose a plan that is more expensive (potentially less efficient) but which uses less space in the system temporary table spaces. A DB2 threshold based on SQLTEMPSPACE does not affect the type of plan chosen by the optimizer. It simply causes DB2 data server to monitor the usage of system temporary table space by that query at each member and generates a threshold violation if the stated limit is exceeded during normal processing.

Now that Query Patroller is discontinued and DB2 Governor is deprecated, how do I migrate to DB2 workload manager?

Following the introduction of DB2 workload manager as the strategic workload management solution in DB2 Version 9.5, Query Patroller has been discontinued in the Version 10.1 release and the DB2 Governor has been deprecated since the DB2 Version 9.7 release and might be removed in a future release.

Although DB2 Governor is still supported in this release, you should begin adopting the new features and capabilities of DB2 workload manager, including those introduced in this release. Note that with DB2 workload manager, you have many more options, and you should explore them, which might require you to rethink your approach to controlling work on your DB2 data server in current workload management terms. The DB2 best practices article [Implementing DB2 workload management in a data warehouse](#) contains a supplement that is specifically designed for those who are migrating from Query Patroller. Pertinent task topics are also available in the Related tasks section.

To facilitate migration from DB2 Query Patroller to DB2 workload manager, a sample script (`qpwlmmig.pl`) has been included starting with DB2 V9.7 Fix Pack 1. For additional information, see one of the following tasks for details on how to migrate from Query Patroller to DB2 workload manager:

- Migrating from Query Patroller to DB2 workload manager using the sample script
- Migrating from Query Patroller to DB2 workload manager

What are the licensing requirements for DB2 workload manager?

A subset of the workload management capabilities in DB2 data server has its use restricted through licensing. This licensed subset is referred to as DB2 workload manager, and it controls the creation of any service class, workload, threshold, or work action set. Access to this subset of workload management capabilities requires one of the following licensed products:

- DB2 Enterprise Server Edition for Linux, UNIX, and Windows
- DB2 Advanced Enterprise Server Edition for Linux, UNIX, and Windows
- Database Enterprise Developer Edition for Linux, UNIX, and Windows
- IBM InfoSphere Warehouse, all editions
- IBM Smart Analytics System

The following workload management functions are not restricted by license:

- Using or altering the default service classes and workloads; this includes all monitoring capabilities
- Creating, altering, or dropping histogram templates
- Using the DB2 workload management table functions or stored procedures
- Creating, activating, stopping, or dropping workload management event monitors
- Granting, altering, or revoking workload privileges

What information do you get from the different event monitors that are associated with workload management?

The threshold violations, statistics, and activities event monitors capture information about threshold violations, operational statistics and aggregate activity data, and individual activity data.

Each event monitor collects one or more logical data groups (see: "Event type mappings to logical data groups" in *Database Monitoring Guide and Reference*) and there are one or more monitoring elements in each logical data group (see: "Event monitor logical data groups and monitor elements" in *Database Monitoring Guide and Reference*).

For example, to discover what information is collected by the threshold violations event monitor, start by looking in Table 3 in "Event type mappings to logical data groups" topic. This table shows that the threshold violations event monitor collects information into a single logical data group called `event_thresholdviolations` (note that some event monitors, like the activity event monitor, collect information into multiple logical data groups). Next, find the `event_thresholdviolations` logical data group in "Event monitor logical data groups and monitor elements" topic. This topic shows which monitor elements are reported in the `event_thresholdviolations` logical data group, which includes the following:

- `activate_timestamp` - Activate timestamp
- `activity_collected` - Activity collected
- `activity_id` - Activity ID
- `agent_id` - Application Handle (agent ID)
- `appl_id` - Application ID
- `coord_partition_num` - Coordinator partition number
- `destination_service_class_id` - Destination service class ID
- `source_service_class_id` - Source service class ID
- `threshold_action` - Threshold action
- `threshold_maxvalue` - Threshold maximum value
- `threshold_predicate` - Threshold predicate
- `threshold_queuesize` - Threshold queue size
- `thresholdid` - Threshold ID
- `time_ofViolation` - Time of violation
- `uow_id` - Unit of work ID

The approach outlined in this example can be used to discover what data is collected by each event monitor.

How do I determine which activities are queued by a workload management threshold and the order of the activities in the queue?

You can do this by first creating a view using the `WLM_GET_SERVICE_CLASS_AGENTS` table function and then running statements to list the queued activities in the order of the queue entry time.

What changes in behavior might I see when I turn ON the workload management dispatcher?

When you turn ON the workload management dispatcher via the **wlm_dispatcher** database manager configuration parameter and if you had been relying on agent priority to prioritize the work of one service class over another, then this agent priority cannot be used while the workload management dispatcher is enabled. As a result, all service classes are treated as if they have the default agent priority.

If you enable CPU shares via the **wlm_disp_cpu_shares** database manager configuration parameter and do not specify CPU shares or CPU limits for your service classes, all service classes receive an equal soft share of the CPU resources on your system. The effect of all service classes receiving an equal soft share of the CPU resources might result in a different allocation of CPU resources to services classes than in previous DB2 releases. As a result, you should consider setting CPU shares or CPU limit values appropriate for your workload.

With the introduction of the workload management dispatcher, are concurrency thresholds such as CONCURRENTDBCOORDACTIVITIES no longer needed or useful?

The DB2 workload management dispatcher and concurrency thresholds can be used together. Concurrency thresholds are still very useful for controlling how much work is running. For each activity that starts running, the DB2 database manager provides other resources to that activity, in addition to CPU resources, which the activity usually retains for as long as it is running. Such non-CPU resources include (among others) the DB2 agent, sort memory, temporary table space, locks, and I/O. By preventing an activity from starting to run, those additional non-CPU resources are not consumed and are available for other activities.

In addition, concurrency thresholds can be applied at different points within the DB2 database manager to determine the origin of the work that is running. For example, putting a concurrency threshold on large queries coming from a specific workload limits the consumption or share of the resources available to that particular workload in a service class, as compared to other workloads contributing to the same service class.

In summary, concurrency thresholds can be used to control when activities start to run and consume the CPU and non-CPU resources on the system. The workload management dispatcher can be used to control how much of the CPU resources such activities get to consume once they start running.

Why would I ever want to use AIX WLM or Linux WLM?

Even if you use the DB2 workload management dispatcher for controlling the CPU consumption of your DB2 workloads, the following are reasons to use AIX WLM or Linux WLM as well:

- Operating system (OS) workload managers provide monitoring of resource consumption at the level of the operating system.
- OS workload managers can provide control for all processes or threads on the entire host or LPAR, not just DB2 database manager threads. This can help when there is a need to control processes that compete for resources with DB2 database manager.

Integration of AIX Workload Manager with DB2 workload management

On the AIX operating system, the optional integration between DB2 service classes and AIX WLM classes permits you to control the amount of processor resource allocated to each service class.

Implementing AIX WLM controls may not be needed to meet your performance objectives, but even if you do not need to exercise AIX WLM, the operating system statistics provided by AIX WLM per AIX class are often useful for monitoring and tuning efforts.

AIX WLM assigns relative or absolute amounts of processor resource as shares to classes which benefit from controls that you can change dynamically and that become effective immediately. If relative AIX CPU shares do not provide the level of control you require, you also have the choice of assigning hard maximum percentage of CPU resource. By doing so, you surrender some of the flexibility of relative CPU allocation, which is useful during off-peak times, but you also gain excellent and guaranteed control with a hard maximum limit on CPU time resource allocation.

Recommended mappings between DB2 service classes and AIX classes

Use a 1:1 mapping of DB2 service classes to AIX Workload Manager service classes to take advantage of AIX WLM processor controls. By having a 1:1 mapping between DB2 service classes and AIX Workload Manager service classes, you can adjust the AIX processor resource for each DB2 service class individually to meet your business priority goals.

The following figure shows the integration of DB2 workload management with the AIX Workload Manager. Note the 1:1 mapping between each DB2 service class and AIX Workload Manager service class at the service superclass and service subclass levels.

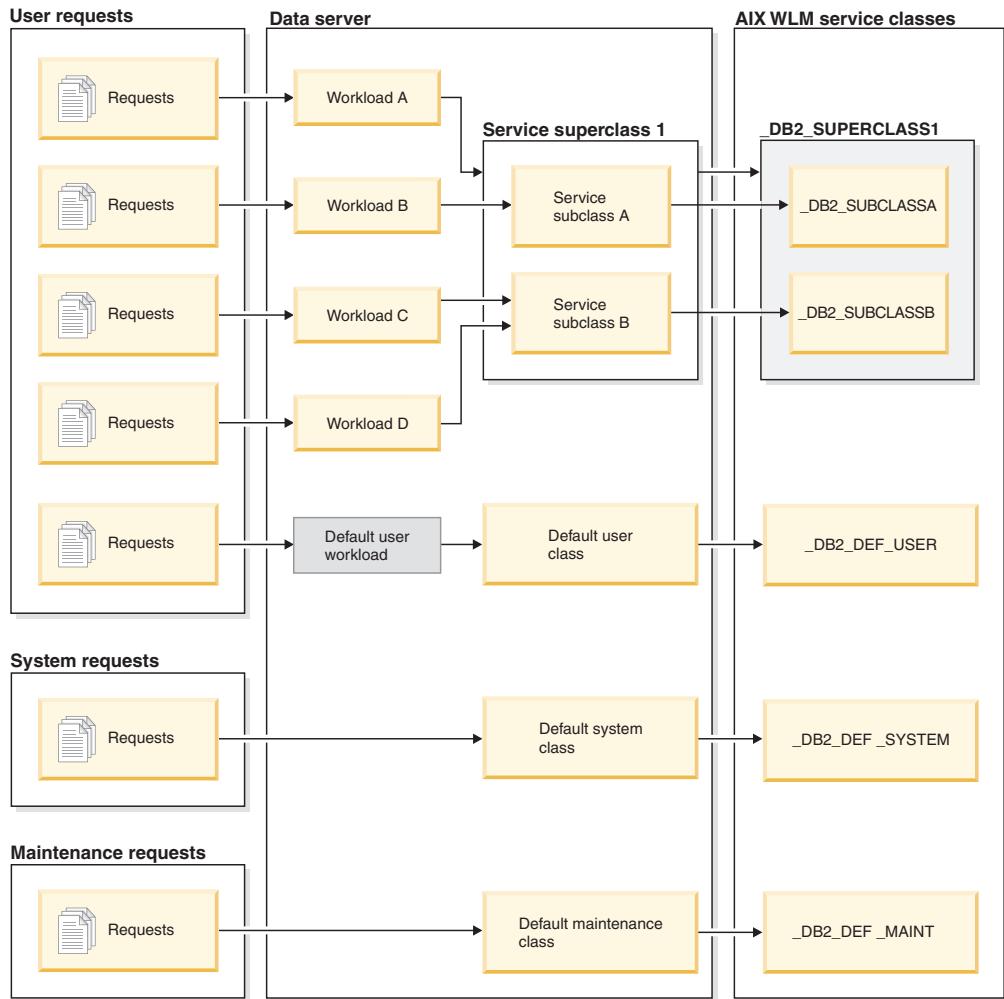


Figure 65. Integration of DB2 workload management with the AIX Workload Manager

When a DB2 environment consists of a single database in a single DB2 instance, such as the example portrayed in the previous figure, it is possible to map directly between DB2 service classes and AIX Workload Manager classes. Each DB2 service superclass can have a corresponding AIX Workload Manager service superclass and each DB2 service subclass can map to a corresponding AIX service subclass.

In situations where the DB2 environment consists of multiple databases and DB2 instances, several levels might be candidates for resource control. Because the AIX Workload Manager supports a two-level hierarchy, that is, superclass and subclass, only two levels of a DB2 environment can be mapped to AIX Workload Manager classes at any time. The following figure shows one way to achieve a 1:1 mapping with multiple databases, each with multiple superclasses. Here, each database has its own AIX Workload Manager superclass and each DB2 service superclass is mapped to an AIX Workload Manager subclass.

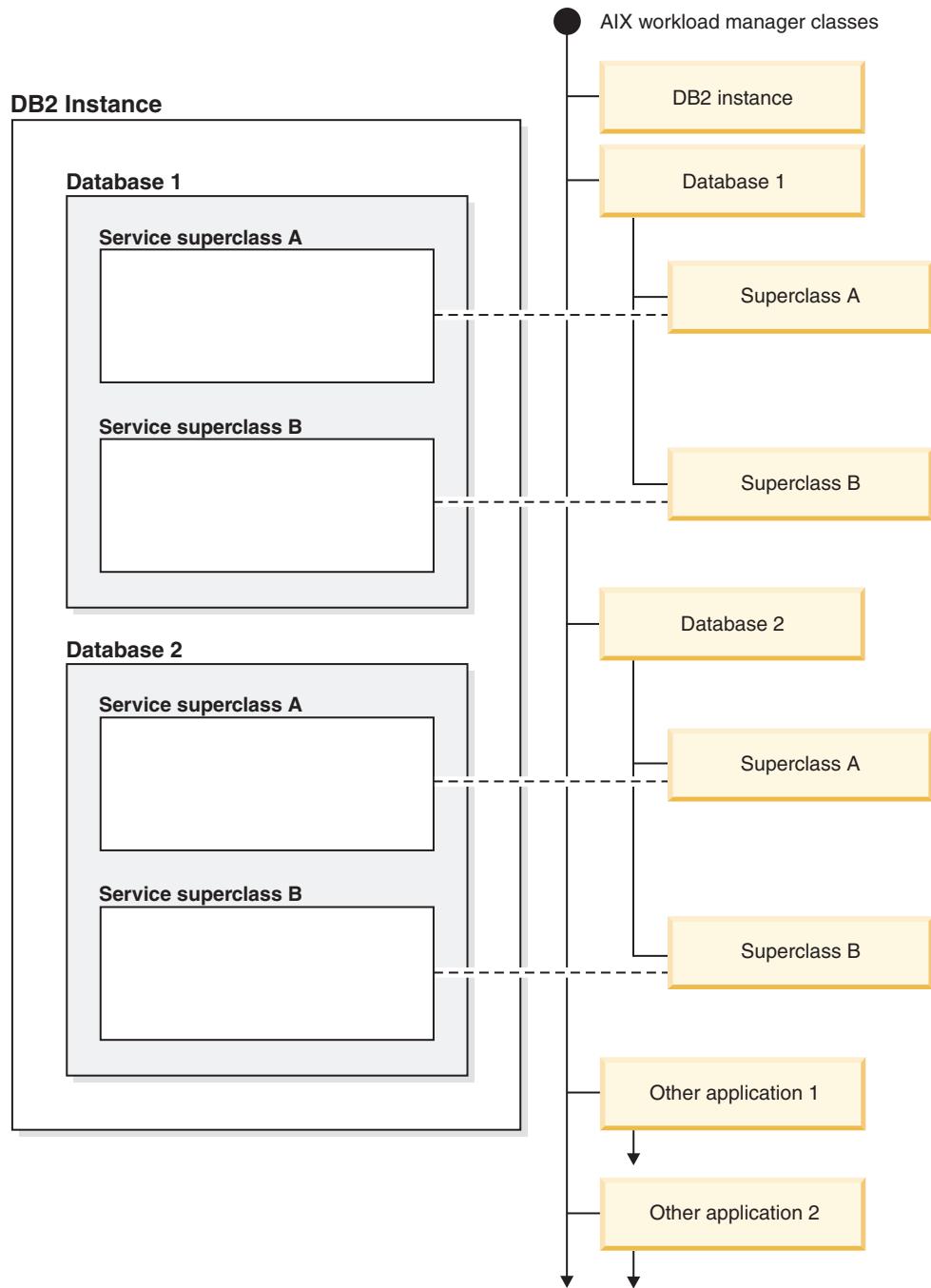


Figure 66. DB2 service classes mapped to AIX classes (with DB2 service superclasses only)

An alternative configuration is to map each DB2 service superclass to its own AIX Workload Manager superclass, which results in four superclasses in this example. In this situation, the database level of resource control is represented explicitly in the AIX Workload Manager service class definitions.

The following figure shows one way to achieve the 1:1 mapping in the situation where you have multiple databases, each with service superclasses and service subclasses. Here, each database corresponds to an AIX superclass and each DB2 service subclass is mapped to an AIX Workload Manager subclass. The DB2 service superclass is not shown explicitly in the AIX Workload Manager service class

definitions.

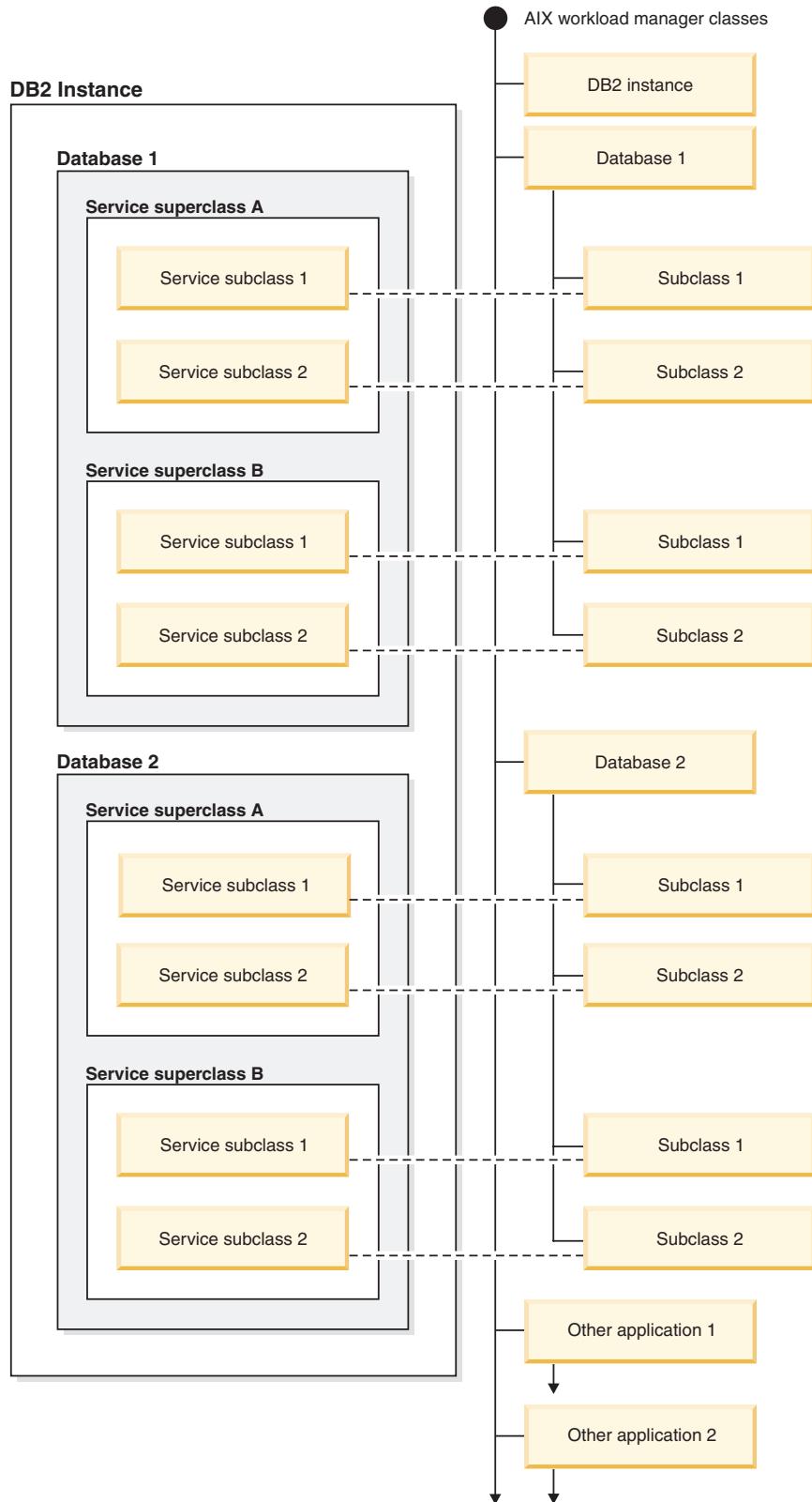


Figure 67. DB2 service classes mapped to AIX Workload Manager classes (with DB2 service subclasses)

Defining mappings between DB2 service classes and AIX classes

Mapping between DB2 service classes and AIX Workload Manager classes is specified for the DB2 service class using the OUTBOUND CORRELATOR keyword of the CREATE SERVICE CLASS or the ALTER SERVICE CLASS statements.

The steps for setting up the AIX Workload Manager classes with the DB2 data server are:

1. Create the DB2 service superclasses and service subclasses, and specify the OUTBOUND CORRELATOR tags.
2. Create the corresponding AIX classes.
3. Create the associated AIX Workload Manager rules files to contain the DB2 workload management to AIX Workload Manager mappings using the OUTBOUND CORRELATOR tags under the tag columns.
4. Start the AIX Workload Manager.
5. If required, set this AIX Workload Manager configuration as active.

When a thread joins a DB2 service class, the DB2 data server calls the appropriate AIX Workload Manager API to associate the thread to the corresponding AIX service class. The DB2 data server sends the thread's target AIX service class to the AIX Workload Manager by passing it the application tag set in the OUTBOUND CORRELATOR parameter.

You must ensure that the AIX Workload Manager is properly installed, configured, and active. If the DB2 data server cannot communicate with the AIX Workload Manager, a message is logged to the db2diag log files and DB2 administrator log. The database activity continues.

The DB2 data server cannot detect whether the OUTBOUND CORRELATOR value that it passes to the AIX Workload Manager is recognized by the AIX Workload Manager. You must verify that the value specified for the DB2 service class matches the application tags that map DB2 threads to the AIX service classes. If the OUTBOUND CORRELATOR value is not recognized by the AIX Workload Manager, the database activity continues to execute.

Other points to note are:

- DB2 service classes cannot work with the AIX Workload Manager inheritance feature. Inheritance is the default setting for an AIX service class; inheritance must be explicitly disabled by setting the inheritance attribute to NO. AIX Workload Manager inheritance forces all child threads and processes to map to the same class as the parent thread or process. If inheritance is enabled, DB2 workload management cannot change the AIX Workload Manager class of a thread by using tagging. This restriction makes any integration of DB2 workload management and the AIX Workload Manager unusable. The DB2 data server cannot detect whether AIX Workload Manager inheritance is enabled and does not issue an error message if inheritance is enabled.
- DB2 service classes are not compatible with the AIX Workload Manager manual assignment feature. With the manual assignment feature, users can manually assign a process to a specific AIX Workload Manager class. By manually assigning the DB2 process, all threads in the process are assigned to a target AIX Workload Manager class, the DB2 service class mapping logic is defeated and results are not predictable.

For more information on the AIX Workload Manager, see the AIX Information Center at <http://publib.boulder.ibm.com/infocenter/pseries/v5r3/index.jsp>

Setting processor controls on AIX classes

The AIX Workload Manager can be used to control the amount of processor resource allocated to each service class. Options include setting a minimum, maximum, or relative proportion share of processor resource for each service class.

When integrating the AIX Workload Manager with DB2 Workload Management, only processor resource allocation is supported. You should not set memory and I/O settings for the AIX classes. DB2 database-level memory is shared among all agents from different DB2 service classes, so you cannot divide memory allocation between different service classes. AIX-level I/O control does not support the DB2 engine threaded model. To control I/O, you can use the prefetcher priority attribute of a DB2 service class to differentiate I/O priorities between different DB2 service classes.

If you use AIX to control the amount of processor resource allocated to a service class, do not also change the agent priority setting for that DB2 service class. Use only one of these mechanisms to govern the access to processor resource. You cannot set both the AGENT PRIORITY and the OUTBOUND CORRELATOR value for a service class.

AIX Workload Manager settings should be consistent on all physical computers that participate in an instance. For example, if the resource setting for an AIX service class is set high on one computer, the same setting should be used for that AIX service class on all other computers. If the resource usage settings are inconsistent across computers, requests running in the same AIX service class will exhibit different performance levels on different database members. This situation can lead to poor overall throughput for connections in an AIX service class.

Integration of Linux workload management with DB2 workload management

On the Linux operating system, the optional integration between DB2 service classes and Linux classes (control groups) permits you to control the amount of processor resource allocated to each service class. If enabled, all threads running in a DB2 service class are mapped to a Linux class where they are subject to the processor resource controls you define.

To make use of Linux workload management support, you require a Linux kernel version 2.6.26 or later on a 64-bit system and the `libcgroup` library package.

Linux workload management supports a hierarchy of classes with superclasses and subclasses, with processor shares for subclasses divided in proportion to the shares of the parent class. These shares provide a method of control over processor resource such that all threads in the system will always run, but the amount of processor time each thread receives is dependent on the number of shares assigned to the Linux class.

Processor resource on the Linux operating system is assigned in shares relative to the Linux workload management default class, which by default has a processor share at a value of 1024. If you define no other Linux classes, all threads run in this default class. If you define a class that has a share value equal to 1024, then

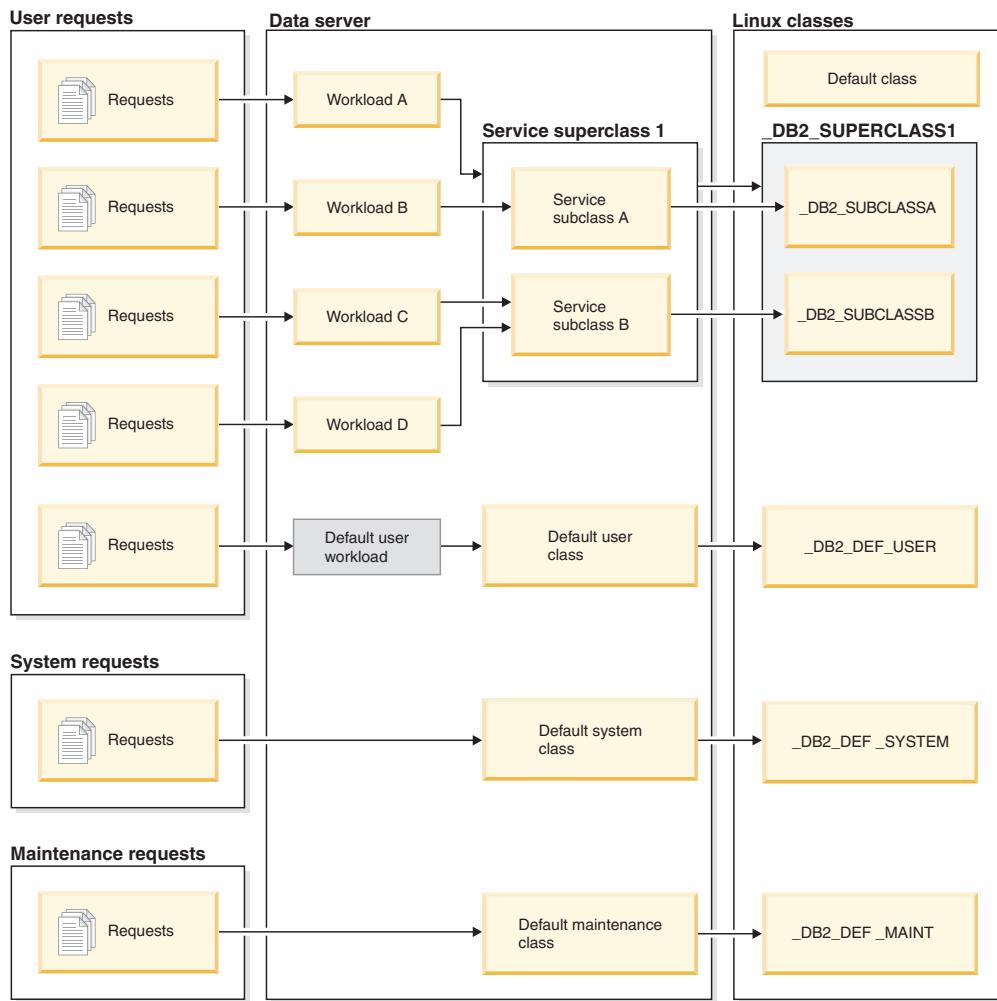
this class receives the same amount of processor resource as the Linux default class with the default processor share. Similarly, a class with a share of 2048 receives a target processor usage quota twice that of the default class. On more complex systems, you should consider raising the processor share of the Linux default class, which improves the granularity for shares across the system so that you can assign processor resources more accurately.

Recommended mappings between DB2 service classes and Linux classes

You should use a 1:1 mapping between DB2 service classes and Linux classes which permits you to adjust the Linux processor shares assigned to activities in each DB2 service class individually according to business priority. It is important that you associate every DB2 service class with a Linux WLM class, either by setting an outbound correlator for each service superclass and subclass, or through inheritance from the parent service class for subclasses. This includes the default SYSDEFAULTSYSTEMCLASS, SYSDEFAULTMAINTENANCECLASS and SYSDEFAULTUSERCLASS service classes.

The following figure shows how two DB2 service subclasses under the same user defined service superclass can get mapped 1:1 to Linux subclasses under a common superclass. In this example, the work identified and assigned by two workloads for each DB2 service subclass is subject to the processor resource controls imposed by the corresponding Linux subclasses (_DB2_SUBCLASSA, _DB2_SUBCLASSB). Also shown are three Linux classes that correspond to the default DB2 workload management service classes (_DB2_DEF_USER, _DB2_DEF_SYSTEM, _DB2_DEF_MAINT). If you integrate DB2 workload management with Linux workload management, you should always create these additional Linux classes to match the default DB2 service classes. To avoid any bottleneck, the Linux class corresponding to the DB2 default system class should receive more processor shares than any other Linux class that DB2 activities map to, whilst the Linux class corresponding to the default maintenance class should receive less processor shares.

Figure 68. Integration of DB2 workload management with Linux workload management



Defining mappings between DB2 service classes and Linux workload management classes

The steps for integrating DB2 workload management with Linux workload management, which runs as an operating system service, are as follows:

1. Define the Linux classes, class permissions, and processor shares by editing the `/etc/cgconfig.conf` control groups configuration file. What Linux classes you create depends on the conditions dictated by your business priorities for the work your data server performs. If you want to apply processor resource based on the source of certain work, for example, create a Linux class to match the DB2 service class that work is going to be assigned to by the workload identifying the work. Define an entry for each Linux class corresponding to the DB2 service class to be created that you want to use for the mapping. The following sections must be provided in the `/etc/cgconfig.conf` configuration file:
 - **group:** The Linux class name. For example, if you specify `group _class1`, you create a superclass `_class1`. If you specify `group _class1/_subclass1`, you create the subclass `_subclass1` under the superclass `_class1`.
 - **perm:** The permissions section that determines who can control what threads are assigned to a Linux class and who can change the processor shares of classes in the `/etc/cgconfig.conf` configuration file.

- task: The user ID (**uid**) and group ID (**gid**) whose threads can run in the Linux workload management class. To enable Linux workload management to work with DB2 workload management, you should set **uid** to the DB2 instance owner user ID.
- admin: The user ID (**uid**) and group ID (**gid**) that can change processor shares for a Linux workload management class.
- cpu: The processor shares definition section
 - **cpu.shares**: The share assigned to this Linux class relative to the default class

The `/etc/cgconfig.conf` configuration file must contain these sections in the following format:

```
# Superclass name
group _name
{
    perm
    {
        task
        {
            uid = db2inst1;
            gid = db2iadm1;
        }
        admin
        {
            uid = db2inst1;
            gid = db2iadm1;
        }
    }

    cpu
    {
        cpu.shares = 1024;
    }
}
```

2. Start the Linux workload management service daemon with the **service cgconfig start** command, then start your DB2 data server with the **db2start** command.
3. To map a DB2 service class to one of the Linux classes, include the Linux class name in the OUTBOUND CORRELATOR clause when you create or alter the service class, which associates threads from the DB2 service class with the external Linux class.
4. If you want to find out what threads are assigned to a particular Linux class, you can use the cat command on the `/cgroup/class_name/tasks` file, where `class_name` represents the name of the Linux class you are interested in. All threads that are not mapped to a user-defined Linux class are assigned to the Linux default class, which you can find at `MOUNTPOINT/sysdefault`, where `MOUNTPOINT` is defined in the `cgconfig.conf` configuration file.
5. To add or remove Linux classes, you must stop with the Linux workload management service with the **service cgconfig stop** command, make your changes, and then restart the service. Note that stopping the service affects the entire system, because all tasks are moved to the default class. If you used the `/etc/init.d/cgred` script to start the service daemon, issue **/etc/init.d/cgred stop** to stop it.

For the integration with DB2 workload management to work, you must ensure that the Linux workload management service is properly installed, configured, and active. If the DB2 data server cannot communicate with the Linux workload

management service, a message is logged to the db2diag log files and DB2 administrator log. Database activities will continue to execute.

The DB2 data server cannot detect whether the outbound correlator that it passes to external workload managers is recognized by Linux workload management. You must verify that the OUTBOUND CORRELATOR value specified for a DB2 service class matches the Linux class name so that DB2 threads are mapped to the Linux class. If an outbound correlator is not recognized, database activities will continue to execute.

Example

The following example illustrates how you can make use of Linux workload management processor controls by integrating with DB2 workload management. In this example, we create two user-defined DB2 service classes, one for batch applications (BATCHAPPS) and one for online applications (ONLINEAPPS). For simplicity, this example does not show the default service classes, which should be included in an implementation that creates the recommended 1:1 mapping between DB2 service classes and Linux classes. Because response time is critical for the online applications, we want the ONLINEAPPS service class to receive three times the amount of processor shares relative to work that runs in the Linux default class ($3 \times 1024 = 3072$ shares). Batch applications have a lower business priority, and the BATCHAPPS class should be assigned half the processor resource of work that runs in the Linux default class ($1024 / 2 = 512$ shares). All other work on the system will run in the Linux default class. Note that this example does not create Linux classes corresponding to the three default DB2 workload management service classes.

To create this setup, first create the two corresponding Linux classes _BATCHAPPS and _ONLINEAPPS and set their relative processor shares by editing the /etc/cgconfig.conf tasks file. After editing, the tasks file contains the following two entries, one for each Linux class:

```
# Superclass ONLINEAPPS
group _ONLINEAPPS
{
    perm
    {
        task
        {
            uid = db2inst1;
            gid = db2iadm1;
        }
        admin
        {
            uid = db2inst1;
            gid = db2iadm1;
        }
    }

    cpu
    {
        # 3 x 1024 = 3072 shares
        cpu.shares = 3072;
    }
}

# Superclass BATCHAPPS
group _BATCHAPPS
{
    perm
```

```

{
  task
  {
    uid = db2inst1;
    gid = db2iadm1;
  }
  admin
  {
    uid = db2inst1;
    gid = db2iadm1;
  }
}

cpu
{
  # 1024 / 2 = 512 shares
  cpu.shares = 512;
}
}

```

The absolute processor time in percent assigned to each Linux class as processor shares is as follows:

Table 26. Processor shares and absolute processor time assigned to Linux classes

Linux class	Shares	Absolute processor time in percent
Default class	1024 (default)	1024 / 4608 = 22%
_ONLINEAPPS	1024 x 3 = 3072	3072 / 4608 = 67%
_BATCHAPPS	1024 x ½ = 512	512 / 4608 = 11%
	Total = 1024 + 3072 + 512 = 4608 shares	

Once the Linux WLM classes are created, you can start the Linux workload management service:

```
service cgconfig start
```

Next, create the associated DB2 service classes with the following statements:

```
DB2 CREATE SERVICE CLASS BATCHAPPS OUTBOUND CORRELATOR '_BATCHAPPS'
DB2 CREATE SERVICE CLASS ONLINEAPPS OUTBOUND CORRELATOR '_ONLINEAPPS'
```

To find out which threads are running in a Linux class, issue the cat command. For the business critical _ONLINEAPPS Linux class, the command and output look as follows. You can see that there are six thread running in this Linux class:

```
cat /cgroup/_ONLINEAPPS/tasks
```

```
1056
1087
1107
985
1036
1205
```

Workload management sample application

Comprehensive workload management features have been integrated into your DB2 data server with DB2 workload management, giving you finer control over activities, resources and performance, and deeper insight into how your system is running. A workload management sample application is now available on developerWorks®.

The workload management sample application demonstrates how you can use DB2 workload management features to achieve the following objectives:

Protect the system from runaway queries

Runaway queries are costly and cause poor performance. The workload management sample application identifies queries with the potential to become runaway queries, and then stops these queries from running after they have violated a specified threshold.

Limit concurrent resource consumption by individual applications

The sample application shows how to use DB2 workload management features to prevent applications that submit large amounts of concurrent work from negatively affecting the performance of other applications.

Achieve a specific response time

Workload management features permit you to achieve a specific response time objective of the form: "transaction X from application Y shall complete within 1 second in 90% of cases," regardless of what other activity is running concurrently on the system. The sample application will demonstrate how to achieve a response time objective.

Consistent response time for short queries

Queries that typically have a response time of less than 1 second should have a relatively consistent response time regardless of what other workloads are running on the system. The sample application uses the query execution time histogram to monitor consistency.

Protect the system during periods of peak demand

Workload management policy features protect the system from capacity overload during bursts of peak demand by queuing work once the system is sufficiently loaded.

Enable concurrent batch extract, transform, and load (ETL) processing and user queries

Workload management features permit you to run ETL jobs (like loading data into tables) while controlling the performance impact for users running queries concurrently.

To obtain the sample application, see Workload management sample on developerWorks.

Workload management scenarios

These scenario illustrates the different uses of WLM to monitor activities.

Scenario: Investigating a workload-related system slowdown

If you notice a system slowdown (for example, some applications take much longer to complete than expected) and are unsure whether the problem is related to the configuration of the workloads, you can use table function data to investigate and, if necessary, correct the problem.

First, create a query that aggregates data across service classes and database members using data from the WLM_GET_SERVICE_SUBCLASS_STATS table function. Set the first and second arguments to empty strings and the third argument to -2 to indicate that data is to be gathered for all service classes on all database members.

Your query might resemble the following one:

```
SELECT SUBSTR(SERVICE_SUPERCLASS_NAME,1,19) AS SUPERCLASS_NAME,
       SUBSTR(SERVICE_SUBCLASS_NAME,1,18) AS SUBCLASS_NAME,
       SUBSTR(CHAR(SUM(COORD_ACT_COMPLETED_TOTAL)),1,13) AS ACTSCOMPLETED,
       SUBSTR(CHAR(SUM(COORD_ACT_ABORTED_TOTAL)),1,11) AS ACTSABORTED,
       SUBSTR(CHAR(MAX(CONCURRENT_ACT_TOP)),1,6) AS ACTSHW,
       CAST(CASE WHEN SUM(COORD_ACT_COMPLETED_TOTAL) = 0 THEN 0
                 ELSE SUM(COORD_ACT_COMPLETED_TOTAL * COORD_ACT_LIFETIME_AVG)
                     / SUM(COORD_ACT_COMPLETED_TOTAL) END / 1000 AS DECIMAL(9,3))
              AS ACTAVGLIFETIME
  FROM TABLE(WLM_GET_SERVICE_SUBCLASS_STATS('', '', -2)) AS SCSTATS
 GROUP BY SERVICE_SUPERCLASS_NAME, SERVICE_SUBCLASS_NAME
 ORDER BY SUPERCLASS_NAME, SUBCLASS_NAME
```

SUPERCLASS_NAME	SUBCLASS_NAME	ACTSCOMPLETED	ACTSABORTED	ACTSHW	ACTAVGLIFETIME
SYSDEFAULTUSERCLASS	SYSDEFAULTSUBCLASS	20	0	1	3.750
SUP1	SUB1	40	0	8	7.223

In the preceding example data, the SUB1 service subclass in the SUP1 service superclass is running more simultaneous activities than usual. To investigate further, you might want to examine the statistics for workloads that map to this service class. Your query might resemble the following one:

```
SELECT SUBSTR(WLSTATS.WORKLOAD_NAME,1,22) AS WL_NAME,
       SUBSTR(CHAR(WLSTATS.MEMBER),1,4) AS MEMB,
       CONCURRENT_WLO_TOP AS WLO_HIGH_WTRMRK,
       CONCURRENT_WLO_ACT_TOP AS WLO_ACT_HIGH_WTRMRK
  FROM TABLE(WLM_GET_WORKLOAD_STATS('', -2)) AS WLSTATS,
       TABLE(WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES('', '', -2)) AS SCWLOS
 WHERE WLSTATS.WORKLOAD_NAME = SCWLOS.WORKLOAD_NAME
   AND SCWLOS.SERVICE_SUPERCLASS_NAME = 'SUP1'
   AND SCWLOS.SERVICE_SUBCLASS_NAME = 'SUB1'
 ORDER BY WL_NAME, MEMB;
```

WL_NAME	MEMB	WLO_HIGH_WTRMRK	WLO_ACT_HIGH_WTRMRK
LYNNSALES	0	2	8
LYNNSALES	1	0	0
SYSDEFAULTUSERWORKLOAD	0	1	1
SYSDEFAULTUSERWORKLOAD	1	0	0

The output shows that an application in the LYNNSALES workload submitted 8 activities concurrently. Consider adding a threshold to restrict concurrency of coordinator activities for each workload occurrence.

Scenario: Identifying activities that are taking too long to complete

Workload management table functions simplify the task of identifying a specific activity inside the data server and, if necessary, canceling it without having to end the entire application.

Identifying an activity that is taking too long to complete

Following is an example of identifying a long-running query. Assume that a user from the Sales department who is running the SalesReport application complains that the application is taking too long to complete.

After identifying the application handle, use the WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function to look up all activities currently running in this application. For example, if the application handle is 1, your query might resemble the following one:

```
SELECT SUBSTR(CHAR(COORD_MEMBER),1,5) AS COORD,
       SUBSTR(CHAR(MEMBER),1,4) AS MEMB,
       SUBSTR(CHAR(UOW_ID),1,5) AS UOWID,
       SUBSTR(CHAR(ACTIVITY_ID),1,5) AS ACTID,
       SUBSTR(CHAR(PARENT_UOW_ID),1,8) AS PARUOWID,
       SUBSTR(CHAR(PARENT_ACTIVITY_ID),1,8) AS PARACTID,
       SUBSTR(ACTIVITY_TYPE,1,8) AS ACTTYPE,
       SUBSTR(CHAR(NESTING_LEVEL),1,7) AS NESTING
  FROM TABLE(WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES(1, -2))
     AS WLOACTS
 ORDER BY MEMB, UOWID, ACTID
```

COORD	MEMB	UOWID	ACTID	PARUOWID	PARACTID	ACTTYPE	NESTING
0	0	2	3	-	-	CALL	0
0	0	2	5	2	3	READ_DML	1

The activity is identified as having a unit of work ID of 2 and an activity ID of 5. You can then use the WLM_GET_SERVICE_CLASS_AGENTS table function to discover what the agents that work on this activity are doing:

```
SELECT APPLICATION_HANDLE, UOW_ID, ACTIVITY_ID,
       SUBSTR(REQUEST_TYPE,1,8) AS REQUEST_TYPE,
       SUBSTR(EVENT_TYPE,1,8) AS EVENT_TYPE,
       SUBSTR(EVENT_OBJECT,1,8) AS EVENT_OBJECT
  FROM TABLE(WLM_GET_SERVICE_CLASS_AGENTS('', '', CAST(NULL AS BIGINT),-2))
     AS AGENTS
 WHERE APPLICATION_HANDLE = 1
   AND UOW_ID = 2
   AND ACTIVITY_ID = 5
```

For example, the activity might be queued, executing, or waiting on a lock. If the activity were queued, the result would be:

APPLICATION_HANDLE	UOW_ID	ACTIVITY_ID	REQUEST_TYPE	EVENT_TYPE	EVENT_OBJECT
1	2	5	OPEN	WAIT	WLM_QUEUE

If the activity were executing, the result would be:

APPLICATION_HANDLE	UOW_ID	ACTIVITY_ID	REQUEST_TYPE	EVENT_TYPE	EVENT_OBJECT
1	2	5	OPEN	PROCESS	REQUEST

If the activity were waiting on a lock, the result would be:

APPLICATION_HANDLE	UOW_ID	ACTIVITY_ID	REQUEST_TYPE	EVENT_TYPE	EVENT_OBJECT
1	2	5	OPEN	ACQUIRE	LOCK

When you know what the activity is doing, you can proceed appropriately:

- If the activity is queued, if the user indicates that the query is taking so long that they no longer care about the results, or you think the query is consuming too many resources, you can cancel it.

- If the activity is important and it is queued, consider cancelling some other less important work that is currently running (reducing the concurrency so that activities leave queue), or maybe the user will be satisfied to know that work is not hanging and is just waiting for chance to run.
- If the activity is waiting for a lock, you can use the snapshot monitor to investigate which locks the application is waiting for.
- If the activity is waiting for a lock held by lower priority activity, consider cancelling that activity.

You might also find it useful to know the DML statement that activity 5 is running. Assuming that you have an active activities event monitor, you can run the **WLM_CAPTURE_ACTIVITY_IN_PROGRESS** procedure to capture information about the DML statement and other information about activity 5 while it is running. Unlike the statement event monitor, the **WLM_CAPTURE_ACTIVITY_IN_PROGRESS** procedure permits you to capture information about a specific query, as opposed to every statement running at the time. You can also obtain the statement text by using **MON_GET_ACTIVITY_DETAILS**.

If you decide that you must cancel the activity, you can use the **WLM_CANCEL_ACTIVITY** routine to cancel the activity without having to end the application that issued it:

```
CALL WLM_CANCEL_ACTIVITY (1, 2, 5)
```

The application that issued the activity receives an SQL4725N error. Any application that handles negative SQL codes is able to handle this SQL code.

Identifying an activity hang caused by lock contention

Assume that you have a situation in which a user is complaining about an application that is taking too long. Also assume that you have either the application name or the authorization ID of the long-running application. With this information, you can use the **LIST APPLICATIONS** command to obtain the application handle. Assuming that application handle returned by the **LIST APPLICATIONS** command is 2, you can use the **WLM_GET_SERVICE_CLASS_AGENTS** table function to determine which agents are working on this activity. Your query might resemble the following one:

```
SELECT SUBSTR(CHAR(APPLICATION_HANDLE),1,7) AS APPHANDLE,
       SUBSTR(CHAR(MEMBER),1,4) AS MEMB,
       SUBSTR(CHAR(AGENT_TID),1,9) AS AGENT_TID,
       SUBSTR(AGENT_TYPE,1,11) AS AGENTTYPE,
       SUBSTR(EVENT_OBJECT,1,11) AS EVENTOBJECT,
       SUBSTR(REQUEST_TYPE,1,7) AS REQTYPE,
       SUBSTR(CHAR(UOW_ID),1,6) AS UOW_ID,
       SUBSTR(CHAR(ACTIVITY_ID),1,6) AS ACT_ID
  FROM TABLE(WLM_GET_SERVICE_CLASS_AGENTS(' ', ' ', 2, -2)) AS SCDETAILS
 ORDER BY APPHANDLE, MEMB, AGENT_TID
```

APPHANDLE	MEMB	AGENT_TID	AGENTTYPE	EVENTOBJECT	REQTYPE	UOW_ID	ACT_ID
2	0	1	COORDINATOR	REQUEST	OPEN	2	1
2	1	3	SUBAGENT	LOCK	-	2	1

The results indicate that agent 1 is waiting on a remote response. Looking at the agent on the remote member that is working on the same activity, the **EVENTOBJECT** field indicates that the agent is waiting to obtain a lock.

The next step is to determine who owns the lock. You can obtain this information by turning on the monitor switches and using the snapshot monitor table function, as shown in the following example:

```
SELECT AGENT_ID AS WAITING_FOR_LOCK,
       SUBSTR(APPL_ID HOLDING_LK,1,40) AS HOLDING_LOCK,
       CAST(LOCK_MODE REQUESTED AS SMALLINT) AS WANTED,
       CAST(LOCK_MODE AS SMALLINT) AS HELD
  FROM TABLE(SNAPSHOT_LOCKWAIT('SAMPLE',-1)) AS SLW
```

WAITING_FOR_LOCK	HOLDING_LOCK	WANTED	HELD
2 *LOCAL.DB2.060131021547		9	5

You can also determine the lock owner by using the following sequence of commands:

```
db2pd -db database alias -locks
db2pd -db database alias -transactions
```

If you want to cancel the long-running activity, you can use the WLM_CANCEL_ACTIVITY procedure. If the successful completion of the long-running application is more important than the successful completion of the lock-owning application, you can force the lock-owning application.

Scenario: How to cancel activities queued for more than an hour

Using the example scripts described here, you can create a procedure to cancel activities that have been queued for more than an hour. In addition, an example script is provided that can be used to schedule the queued-activity-cancelling procedure to run every 10 minutes using the DB2 Administrative Task Scheduler.

The queued-activity-cancelling procedure also captures information about the cancelled activities (if an activity event monitor is active), and maintains a small history table of cancelled activities. Both of these informational components are optional and comments in the example script indicate where to comment out the components, if they are not required.

The statements contained in the example procedure are themselves activities and subject to threshold control (depending on how thresholds are configured on your system). Consider running the example queued-activity-cancelling procedure in a service class that does not have any queuing thresholds applied.

1. Copy the following example script, that creates the procedure to cancel activities queued for more than 1 hour, into a file you have created (for example, a file named x.clp):

```
-- Simple history table to track cancelled
-- activities

CREATE TABLE SAMPLE.CANCELED_ACTIVITIES(
  APPLICATION_HANDLE BIGINT,
  UOW_ID BIGINT,
  ACTIVITY_ID BIGINT )@

-- Cancel any activities that have been queued
-- for more than 1 hour

CREATE PROCEDURE SAMPLE.CANCEL_QUEUED_ACTIVITIES()
  LANGUAGE SQL
  BEGIN
    DECLARE APPHANDLE     BIGINT;
```

```

DECLARE UOWID      BIGINT;
DECLARE ACTIVITYID BIGINT;
DECLARE QUEUETIME  BIGINT;
DECLARE AT_END      INT DEFAULT 0;

DECLARE QUEUEDAPPS CURSOR WITH HOLD FOR SELECT APPLICATION_HANDLE,
    UOW_ID, ACTIVITY_ID
    FROM TABLE(WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES(NULL,-2)) AS T
    WHERE ACTIVITY_STATE = 'QUEUED' AND LOCAL_START_TIME IS NULL;

DECLARE QTIMECUR CURSOR FOR SELECT TIMESTAMPDIFF(8, CHAR
    (CURRENT TIMESTAMP - TIMESTAMP(VALUE)))
    FROM TABLE(WLM_GET_ACTIVITY_DETAILS(APPHANDLE ,
    UOWID , ACTIVITYID , -2)) AS T WHERE NAME = 'ENTRY_TIME';

DECLARE CONTINUE HANDLER FOR NOT FOUND
    SET AT_END = 1;

-- Ignore errors for activity not found and activity event
-- monitor does not exist.
DECLARE CONTINUE HANDLER FOR SQLSTATE '5U035', SQLSTATE '01H53'
    BEGIN
    END;

-- Find all activities that are queued by WLM
-- thresholds where (ACTIVITY_STATE = 'QUEUED')
OPEN QUEUEDAPPS;
FETCH QUEUEDAPPS INTO APPHANDLE, UOWID, ACTIVITYID;

WHILE AT_END = 0 DO

-- Now use activity entry time to estimate the time spent queued.
-- Queuing occurs before an activity begins execution, so queue
-- time is approximated using current time - entry time
OPEN QTIMECUR;
FETCH QTIMECUR INTO QUEUETIME;
CLOSE QTIMECUR;

IF ( QUEUETIME >= 1 ) THEN

-- Optional: Insert a record into a table to record the
-- cancellation of the statement (for monitoring purposes, to
-- understand how many statements were cancelled). Modify this
-- insert as required to capture more info such as the name of
-- the application that submitted the cancelled query. Comment out
-- these 2 lines if the monitoring is not important to you.
    INSERT INTO SAMPLE.CANCELED_ACTIVITIES VALUES ( APPHANDLE,
        UOWID, ACTIVITYID );

-- Optional: Send details about activity to any activity activities
-- event monitor before cancelling. Comment out
-- this line if you don't care about the details of the
-- statements that were cancelled
    CALL WLM_CAPTURE_ACTIVITY_IN_PROGRESS( APPHANDLE, UOWID,
        ACTIVITYID );

-- Cancel the activity
    CALL WLM_CANCEL_ACTIVITY( APPHANDLE, UOWID, ACTIVITYID );

-- Explicit commit, required for the insert statement above. The
-- admin task scheduler will not perform a commit. Comment out this
-- line if the insert statement is removed.
    COMMIT;

END IF;

FETCH QUEUEDAPPS INTO APPHANDLE, UOWID, ACTIVITYID;

```

- ```

 END WHILE;

 CLOSE QUEUEDAPPS;

 END@
```
2. Create the queued-activity-cancelling procedure by executing script x.clp using the following command:  
`db2 -td@ -f x.clp`
  3. Execute the queued-activity-cancelling procedure by issuing the following command:  
`db2 "call sample.cancel_queued_activities()"`  
Any activities that have been queued for more than 1 hour will be cancelled.
  4. The following example script schedules the queued-activity-cancelling procedure to run every 10 minutes using the DB2 Administrative Task Scheduler. Copy the example script into a file you have created (for example, a file named y.clp):
- ```

-----  

-- Enable DB2 Admin Task Scheduler if  

-- not already enabled.  

-----  

!db2set DB2_ATS_ENABLE=YES@  

-----  

-- Create SYSTOOLSPACE tablespace.  

-- Enable if SYSTOOLSPACE does not already  

-- exist on your database.  

-----  

-- CREATE TABLESPACE SYSTOOLSPACE IN IBMCATGROUP MANAGED BY AUTOMATIC STORAGE  

-- EXTENTSIZE 4@  

-----  

-- Add a task to automatically cancel  

-- activities that have been queued  

-- for more than 1 hour. Task is scheduled  

-- to run every 10 minutes. Adjust the  

-- schedule as necessary using the  

-- schedule input parameter (specified in  

-- cron format).  

-----  

CALL SYSPROC.ADMIN_TASK ADD(
    'CANCEL ACTIVITIES QUEUED FOR MORE 1 HOUR',
    NULL,
    NULL,
    NULL,
    '*/10 * * * *',
    'SAMPLE',
    'CANCEL_QUEUED_ACTIVITIES',
    NULL,
    NULL,
    NULL )@
```
5. Schedule the queued-activity-cancelling procedure to run every 10 minutes by executing script y.clp using the following command:
`db2 -td@ -f y.clp`

Scenario: Moving table spaces to different storage devices

This scenario uses multi-temperature storage to set up a database system that uses service classes of different priorities, each class using a different type of storage device.

Assume there are two storage devices: disk and SSD. You can set up a system to run short queries, based on the estimated cost, in a high-priority service class and large queries, based on the estimated cost, in a low-priority service class. In the low-priority service class, the number of large queries that can run concurrently is throttled. Initially, the table space data is on SSD storage, but as the data ages, you move it to slower storage.

To set up your database to use multi-temperature storage:

1. Create the storage group SSDGROUP for SSD and the storage group DISKGROUP for disk.

```
CREATE STOGROUP SSDGROUP on '/db2/ssdsystem' DEVICE READ RATE 350;
CREATE STOGROUP DISKGROUP on '/db2/disksystem' DEVICE READ RATE 70;
```

2. Create the initial table space, initially in the SSDGROUP storage group.

```
CREATE TABLESPACE Q1_2010_TBSPC MANAGED BY AUTOMATIC STORAGE
    USING SSDGROUP PAGESIZE 8K
```

3. Create a service superclass that contains two subclasses: one for short queries and one for long queries:

```
CREATE SERVICE CLASS SC_SUPER;
CREATE SERVICE CLASS SC_HIGH UNDER SC_SUPER SOFT CPU SHARES 5000;
CREATE SERVICE CLASS SC_LOW UNDER SC_SUPER HARD CPU SHARES 2000;
```

Note: You can alternatively use AIX WLM or Linux WLM integration to limit the impact of low-priority work.

4. Map all user work to the SC_SUPER service superclass by altering the SYSDEFAULTUSERWORKLOAD workload:

```
ALTER WORKLOAD SYSDEFAULTUSERWORKLOAD SERVICE CLASS SC_SUPER
```

5. Create a work class set to isolate long queries from short queries based on the estimated cost:

```
CREATE WORK CLASS SET WLM_WCS
(WORK CLASS WLM_DML_SHORT WORK TYPE DML FOR TIMERONCOST FROM 1 to 1000,
 WORK CLASS WLM_DML_LONG WORK TYPE DML FOR TIMERONCOST FROM 1001 to UNBOUNDED)
```

6. Create a work action set that maps the short queries to the high-priority service class and the long queries to the low-priority service class:

```
CREATE WORK ACTION SET WLM_WAS for SERVICE CLASS SC_SUPER
    USING WORK CLASS SET WLM_WCS
    (WORK ACTION WLM_MAP_HIGH_WA ON WORK CLASS WLM_DML_SHORT
        MAP ACTIVITY TO SC_HIGH,
     WORK ACTION WLM_MAP_LOW_WA ON WORK CLASS WLM_DML_LONG
        MAP ACTIVITY TO SC_LOW)
```

7. Create a threshold named LIMITLOW that limits the number of concurrent long activities to five:

```
CREATE THRESHOLD LIMITLOW FOR SERVICE CLASS SC_LOW
    UNDER SC_SUPER ACTIVITIES ENFORCEMENT DATABASE
    WHEN CONCURRENTDBCOORDACTIVITIES > 5 CONTINUE
```

8. Create a new table space to hold the data for the new quarter and move the data from last quarter from SSD storage to disk storage:

```
CREATE TABLESPACE Q2_2010_TBSPC MANAGED BY AUTOMATIC STORAGE
    USING SSDGROUP PAGESIZE 8K;
ALTER TABLESPACE Q1_2010_TBSPC USING STOGROUP DISKGROUP;
```

Note: If the transfer rate of the SSD device is slower than what is noted in the catalog table, the estimated cost of queries using the device increases, and the queries using that device are mapped to the low-priority service class.

Additional scenarios

For more information about additional scenarios, visit <http://pic.dhe.ibm.com/infocenter/db2luw/v10r1/topic/com.ibm.db2.luw.admin.wlm.doc/doc/c0052465.html>.

DB2 workload management tutorial

You can use this tutorial to learn how to set up DB2 workload management and to perform basic operations with it. Each exercise highlights one or more of the workload management capabilities that are available with DB2 workload management. The tutorial is available at <http://pic.dhe.ibm.com/infocenter/db2luw/v10r1/topic/com.ibm.db2.luw.admin.wlm.doc/doc/c0053139.html>.

Chapter 17. High availability disaster recovery (HADR)

The high availability disaster recovery (HADR) feature provides a high availability solution for both partial and complete site failures. HADR protects against data loss by replicating data changes from a source database, called the *primary database*, to one or more target databases, called the *standby databases*.

A partial site failure can be caused by a hardware, network, or software (DB2 database system or operating system) failure. Without HADR, a partial site failure requires restarting the database management system (DBMS) server that contains the database. The length of time that it takes to restart the database and the server where it is located is unpredictable. It can take several minutes before the database is brought back to a consistent state and made available. With HADR, a standby database can take over in seconds. Further, you can redirect the clients that used the original primary database to the new primary database by using automatic client reroute or retry logic in the application.

A complete site failure can occur when a disaster, such as a fire, causes the entire site to be destroyed. However, because HADR uses TCP/IP for communication between the primary and standby databases, they can be situated in different locations. For example, the primary database might be located at your head office in one city, and a standby database might be located at your sales office in another city. If a disaster occurs at the primary site, data availability is maintained by having the remote standby database take over as the primary database with full DB2 functionality. After a takeover operation occurs, you can bring the original primary database back up and return it to its primary database status; this is known as *failback*. You can initiate a failback if you can make the old primary database consistent with the new primary database. After you reintegrate the old primary database into the HADR setup as a standby database, you can switch the roles of the databases to enable the original primary database to once again be the primary database.

With HADR, you base the level of protection from potential loss of data on your configuration and topology choices. Some of the key choices that you must make are as follows:

What level of synchronization will you use?

Standby databases are synchronized with the primary database through log data that is generated on the primary and shipped to the standbys. The standbys constantly roll forward through the logs. You can choose from four different synchronization modes. In order of most to least protection, these are SYNC, NEARSYNC, ASYNC, and SUPERASYNC. For more information, see “High Availability Disaster Recovery (HADR) synchronization mode” on page 431.

Will you use a peer window?

The peer window feature specifies that the primary and standby databases are to behave as though they are still in peer state for a configured amount of time if the primary loses the HADR connection in peer state. If primary fails in peer or this “disconnected peer” state, the failover to standby will have zero data loss. This feature provides the greatest protection. For more information, see “Setting the hadr_timeout and hadr_peer_window database configuration parameters” on page 493.

How many standbys will you deploy?

With HADR, you can use either single standby mode or multiple standby mode. With multiple standbys, you can achieve both your high availability and disaster recovery objectives with a single technology. For more information, see "HADR multiple standby databases" on page 435.

There are a number of ways that you can use your HADR standby or standbys beyond their HA or DR purpose:

Reads on standby

You can use the reads on standby feature to direct read-only workload to one or more standby databases without affecting the HA or DR responsibility of the standby. This feature can help reduce the workload on the primary without affecting the main responsibility of the standby. For more information on this topic, see "HADR reads on standby feature" on page 458.

Unless you have reads on standby enabled, applications can access the current primary database only. If you have reads on standby enabled, read-only applications can be redirected to the standby. Applications connecting to the standby database do not affect the availability of the standby in the case of a failover.

Delayed replay

You can use delayed replay to specify that a standby database is to remain at an earlier point in time than the primary, in terms of log replay. If data is lost or corrupted on the primary, you can recover this data on the time delayed standby. For more information, see "HADR delayed replay" on page 463.

Rolling updates and upgrades

Using an HADR setup, you can make various types of upgrades and DB2 fix pack updates to your databases without an outage. If you are using multiple standby mode enabled, you can perform an upgrade while at the same time keeping the protection provided by HADR. For more information, see "Performing rolling updates in a DB2 High Availability Disaster Recovery (HADR) environment" on page 466.

HADR might be your best option if most or all data in your database requires protection or if you perform DDL operations that must be automatically replicated on a standby database. However, HADR is only one of several replication solutions that are offered in the DB2 product family. The InfoSphere Federation Server software and the DB2 database system include SQL replication and Q replication solutions that you can also use, in some configurations, to provide high availability. These solutions maintain logically consistent copies of database tables at multiple locations. In addition, they provide flexibility and complex functionality such as support for column and row filtering, data transformation, and updates to any copy of a table. You can also use these solutions in partitioned database environments.

In IBM Data Studio Version 3.1 or later, you can use the task assistant for setting up HADR. Task assistants can guide you through the process of setting options, reviewing the automatically generated commands to perform the task, and running these commands. For more details, see Administering databases with task assistants.

High Availability Disaster Recovery (HADR) synchronization mode

The HADR synchronization mode determines the degree of protection your DB2 High Availability Disaster Recovery (HADR) database solution has against transaction loss. The synchronization mode determines when the primary database server considers a transaction complete, based on the state of the logging on the standby database.

The more strict the synchronization mode configuration parameter value, the more protection your database solution has against transaction data loss, but the slower your transaction processing performance. You must balance the need for protection against transaction loss with the need for performance.

Figure 69 shows the DB2 HADR synchronization modes that are available and also when transactions are considered committed based on the synchronization mode chosen:

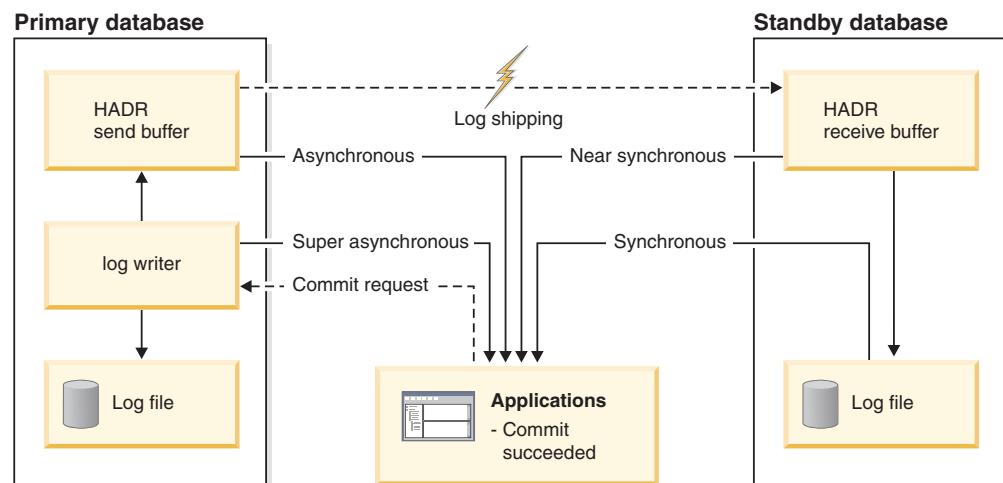


Figure 69. Synchronization modes for high availability and disaster recovery (HADR)

In multiple standby mode, the setting for **hadr_syncmode** does not need to be the same on the primary and standby databases. Whatever setting for **hadr_syncmode** is specified on a standby is considered its *configured synchronization mode*; this setting only has relevance if the standby becomes a primary. Instead, the standby is assigned an *effective synchronization mode*. For any auxiliary standby, the effective synchronization mode is always SUPERASYNC. For the principal standby, the effective synchronization mode is the primary's setting for **hadr_syncmode**. A standby's effective synchronization mode is the value that is displayed by any monitoring interface.

Use the **hadr_syncmode** database configuration parameter to set the synchronization mode. The following values are valid:

SYNC (synchronous)

This mode provides the greatest protection against transaction loss, and using it results in the longest transaction response time among the four modes.

In this mode, log writes are considered successful only when logs have been written to log files on the primary database and when the primary database has received acknowledgement from the standby database that

the logs have also been written to log files on the standby database. The log data is guaranteed to be stored at both sites.

If the standby database crashes before it can replay the log records, the next time it starts it can retrieve and replay them from its local log files. If the primary database fails, a failover to the standby database guarantees that any transaction that has been committed on the primary database has also been committed on the standby database. After the failover operation, when the client reconnects to the new primary database, there can be transactions committed on the new primary database that were never reported as committed to the application on the original primary. This occurs when the primary database fails before it processes an acknowledgement message from the standby database. Client applications should consider querying the database to determine whether any such transactions exist.

If the primary database loses its connection to the standby database, what happens next depends on the configuration of the **hadr_peer_window** database configuration parameter. If **hadr_peer_window** is set to a non-zero time value, then upon losing connection with the standby database the primary database will move into disconnected peer state and continue to wait for acknowledgement from the standby database before committing transactions. If the **hadr_peer_window** database configuration parameter is set to zero, the primary and standby databases are no longer considered to be in peer state and transactions will not be held back waiting for acknowledgement from the standby database. If the failover operation is performed when the databases are not in peer or disconnected peer state, there is no guarantee that all of the transactions committed on the primary database will appear on the standby database.

If the primary database fails when the databases are in peer or disconnected peer state, it can rejoin the HADR pair as a standby database after a failover operation. Because a transaction is not considered to be committed until the primary database receives acknowledgement from the standby database that the logs have also been written to log files on the standby database, the log sequence on the primary will be the same as the log sequence on the standby database. The original primary database (now a standby database) just needs to catch up by replaying the new log records generated on the new primary database since the failover operation.

If the primary database is not in peer state when it fails, its log sequence might be different from the log sequence on the standby database. If a failover operation has to be performed, the log sequence on the primary and standby databases might be different because the standby database starts its own log sequence after the failover. Because some operations cannot be undone (for example, dropping a table), it is not possible to revert the primary database to the point in time when the new log sequence was created. If the log sequences are different and you issue the **START HADR** command with the **AS STANDBY** parameter on the original primary, you will receive a message that the command was successful. However, this message is issued before reintegration is attempted. If reintegration fails, pair validation messages will be issued to the administration log and the diagnostics log on both the primary and the standby. The reintegrated standby will remain the standby, but the primary will reject the standby during pair validation causing the standby database to shut down. If the original primary database successfully rejoins the HADR pair, you can achieve failback of the database by issuing the

TAKEOVER HADR command without specifying the **BY FORCE** parameter. If the original primary database cannot rejoin the HADR pair, you can reinitialize it as a standby database by restoring a backup image of the new primary database.

NEARSYNC (near synchronous)

While this mode has a shorter transaction response time than synchronous mode, it also provides slightly less protection against transaction loss.

In this mode, log writes are considered successful only when the log records have been written to the log files on the primary database and when the primary database has received acknowledgement from the standby system that the logs have also been written to main memory on the standby system. Loss of data occurs only if both sites fail simultaneously and if the target site has not transferred to nonvolatile storage all of the log data that it has received.

If the standby database crashes before it can copy the log records from memory to disk, the log records will be lost on the standby database. Usually, the standby database can get the missing log records from the primary database when the standby database restarts. However, if a failure on the primary database or the network makes retrieval impossible and a failover is required, the log records will never appear on the standby database, and transactions associated with these log records will never appear on the standby database.

If transactions are lost, the new primary database is not identical to the original primary database after a failover operation. Client applications should consider resubmitting these transactions to bring the application state up to date.

If the primary database fails when the primary and standby databases are in peer state, it is possible that the original primary database cannot rejoin the HADR pair as a standby database without being reinitialized using a full restore operation. If the failover involves lost log records (because both the primary and standby databases have failed), the log sequences on the primary and standby databases will be different and attempts to restart the original primary database as a standby database without first performing a restore operation will fail. If the original primary database successfully rejoins the HADR pair, you can achieve failback of the database by issuing the **TAKEOVER HADR** command without specifying the **BY FORCE** parameter. If the original primary database cannot rejoin the HADR pair, you can reinitialize it as a standby database by restoring a backup image of the new primary database.

ASYNC (asynchronous)

Compared with the SYNC and NEARSYNC modes, the ASYNC mode results in shorter transaction response times but might cause greater transaction losses if the primary database fails.

In ASYNC mode, log writes are considered successful only when the log records have been written to the log files on the primary database and have been delivered to the TCP layer of the primary system's host machine. Because the primary system does not wait for acknowledgement from the standby system, transactions might be considered committed when they are still on their way to the standby database.

A failure on the primary database host machine, on the network, or on the standby database can cause log records in transit to be lost. If the primary database is available, the missing log records can be resent to the standby

database when the pair reestablishes a connection. However, if a failover operation is required while there are missing log records, those log records will never reach the standby database, causing the associated transactions to be lost in the failover.

If transactions are lost, the new primary database is not exactly the same as the original primary database after a failover operation. Client applications should consider resubmitting these transactions to bring the application state up to date.

If the primary database fails when the primary and standby databases are in peer state, it is possible that the original primary database will not be able to rejoin the HADR pair as a standby database without being reinitialized using a full restore operation. If the failover involves lost log records, the log sequences on the primary and standby databases will be different, and attempts to restart the original primary database as a standby database will fail. Because there is a greater possibility of log records being lost if a failover occurs in asynchronous mode, there is also a greater possibility that the primary database will not be able to rejoin the HADR pair. If the original primary database successfully rejoins the HADR pair, you can achieve failback of the database by issuing the **TAKEOVER HADR** command without specifying the **BY FORCE** parameters. If the original primary database cannot rejoin the HADR pair, you can reinitialize it as a standby database by restoring a backup image of the new primary database.

SUPERASYNC (super asynchronous)

This mode has the shortest transaction response time but has also the highest probability of transaction losses if the primary system fails. This mode is useful when you do not want transactions to be blocked or experience elongated response times due to network interruptions or congestion.

In this mode, the HADR pair can never be in peer state or disconnected peer state. The log writes are considered successful as soon as the log records have been written to the log files on the primary database. Because the primary database does not wait for acknowledgement from the standby database, transactions are considered committed irrespective of the state of the replication of that transaction.

A failure on the primary database host machine, on the network, or on the standby database can cause log records in transit to be lost. If the primary database is available, the missing log records can be resent to the standby database when the pair reestablishes a connection. However, if a failover operation is required while there are missing log records, those log records will never reach the standby database, causing the associated transactions to be lost in the failover.

If transactions are lost, the new primary database is not exactly the same as the original primary database after a failover operation. Client applications should consider resubmitting these transactions to bring the application state up to date.

Since the transaction commit operations on the primary database are not affected by the relative slowness of the HADR network or the standby HADR server, the log gap between the primary database and the standby database might continue to increase. It is important to monitor the log gap as it is an indirect measure of the potential number of transactions that might be lost should a true disaster occur on the primary system. In

disaster recovery scenarios, any transactions committed during the log gap would not be available to the standby database. Therefore, monitor the log gap by using the **hadr_log_gap** monitor element; if it occurs that the log gap is not acceptable, investigate the network interruptions or the relative speed of the standby database node and take corrective measures to reduce the log gap.

If the primary database fails, it is possible that the original primary database will not be able to rejoin the HADR pair as a standby database without being reinitialized using a full restore operation. If the failover involves lost log records, the log sequences on the primary and standby databases will be different, and attempts to restart the original primary database as a standby database will fail. Because there is a greater probability of log records being lost if a failover occurs in super asynchronous mode, there is also a greater probability that the primary database will not be able to rejoin the HADR pair. If the original primary database successfully rejoins the HADR pair, you can achieve failback of the database by issuing the **TAKEOVER HADR** command without specifying the **BY FORCE** parameter. If the original primary database cannot rejoin the HADR pair, you can reinitialize it as a standby database by restoring a backup image of the new primary database.

HADR multiple standby databases

The high availability disaster recover (HADR) feature supports multiple standby databases. Using multiple standbys, you can have your data in more than two sites, which provides improved data protection with a single technology.

When you deploy the HADR feature in multiple standby mode, you can have up to three standby databases in your setup. You designate one of these databases as the *principal HADR standby database*; any other standby database is an *auxiliary HADR standby database*. Both types of HADR standbys are synchronized with the HADR primary database through a direct TCP/IP connection, both types support reads on standby, and you can configure both types for time-delayed log replay. In addition, you can issue a forced or non-forced takeover on any standby. There are a couple of important distinctions between the principal and auxiliary standbys, however:

- IBM Tivoli System Automation for Multiplatforms (SA MP) automated failover is supported only for the principal standby. You must issue a takeover manually on one of the auxiliary standbys to make one of them the primary. Before issuing a manual takeover, you should disable SA MP.
- All of the HADR sync modes are supported on the principal standby, but the auxiliary standbys can only be in SUPERASYNC mode.

There are a number of benefits to using a multiple HADR standby setup. Instead of employing the HADR feature to achieve your high availability objectives and another technology to achieve your disaster recovery objectives, you can use HADR for both. You can deploy your principal standby in the same location as the primary. If there is an outage on the primary, the principal standby can take over the primary role within your recovery time objectives. You can also deploy auxiliary standbys in a distant location, which provides protection against a widespread disaster that affects both the primary and the principal standby. The distance, and the potential for network delays between the primary and the auxiliaries, has no effect on activity on the primary because the auxiliaries use SUPERASYNC mode. If a disaster affects the primary and principal standby, you can issue a takeover on either of the auxiliaries. You can configure the other

auxiliary standby database to become the new principal standby using the **hadr_target_list** database configuration parameter. However, an auxiliary standby can take over as the primary even if that auxiliary does not have an available standby. For example, if there is an outage on the primary and principal standby, one auxiliary can take over as the primary even if it does not have a corresponding standby. However, if you stop that database after it becomes the new primary, it cannot start again as an HADR primary unless its principal standby is started.

Restrictions for multiple standby databases

There are a number of restrictions that you should be aware of if you are planning to deploy the HADR feature in multiple standby mode.

The restrictions are as follows:

- You can have a maximum of three standby databases: one principal standby and up to two auxiliary standbys.
- Only the principal standby supports all the HADR synchronization modes; all auxiliary standbys will be in SUPERASYNC mode.
- IBM Tivoli System Automation for Multiplatforms (SA MP) support applies only between the primary HADR database and its principal standby.
- The **hadr_target_list** database configuration parameter must be set on all the databases in the multiple standby setup. Each standby must include the primary in its **hadr_target_list** setting.

Initializing HADR in multiple standby mode

Initializing an HADR system in multiple standby mode is similar to single standby mode. The main difference is that you must enable multiple standby mode by setting the **hadr_target_list** database configuration parameter on all the databases in your setup.

About this task

This task covers how to initialize HADR in multiple standby mode. If you want to convert a single standby setup to a multiple standby setup, see “Enabling multiple standby mode on a preexisting HADR setup” on page 438.

Multiple standby mode requires the **hadr_target_list** configuration parameter to be set on all participating databases. This parameter lists the standbys in the scenario when the database becomes a primary. It is required even on a standby. Mutual inclusion is required (that is, if A has B in its target list, B must have A in its target list). This ensures that after a takeover from any standby, the new primary can always keep the old primary as its standby. The first standby that you specify in the target list is designated as the *principal HADR standby database*. Additional standbys are *auxiliary HADR standby databases*. The target list need not always include all participants. As well, there is no requirement for symmetry or reciprocity if there is more than one standby; even if you designate that database A has database B as its principal standby, database B does not have to designate A as its principal standby. Each standby specified in the target list of database A, must also have database A in its target list. Working out the target list for each database is an important step.

As a special case, multiple standby mode can be configured with only one standby. For example, you can configure two databases as primary and standby in multiple standby mode. The behavior is not same as single standby setup because multiple

standby behavior such as automated configuration will be in effect and because standby targets can be added or removed dynamically.

Tip: You can perform steps 2 to 4 in a single update on each database.

Procedure

To initialize HADR in multiple standby mode:

1. Create your standby database or databases by using either a restored backup or split mirror. For instructions on how to do this, see “Initializing a standby database” on page 479 or step 2 of “Initializing high availability disaster recovery (HADR)” on page 477.
 - a. On the primary, issue the following command:

```
BACKUP DB dbname
```
 - b. If the database already exists on a standby instance, drop it first for a clean start. Files from the existing database can interfere with HADR operation. For example, left over log files can lead the standby onto a log chain not compatible with the primary. Issue the following command to drop the database:

```
DROP DB dbname
```
 - c. On each standby instance, issue the following command :

```
RESTORE DB dbname
```
2. On each of the databases, set the **hadr_local_host**, **hadr_local_svc**, **hadr_local_svc**, and **hadr_syncmode** configuration parameters:

```
UPDATE DB CFG FOR dbname USING
    HADR_LOCAL_HOST    hostname
    HADR_LOCAL_SVC     servicename
    HADR_SYNCMODE      syncmode
```
3. Set the **hadr_target_list** configuration parameter on all of the standbys and the primary:

```
UPDATE DB CFG FOR dbname USING
    HADR_TARGET_LIST   principalhostname:principalservicename|
                        auxhostname1:auxservicename1|auxhostname2:auxservicename2
```
4. Optional: On all the databases, set the **hadr_remote_host**, **hadr_remote_svc**, and **hadr_remote_inst** configuration parameters.

This step is not required because these values are automatically set if you do not set them and are automatically reset if you set them incorrectly. However, explicitly setting them to the correct values makes correct values available immediately. These values are helpful for the IBM Tivoli System Automation for Multiplatforms (SA MP) software, which might require the **hadr_remote_inst** value to construct the resource name.

 - On the primary, set the parameters to the corresponding values on the principal standby by issuing the following command:

```
UPDATE DB CFG FOR dbname USING
    HADR_REMOTE_HOST   principalhostname
    HADR_REMOTE_SVC    principalservicename
    HADR_REMOTE_INST   principalinstname
```
 - On each standby, set the parameters to the corresponding values on the primary by issuing the following command:

```
UPDATE DB CFG FOR dbname USING
    HADR_REMOTE_HOST   primaryhostname
    HADR_REMOTE_SVC    primaryservicename
    HADR_REMOTE_INST   primaryinstname
```
5. Connect to each standby instance.

6. On the standby instance, issue the **START HADR** command with the **AS STANDBY** parameter:

```
START HADR ON DB dbname AS STANDBY
```

7. Connect to the primary instance.

8. On the primary instance, issue the **START HADR** command with the **AS PRIMARY** parameter:

```
START HADR ON DB dbname AS PRIMARY
```

Results

The standby databases start in local catchup state, in which locally available log files are read and replayed. After all local logs have been replayed, the databases enter remote catchup pending state. After the primary starts, the standbys enter remote catchup state, in which log pages are received from the primary and replayed. After all of the log files that are on the disk of the primary database have been replayed on the standbys, what happens depends on the type of what happens next depends on the type of synchronization mode. A principal standby in SUPERASYNC and any auxiliary standby will stay in remote catchup mode. A principal standby with a SYNC, NEARSYNC, or ASYNC mode will enter peer mode.

Enabling multiple standby mode on a preexisting HADR setup

Initializing an HADR system in multiple standby mode is similar to single standby mode. The main difference is that you must enable multiple standby mode by setting the **hadr_target_list** database configuration parameter on all the databases in your setup.

Before you begin

- Determine the host name or host IP address (to be used for the **hadr_local_host** setting), service name or port number (to be used for the **hadr_local_svc** setting) of all participating databases.
- Determine the target list for each database.
- Determine the synchronization mode and peer window for each database's principal standby in the event that the database becomes the primary.
- Determine the setting for the **hadr_timeout** configuration parameter; this parameter must have the same setting on all databases.
- Determine if there is sufficient network bandwidth between the primary and each standby. Upgrade if necessary.
- Determine if the primary network interface can support outgoing data flow of the additional standbys. Upgrade if needed.

About this task

Multiple standby mode requires the **hadr_target_list** configuration parameter to be set on all participating databases. This parameter lists the standbys in the scenario when the database becomes a primary. It is required even on a standby. Mutual inclusion is required (that is, if A has B in its target list, B must have A in its target list). This ensures that after a takeover from any standby, the new primary can always keep the old primary as its standby. The first standby that you specify in the target list is designated as the *principal HADR standby database*. Additional standbys are *auxiliary HADR standby databases*. The target list need not always include all participants. As well, there is no requirement for symmetry or reciprocity if there is more than one standby; even if you designate that database A

has database B as its principal standby, database B does not have to designate A as its principal standby. Each standby specified in the target list of database A, must also have database A in its target list. Working out the target list for each database is an important step.

As a special case, multiple standby mode can be configured with only one standby. For example, you can configure two databases as primary and standby in multiple standby mode. The behavior is not same as single standby setup because multiple standby behavior such as automated configuration will be in effect and because standby targets can be added or removed dynamically.

In this task, you first create and configure the new standbys only. By keeping the original configuration until the final steps, you can keep your primary-standby pair functioning for as long as possible. If you change the original standby's configuration too early, you can break the old HADR pair if the standby is deactivated and reactivated unintentionally to pick up the new configuration.

Procedure

To enable HADR in multiple standby mode:

1. Create any additional standby databases using either a restored backup or split mirror. For instructions on how to do this, see "Initializing a standby database" on page 479 or step 2 of "Initializing high availability disaster recovery (HADR)" on page 477.
 - On the primary:
`DB2 BACKUP DB dbname`
 - On the standbys:
`DB2 RESTORE DB dbname`
2. Configure each of the new standby databases as follows:
 - a. Set the **hadr_local_host** and **hadr_local_svc** to the TCP address used by the HADR connection.
 - b. Set the **hadr_remote_host**, **hadr_remote_svc**, **hadr_remote_inst** configuration parameters to point to the primary database.
 - c. Set the **hadr_timeout** configuration, with the same setting on all of the databases.
 - d. Set the **hadr_target_list** configuration parameter, as previously planned.
 - e. Set the **hadr_syncmode** and **hadr_peer_window** configuration parameters for the principal standby in case this database becomes the primary.
 - f. Set any other HADR-specific parameters such as **hadr_spool_limit** or **hadr_replay_delay**, depending on your desired setup.
3. Reconfigure the original standby by following the same instructions as in Step 2.
4. Reconfigure the primary as follows:
 - a. Set the **hadr_local_host** and **hadr_local_svc** to the TCP address used by the HADR connection. You might need to make an update if you are using a new network interface card (NIC) to support higher network bandwidth to accommodate more standbys.
 - b. Set the **hadr_remote_host**, **hadr_remote_svc**, **hadr_remote_inst** configuration parameters to point to the principal standby database.
 - c. Set the **hadr_timeout** configuration, with the same setting as on all of the standby databases.
 - d. Set the **hadr_target_list** configuration parameter, as previously planned.

- e. Set the **hadr_syncmode** and **hadr_peer_window** configuration parameters, which the principal standby will use.
 - f. Set any other HADR-specific parameters such as **hadr_spool_limit** or **hadr_replay_delay**, depending on your desired setup.
5. Stop HADR on the primary.
`STOP HADR ON DB dbname`

If a primary (in single standby mode) is still running when a new HADR standby is started, the standby is found incompatible and shut down when it attempts to connect to the primary.
 6. Deactivate and then reactivate the original standby to pick up the new configuration.
 7. Connect to each new standby instance and issue the **START HADR** command with the **AS STANDBY** option.
`START HADR ON DB dbname AS STANDBY`
 8. Stop HADR on the primary.
`START HADR ON DB dbname AS PRIMARY`

Results

All of the standbys should connect to the primary within seconds. You can monitor their status using the **db2pd** command with the **-hadr** option or the **MON_GET_HADR** table function.

Modifications to a multiple standby database setup

After your multiple HADR standby setup is up and running, you might want to make additional changes, such as adding or removing auxiliary standby databases or changing the principal standby database designation. You can make these kinds of modifications without causing an outage on your primary database.

Adding auxiliary standbys

There are a few reasons why you might want to add an auxiliary standby:

- To deploy an additional standby for processing read-only workloads
- To deploy an additional standby for time-delayed replay
- To deploy an additional standby for disaster recovery purposes
- To add a standby that was a part of a previously active HADR deployment but was *orphaned* because the **hadr_target_list** configuration parameter for the new primary does not specify that standby

You can add an auxiliary standby only if your HADR deployment is in multiple standby mode. That is, the **hadr_target_list** configuration parameter must already be set to at least one standby.

To add an auxiliary standby to your HADR deployment, update the target list of the primary with the host and port information from the standby. This information corresponds to the settings for the **hadr_local_host** and **hadr_local_svc** parameters on the standby. You must also add the host and port information for the primary to the target list of the new standby.

Tip: Although it is not required, a best practice is to also add the host and port information for the new standby to the target lists of the other standbys in the deployment. You should also specify the host and port information for those

standbys in the target list of the new standby. If you do not make these additional updates and one of the other standbys takes over as the new primary, the new standby is rejected as a standby target and is shut down.

Removing auxiliary standbys

The only standbys that you can remove dynamically are auxiliary standbys. If you dynamically remove an auxiliary standby from your multiple standby deployment, there is no effect on normal HADR operations on the primary and the principal standby. To remove an auxiliary standby, issue the **STOP HADR** command on the standby; afterward, you can remove it from the target lists of the primary and any other standby.

Changing the principal standby

You can change the principal standby only if you first stop HADR on the primary database; this does not cause an outage, because you do not have to deactivate the primary.

To change the principal standby, you must stop HADR on the primary database. Then, update the target list of the primary database to list the new principal standby first. If the new principal standby is not already a standby, add the primary database's address to its target list, configure the other HADR parameters, and activate the standby. If it is already a standby, no action is needed.

Tip: Although it is not required, it is a best practice to also add the host and port information for the new principal standby to the target list of the other standby in the deployment. You should also specify the host and port information for that standby in the target list of the new principal standby. If you do not make these additional updates and either one of the standbys takes over as the new primary, the other standby is rejected as a standby target and is shut down.

Database configuration for multiple HADR standby databases

There are a number of considerations for database configuration in a multiple HADR standby setup.

Automatic reconfiguration of HADR parameters

Reconfiguration after HADR starts

In multiple standby mode, the configuration parameters that identify the primary database for the standbys and identify the principal standby for the primary are automatically reset when HADR starts if you did not correctly set them; however, an initial non-NULL value is required. This behavior applies to the following configuration parameters:

- **hadr_remote_host**
- **hadr_remote_inst**
- **hadr_remote_svc**

Tip: Even though this automatic reconfiguration occurs, you should always try to set the correct initial values because that reconfiguration might not take effect until a connection is made between a standby and its primary. In some HADR deployments, those initial values might be needed. For example, if you are using the IBM Tivoli System Automation for Multiplatforms software, the value for the **hadr_remote_inst** configuration parameter is needed to construct a resource name.

Note: If the **DB2_HADR_NO_IP_CHECK** registry variable is set to ON, the **hadr_remote_host** and **hadr_remote_svc** are not automatically updated.

Reconfiguration is predicated on the values of the **hadr_target_list** configuration parameter being correct; if anything is wrong in a target list entry, you must correct it manually.

On the primary, the reconfiguration occurs in the following manner:

- If the values for the **hadr_remote_host** and **hadr_remote_svc** configuration parameters do not match the *host:port* pair that is the first entry of the **hadr_target_list** configuration parameter (namely, the principal standby), the **hadr_remote_host** and **hadr_remote_svc** configuration parameters are updated with the values from the target list.
- If the value for the **hadr_remote_inst** configuration parameter does not match the instance name of the principal standby, the correct instance name is copied to the **hadr_remote_inst** configuration parameter for the primary after the principal standby connects to it.

On a standby database, the reconfiguration occurs in the following manner:

- When a standby starts, it attempts to connect to the database that you specified for its **hadr_remote_host**, **hadr_remote_inst**, and **hadr_remote_svc** configuration parameters.
- If the standby cannot connect to the primary, it waits for the primary to connect to it.
- The primary attempts to connect to its standbys using addresses listed in its **hadr_target_list** parameter. After the primary connects to a standby, the **hadr_remote_host**, **hadr_remote_inst**, and **hadr_remote_svc** configuration parameters for the standby are updated with the correct values for the primary.

Reconfiguration during and after a takeover

In a non-forced takeover, the values for the **hadr_remote_host**, **hadr_remote_inst**, and **hadr_remote_svc** configuration parameters on the new primary are automatically updated to its principal standby, and these parameters on the standbys listed in the new primary's **hadr_target_list** are automatically updated to point to the new primary. Any database that is not listed in the new primary's **hadr_target_list** is not updated. Those databases continue to attempt to connect to the old primary and get rejected because the old primary is now a standby. The old primary is guaranteed to be in the new primary's target list because of the requirement of mutual inclusion in the target list.

In a forced takeover, automatic update on the new primary and its standbys (excluding the old primary) work the same way as non-forced takeover. However, automatic update on the old primary does not happen until it is shut down and restarted as a standby for reintegration.

Any database that is not online during the takeover will be automatically reconfigured after it starts. Automatic reconfiguration might not take effect immediately on startup, because it relies on the new primary to periodically contact the standby. On startup, a standby might attempt to connect to the old primary and follow the log stream of the old primary, causing it to diverge from the new primary's log stream and, making that standby unable to pair with the new primary. As a result, you must shut down the old primary before takeover to avoid that kind of *split brain* scenario.

Lack of standby control of the synchronization mode and peer window

In multiple standby mode, only the settings of the **hadr_syncmode** and **hadr_peer_window** configuration parameters of the current primary are relevant. The standby databases either have the settings for those parameters defined by the primary, in the case of the principal standby, or by their role as an auxiliary standby.

Synchronization mode

In multiple standby mode, the setting for the **hadr_syncmode** configuration parameter do not have to be the same on the primary and standby databases. Whatever setting you specify for the **hadr_syncmode** configuration parameter on a standby is considered its *configured synchronization mode*; this setting has relevance only if the standby becomes a primary. The standby is assigned an *effective synchronization mode*. For any auxiliary standby, the effective synchronization mode is always SUPERASYNC. For the principal standby, the effective synchronization mode is the setting for the **hadr_syncmode** configuration parameter for the primary. For a standby, the monitoring interfaces display the effective synchronization mode as the synchronization mode.

Peer window

In multiple standby mode, the setting for the **hadr_peer_window** configuration parameter does not have to be the same on the primary and standby databases. In fact, any setting for the **hadr_peer_window** configuration parameter on the auxiliary standbys is ignored because peer window functionality is incompatible with SUPERASYNC mode. The principal standby uses the peer window setting of the primary, which is applicable only if the value of the **hadr_syncmode** configuration parameter for the standby is SYNC or NEARSYNC, just as with single standby mode.

Rolling upgrades in HADR multiple standby mode

As with HADR single standby mode, you can use a rolling upgrade. The crucial difference is that with multiple standbys you can use this procedure while maintaining HADR protection by keeping a primary and a standby active.

There is always a primary to provide database service and this primary always has at least one standby providing HA and DR protection.

With multiple standbys, you should perform the update or upgrade on all of the standbys before doing so on the primary. This is particularly important if you are updating the fixpack level because HADR does not allow the primary to be at a higher fixpack level than the standby.

The procedure is essentially the same as with single standby mode, except you should perform the upgrade on one database at a time and starting with an auxiliary standby. For example, consider the following HADR setup:

- host1 is the primary
- host2 is the principal standby
- host 3 is the auxiliary standby

For this setup, perform the rolling upgrade or update according to the following sequence:

1. Deactivate host3, make the required changes, activate host3, and start HADR on host3 (as a standby).
2. After host3 is caught up in log replay, deactivate host2, make the required changes, activate host2, and start HADR on host2 (as a standby).
3. After host2 is caught up in log replay and in peer state with host1, issue a takeover on host2.
4. Deactivate host1, make the required changes, activate host1, and start HADR on host1 (as a standby).
5. After host1 is in peer state with host 2, issue a takeover on host1 so that it becomes the primary again and host2 becomes the principal standby again.

High availability disaster recovery (HADR) monitoring in multiple standby mode

HADR multiple standby mode supports the same monitoring interfaces as in single standby mode; however, you should only use the **db2pd** command and the MON_GET_HADR table function because other monitoring interfaces do not give a complete view of all of the standbys.

The information returned by the monitoring interface depends on where it is issued. Monitoring on a standby returns information about that standby and the primary only; no information is provided about any other standbys. Monitoring on the primary returns information about all of the standbys if you are using the **db2pd** command or the MON_GET_HADR table function. Even standbys that are not connected, but are configured in the primary's **hadr_target_list** configuration parameter are displayed. Other interfaces like the **GET SNAPSHOT FOR DATABASE** command report the primary and the principal standby only.

The **db2pd** command and the MON_GET_HADR table function return essentially the same information, but the **db2pd** command does not require reads on standby to be enabled (for reporting from a standby). As well, the **db2pd** command is preferred during takeover because there could be a time window where neither the primary nor the standby allows client connections.

db2pd command

In the following example, the DBA issues the **db2pd** command on a primary database with three standbys. Three sets of data are returned, with each representing a primary-standby log shipping channel. The HADR_ROLE field represents the role of the database to which **db2pd** is issued, so it is listed as PRIMARY in all sets. The HADR_STATE for the two auxiliary standbys (hostS2 and hostS3) is REMOTE_CATCHUP because they automatically run in SUPERASYNC mode (which is also reflected in the **db2pd** output) regardless of their configured setting for **hadr_syncmode**. The STANDBY_ID differentiates the standbys. It is system generated and the ID-to-standby mapping can change from query to query; however, the ID "1" is always assigned to the principal standby.

Note: Fields not relevant to current status might be omitted in the output. For example, in the following output, information about the replay-only window (like start time and transaction count) is not included because the replay-only window is not active.

```
db2pd -db hadr_db -hadr
```

```
Database Member 0 -- Database hadr_db -- Active -- Up 0 days 00:23:17 --
Date 06/08/2011 13:57:23
```

```

        HADR_ROLE = PRIMARY
        REPLAY_TYPE = PHYSICAL
        HADR_SYNCMODE = SYNC
        STANDBY_ID = 1
        LOG_STREAM_ID = 0
        HADR_STATE = PEER
PRIMARY_MEMBER_HOST = hostP.ibm.com
PRIMARY_INSTANCE = db2inst1
PRIMARY_MEMBER = 0
STANDBY_MEMBER_HOST = hostS1.ibm.com
STANDBY_INSTANCE = db2inst2
STANDBY_MEMBER = 0
HADR_CONNECT_STATUS = CONNECTED
HADR_CONNECT_STATUS_TIME = 06/08/2011 13:38:10.199479 (1307565490)
HEARTBEAT_INTERVAL(seconds) = 30
HADR_TIMEOUT(seconds) = 120
TIME_SINCE_LAST_RECV(seconds) = 3
PEER_WAIT_LIMIT(seconds) = 0
LOG_HADR_WAIT_CUR(seconds) = 0.000
LOG_HADR_WAIT_RECENT_AVG(seconds) = 0.006298
LOG_HADR_WAIT_ACCUMULATED(seconds) = 0.516
LOG_HADR_WAIT_COUNT = 82
SOCK_SEND_BUF_REQUESTED,ACTUAL(bytes) = 0, 50772
SOCK_RECV_BUF_REQUESTED,ACTUAL(bytes) = 0, 87616
PRIMARY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
STANDBY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
HADR_LOG_GAP(bytes) = 0
STANDBY_REPLY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
STANDBY_RECV_REPLY_GAP(bytes) = 0
PRIMARY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
STANDBY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
STANDBY_REPLY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
STANDBY_RECV_BUF_SIZE(pages) = 16
STANDBY_RECV_BUF_PERCENT = 0
STANDBY_SPOOL_LIMIT(pages) = 0
PEER_WINDOW(seconds) = 0
READS_ON_STANDBY_ENABLED = Y
STANDBY_REPLY_ONLY_WINDOW_ACTIVE = N

        HADR_ROLE = PRIMARY
        REPLAY_TYPE = PHYSICAL
        HADR_SYNCMODE = SUPERASYNC
        STANDBY_ID = 2
        LOG_STREAM_ID = 0
        HADR_STATE = REMOTE_CATCHUP
PRIMARY_MEMBER_HOST = hostP.ibm.com
PRIMARY_INSTANCE = db2inst1
PRIMARY_MEMBER = 0
STANDBY_MEMBER_HOST = hostS2.ibm.com
STANDBY_INSTANCE = db2inst3t
STANDBY_MEMBER = 0
HADR_CONNECT_STATUS = CONNECTED
HADR_CONNECT_STATUS_TIME = 06/08/2011 13:35:51.724447 (1307565351)
HEARTBEAT_INTERVAL(seconds) = 30
HADR_TIMEOUT(seconds) = 120
TIME_SINCE_LAST_RECV(seconds) = 16
PEER_WAIT_LIMIT(seconds) = 0
LOG_HADR_WAIT_CUR(seconds) = 0.000
LOG_HADR_WAIT_RECENT_AVG(seconds) = 0.006298
LOG_HADR_WAIT_ACCUMULATED(seconds) = 0.516
LOG_HADR_WAIT_COUNT = 82
SOCK_SEND_BUF_REQUESTED,ACTUAL(bytes) = 0, 16384
SOCK_RECV_BUF_REQUESTED,ACTUAL(bytes) = 0, 87380
PRIMARY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
STANDBY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
HADR_LOG_GAP(bytes) = 0
STANDBY_REPLY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
STANDBY_RECV_REPLY_GAP(bytes) = 0
PRIMARY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
STANDBY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
STANDBY_REPLY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)

```

```

STANDBY_RECV_BUF_SIZE(pages) = 16
    STANDBY_RECV_BUF_PERCENT = 0
    STANDBY_SPOOL_LIMIT(pages) = 0
        PEER_WINDOW(seconds) = 0
        READS_ON_STANDBY_ENABLED = Y

        HADR_ROLE = PRIMARY
        REPLAY_TYPE = PHYSICAL
        HADR_SYNCMODE = SUPERASYNC
        STANDBY_ID = 3
        LOG_STREAM_ID = 0
            HADR_STATE = REMOTE_CATCHUP
        PRIMARY_MEMBER_HOST = hostP.ibm.com
        PRIMARY_INSTANCE = db2inst1
        PRIMARY_MEMBER = 0
        STANDBY_MEMBER_HOST = hostS3.ibm.com
        STANDBY_INSTANCE = db2inst3
        STANDBY_MEMBER = 0
        HADR_CONNECT_STATUS = CONNECTED
        HADR_CONNECT_STATUS_TIME = 06/08/2011 13:46:51.561873 (1307566011)
        HEARTBEAT_INTERVAL(seconds) = 30
            HADR_TIMEOUT(seconds) = 120
        TIME_SINCE_LAST_RECV(seconds) = 6
            PEER_WAIT_LIMIT(seconds) = 0
        LOG_HADR_WAIT_CUR(seconds) = 0.000
        LOG_HADR_WAIT_RECENT_AVG(seconds) = 0.006298
        LOG_HADR_WAIT_ACCUMULATED(seconds) = 0.516
            LOG_HADR_WAIT_COUNT = 82
        SOCK_SEND_BUF_REQUESTED,ACTUAL(bytes) = 0, 16384
        SOCK_RECV_BUF_REQUESTED,ACTUAL(bytes) = 0, 87380
            PRIMARY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
            STANDBY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
            HADR_LOG_GAP(bytes) = 0
        STANDBY_REPLY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
        STANDBY_RECV_REPLY_GAP(bytes) = 0
            PRIMARY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
            STANDBY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
            STANDBY_REPLY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
        STANDBY_RECV_BUF_SIZE(pages) = 16
        STANDBY_RECV_BUF_PERCENT = 0
        STANDBY_SPOOL_LIMIT(pages) = 0
        PEER_WINDOW(seconds) = 0
        READS_ON_STANDBY_ENABLED = N

```

MON_GET_HADR table function

In the following example, the DBA calls the **MON_GET_HADR** table function on the primary database with three standbys. Three rows are returned. Each row represents a primary-standby log shipping channel. The HADR_ROLE column represents the role of the database to which the query is issued. Therefore it is PRIMARY on all rows. The HADR_STATE for the two auxiliary standbys (hostS2 and hostS3) is REMOTE_CATCHUP because they automatically run in SUPERASYNC mode regardless of their configured setting for **hadr_syncmode**.

```

db2 "select HADR_ROLE, STANDBY_ID, HADR_STATE, varchar(PRIMARY_MEMBER_HOST,20)
as PRIMARY_MEMBER_HOST, varchar(STANDBY_MEMBER_HOST,20)
as STANDBY_MEMBER_HOST from table (mon_get_hadr(NULL))"

HADR_ROLE  STANDBY_ID  HADR_STATE      PRIMARY_MEMBER_HOST  STANDBY_MEMBER_HOST
-----  -----  -----  -----
PRIMARY      1  PEER          hostP.ibm.com   hostS1.ibm.com
PRIMARY      2  REMOTE_CATCHUP  hostP.ibm.com   hostS2.ibm.com
PRIMARY      3  REMOTE_CATCHUP  hostP.ibm.com   hostS3.ibm.com

```

3 record(s) selected.

Takeover in HADR multiple standby mode

When an HADR standby database takes over as the primary database in a multiple standby environment, there are a number of important differences from single standby mode.

With HADR, there are two types of takeover: *role switch* and *failover*. Role switch, sometimes called graceful takeover or non-forced takeover, can be performed only when the primary is available and it switches the role of primary and standby. Failover, or forced takeover, can be performed when the primary is not available. It is commonly used in primary failure cases to make the standby the new primary. The old primary remains in primary role in a forced takeover. Both types of takeover are supported in multiple standby mode, and any of the standby databases can take over as the primary. A crucial thing to remember, though, is that if a standby is not included in the new primary's target list, it is considered to be orphaned and cannot connect to the new primary.

In a takeover, DB2 automatically makes a number of configuration changes for you so that the standbys listed in new primary's target list can connect to the new primary. The **hadr_remote_host**, **hadr_remote_svc**, and **hadr_remote_inst** configuration parameters are updated on the new primary and listed standbys in the following way:

- **On the new primary:** They refer to the principal standby (the first database listed in the new primary's target list).
- **On the standbys:** They refer to the new primary. When an old primary is reintegrated to become standby, the **START HADR AS STANDBY** command first converts it to a standby. Thus it can also be automatically redirected to the new primary if it is listed in the target list of the new primary.

Note: Orphaned standbys are not automatically updated in this way. If you want them to join as standbys, you need to ensure they are in the new primary's target list and that they include the new primary in their target lists.

Role switch

Just as in single standby mode, role switch in multiple standby mode guarantees no data is lost between the old primary and new primary. Other standbys configured in the new primary's **hadr_target_list** configuration parameter are automatically redirected to the new primary and continue receiving logs. If you are issuing the **TAKEOVER HADR** command on an auxiliary standby and you have IBM Tivoli System Automation for Multiplatforms (SA MP) configured, you must ensure that you disable SA MP before attempting the takeover. You cannot fallback the primary role to the auxiliary primary if SA MP is enabled.

Failover

Just as in single standby mode, if a failover results in any data loss in multiple standby mode (meaning that the new primary does not have all of the data of the old primary), the old and new primary's log streams diverge and the old primary has to be reinitialized. For the other standbys, if a standby received logs from the old primary beyond the diverge point, it has to be reinitialized. Otherwise, it can connect to the new primary and continue log shipping and replay. As a result, it is very important that you check the log positions of all of the standbys and choose the standby with the most data as the failover target. You can query this information using the **db2pd** command or the **MON_GET_HADR** table function.

Note: Successful automatic reconfiguration of a standby's `hadr_remote_host`, `hadr_remote_svc`, and `hadr_remote_inst` configuration parameters to point to the new primary does not mean the standby will be accepted to pair with the new primary. It only allows the standby to make a TCP connection to the primary. Upon connection, if DB2 determines that the two databases have diverging log streams, the pairing request will be rejected and the connection closed.

Scenario: Deploying an HADR multiple standby database setup

This scenario describes the planning, configuring, and deploying of an HADR setup for a bank called ExampleBANK. The setup has three standby databases: one principal standby and two auxiliary standbys.

Background

Because banking is a 24x7 business, high availability is crucial to ExampleBANK's technology strategy. In addition, ExampleBANK experienced a close call with a hurricane hitting City A, where its head office is located, so the bank also requires a disaster recovery strategy. High availability disaster recovery (HADR) offers a solution that can help the bank achieve both of these goals with a single technology: HADR multiple standby databases.

ExampleBANK considers the following requirements essential for its HADR solution:

An aggressive recovery time objective

As a bank that offers 24-hour online service, ExampleBANK wants to minimize the time that applications cannot connect to their database.

An aggressive recovery point objective

ExampleBANK cannot tolerate data loss, so the RPO should be as close to 0 as possible.

Near-zero planned downtime

ExampleBANK's database should be available as much as possible, even through planned activities such as upgrades and maintenance.

Data protection through geographic dispersion

As part of its compliance standards, ExampleBANK wants the capability to recover operations at a remote location.

Easy deployment and management

ExampleBANK's overburdened IT department wants a solution that is relatively simple to configure and that has automation capabilities.

As the following scenarios illustrate, using the HADR feature in multiple standby mode helps ExampleBANK meet all these requirements.

Planning for a multiple standby setup

ExampleBANK wants to have both high availability and disaster recovery protection from its HADR setup, so the bank decides to use the maximum number of standbys: three. To achieve the RTO, the bank must have a standby that is in close synchronization with the primary (a standby that uses SYNC or NEARSYNC mode) and is collocated with the primary. It makes the most sense to have this standby be the principal standby because only that standby supports all

synchronization modes. Both the primary and the principal standby are located in ExampleBANK's head office in City A and are connected by a LAN.

In addition, to protect the bank's data from being lost because of a disaster, the ExampleBANK DBA chooses to set up two standbys in the bank's regional office in City B. The regional office is connected to the head office in City A by a WAN. The distance between the two cities will not affect the primary because the standbys are auxiliary standbys, which automatically run in SUPERASYNC mode. The DBA can provide additional justification for the costs of these additional databases by setting up one of them to use the reads on standby feature and the other to use the time-delayed replay feature. Also, these standbys can help maintain database availability through a rolling update or maintenance scenario, preventing the loss of HADR protection.

Configuring a multiple standby setup

The ExampleBANK DBA takes a backup of the intended primary database, HADR_DB:

```
DB2 BACKUP DB hadr_db TO backup_dir
```

The DBA then restores the backup onto each of the intended standby hosts by issuing the following command:

```
DB2 RESTORE DB hadr_db FROM backup_dir
```

Tip: For more information about options for creating a standby, see "Initializing a standby database" on page 479.

For the initial setup, the ExampleBANK DBA decides that most of the default configuration settings are sufficient. However, as in a regular HADR setup, the following database configuration parameters must be explicitly set:

- **hadr_local_host**
- **hadr_local_svc**
- **hadr_remote_host**
- **hadr_remote_inst**
- **hadr_remote_svc**

To obtain the correct values for those configuration parameters, the DBA determines the host name, port number, and instance name of the four databases that will be in the HADR setup:

Table 27. Host name, port number, and instance name for databases

Intended role	Host name	Port number	Instance name
Primary	host1	10	dbinst1
Principal standby	host2	40	dbinst2
Auxiliary standby	host3	41	dbinst3
Auxiliary standby	host4.ibm.com	42	dbinst4

On the primary, the settings for the **hadr_remote_host**, **hadr_remote_inst**, and **hadr_remote_svc** configuration parameters correspond to the host name, instance name, and port number of the principal standby. On the standbys, the values of these configuration parameters correspond to the host name, port number, and instance name of the primary. In addition, the DBA uses the host name and port values to set the **hadr_target_list** configuration parameter on all the databases. Also, although it is not required, the DBA adds the information about all the

standbys in the setup to the target list of each of the other standbys. For more information about this topic, see “Database configuration for high availability disaster recovery (HADR)” on page 485.

As mentioned earlier, the bank wants the closest possible synchronization between the primary and principal standby, so the DBA sets the **hadr_syncmode** parameter on the primary to SYNC. Although the principal standby will automatically have its effective synchronization mode set to SYNC after it connects to the primary, the DBA still sets the **hadr_syncmode** parameter to SYNC on the principal standby. The reason is that if the principal standby switches role with the primary, the synchronization mode for the new primary and principal standby pair will also be SYNC.

The DBA decides to specify host2, which is in a different city from the auxiliary standbys, as the principal standbys for the auxiliary standbys. If one of the auxiliaries becomes the primary, SUPERASYNC would be a good synchronization mode between the primary and the remotely located host2. Thus DBA sets the **hadr_syncmode** parameter on the auxiliary standbys to SUPERASYNC, although the auxiliary standbys will automatically have their effective synchronization modes set to SUPERASYNC after they connect to the primary. For more information about this topic, see “High Availability Disaster Recovery (HADR) synchronization mode” on page 431.

Finally, the DBA has read about the new HADR delayed replay feature, which can be used to intentionally keep a standby database at a point in time that is earlier than the primary by delaying replay of logs. The DBA decides that enabling this feature would improve ExampleBANK's data protection against errant transactions on the primary. The DBA chooses host4, an auxiliary standby, for this feature, and makes a note that this feature must be disabled before host4 can take over as the primary database. For more information about this topic, see “HADR delayed replay” on page 463.

The DBA issues the following commands to update the configuration parameters on each of the databases:

- On host1 (the primary):

```
DB2 "UPDATE DB CFG FOR hadr_db USING
      HADR_TARGET_LIST host2:40|host3:41|host4:42
      HADR_REMOTE_HOST host2
      HADR_REMOTE_SVC 40
      HADR_LOCAL_HOST host1
      HADR_LOCAL_SVC 10
      HADR_SYNCMODE sync
      HADR_REMOTE_INST db2inst2"
```

- On host2 (the principal standby):

```
DB2 "UPDATE DB CFG FOR hadr_db USING
      HADR_TARGET_LIST host1:10|host3:41|host4:42
      HADR_REMOTE_HOST host1
      HADR_REMOTE_SVC 10
      HADR_LOCAL_HOST host2
      HADR_LOCAL_SVC 40
      HADR_SYNCMODE sync
      HADR_REMOTE_INST db2inst1"
```

- On host3 (an auxiliary standby):

```
DB2 "UPDATE DB CFG FOR hadr_db USING
      HADR_TARGET_LIST host2:40|host1:10|host4:42
      HADR_REMOTE_HOST host1
      HADR_REMOTE_SVC 10
```

```

HADR_LOCAL_HOST  host3
HADR_LOCAL_SVC   41
HADR_SYNCMODE    superasync
HADR_REMOTE_INST db2inst1"

```

- On host4 (an auxiliary standby):

```

DB2 "UPDATE DB CFG FOR hadr_db USING
      HADR_TARGET_LIST host2.:40|host1:10|host3:41
      HADR_REMOTE_HOST host2
      HADR_REMOTE_SVC  10
      HADR_LOCAL_HOST  host4
      HADR_LOCAL_SVC   42
      HADR_SYNCMODE    superasync
      HADR_REMOTE_INST db2inst1
      HADR_REPLAY_DELAY 86400"

```

Finally, the ExampleBANK DBA wants to enable the HADR reads on standby feature for the following reasons:

- To make online changes to some of the HADR configuration parameters on the standbys
- To call the MON_GET_HADR table function on the standbys
- To divert some of the read-only workload from the primary

The DBA updates the registry variables on the standby databases by issuing the following commands on each of host2, host3, and host4:

```

DB2SET DB2_HADR_ROS=ON
DB2SET DB2_STANDBY_ISO=UR

```

Starting the HADR databases

The DBA starts the standby databases first, by issuing the following command on each of host2, host3, and host 4:

```
DB2 START HADR ON DB hadr_db AS STANDBY
```

Next, the DBA starts HADR on the primary database, on host1:

```
DB2 START HADR ON DB hadr_db AS PRIMARY
```

To verify that HADR is up and running, the DBA queries the status of the databases from the primary on host1 by issuing the **db2pd** command, which returns information about all of the standbys:

```
db2pd -db hadr_db -hadr
```

```

Database Member 0 -- Database hadr_db -- Active -- Up 0 days 00:23:17 --
Date 06/08/2011 13:57:23

```

```

          HADR_ROLE = PRIMARY
          REPLAY_TYPE = PHYSICAL
          HADR_SYNCMODE = SYNC
          STANDBY_ID = 1
          LOG_STREAM_ID = 0
          HADR_STATE = PEER
          PRIMARY_MEMBER_HOST = host1
          PRIMARY_INSTANCE = db2inst1
          PRIMARY_MEMBER = 0
          STANDBY_MEMBER_HOST = host2
          STANDBY_INSTANCE = db2inst2
          STANDBY_MEMBER = 0
          HADR_CONNECT_STATUS = CONNECTED
          HADR_CONNECT_STATUS_TIME = 06/08/2011 13:38:10.199479 (1307565490)
          HEARTBEAT_INTERVAL(seconds) = 30
          HADR_TIMEOUT(seconds) = 120
          TIME_SINCE_LAST_RECV(seconds) = 3

```

```

        PEER_WAIT_LIMIT(seconds) = 0
        LOG_HADR_WAIT_CUR(seconds) = 0.000
        LOG_HADR_WAIT_RECENT_AVG(seconds) = 0.006298
        LOG_HADR_WAIT_ACCUMULATED(seconds) = 0.516
            LOG_HADR_WAIT_COUNT = 82
        SOCK_SEND_BUF_REQUESTED,ACTUAL(bytes) = 0, 50772
        SOCK_RECV_BUF_REQUESTED,ACTUAL(bytes) = 0, 87616
            PRIMARY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
            STANDBY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
            HADR_LOG_GAP(bytes) = 0
        STANDBY_REPLAY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
        STANDBY_RECV_REPLY_GAP(bytes) = 0
            PRIMARY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
            STANDBY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
            STANDBY_REPLY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
        STANDBY_RECV_BUF_SIZE(pages) = 16
            STANDBY_RECV_BUF_PERCENT = 0
        STANDBY_SPOOL_LIMIT(pages) = 0
            PEER_WINDOW(seconds) = 0
            READS_ON_STANDBY_ENABLED = Y
        STANDBY_REPLY_ONLY_WINDOW_ACTIVE = N

            HADR_ROLE = PRIMARY
            REPLAY_TYPE = PHYSICAL
            HADR_SYNCMODE = SUPERASYNC
            STANDBY_ID = 2
            LOG_STREAM_ID = 0
                HADR_STATE = REMOTE_CATCHUP
        PRIMARY_MEMBER_HOST = host1
            PRIMARY_INSTANCE = db2inst1
            PRIMARY_MEMBER = 0
        STANDBY_MEMBER_HOST = host3
            STANDBY_INSTANCE = db2inst3
            STANDBY_MEMBER = 0
            HADR_CONNECT_STATUS = CONNECTED
            HADR_CONNECT_STATUS_TIME = 06/08/2011 13:35:51.724447 (1307565351)
        HEARTBEAT_INTERVAL(seconds) = 30
            HADR_TIMEOUT(seconds) = 120
            TIME_SINCE_LAST_RECV(seconds) = 16
            PEER_WAIT_LIMIT(seconds) = 0
            LOG_HADR_WAIT_CUR(seconds) = 0.000
        LOG_HADR_WAIT_RECENT_AVG(seconds) = 0.006298
        LOG_HADR_WAIT_ACCUMULATED(seconds) = 0.516
            LOG_HADR_WAIT_COUNT = 82
        SOCK_SEND_BUF_REQUESTED,ACTUAL(bytes) = 0, 16384
        SOCK_RECV_BUF_REQUESTED,ACTUAL(bytes) = 0, 87380
            PRIMARY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
            STANDBY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
            HADR_LOG_GAP(bytes) = 0
        STANDBY_REPLAY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
        STANDBY_RECV_REPLY_GAP(bytes) = 0
            PRIMARY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
            STANDBY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
            STANDBY_REPLY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
        STANDBY_RECV_BUF_SIZE(pages) = 16
            STANDBY_RECV_BUF_PERCENT = 0
        STANDBY_SPOOL_LIMIT(pages) = 0
            PEER_WINDOW(seconds) = 0
            READS_ON_STANDBY_ENABLED = Y
        STANDBY_REPLY_ONLY_WINDOW_ACTIVE = N

            HADR_ROLE = PRIMARY
            REPLAY_TYPE = PHYSICAL
            HADR_SYNCMODE = SUPERASYNC
            STANDBY_ID = 3
            LOG_STREAM_ID = 0
                HADR_STATE = REMOTE_CATCHUP
        PRIMARY_MEMBER_HOST = host1
            PRIMARY_INSTANCE = db2inst1
            PRIMARY_MEMBER = 0
        STANDBY_MEMBER_HOST = host4

```

```

        STANDBY_INSTANCE = db2inst4
        STANDBY_MEMBER = 0
        HADR_CONNECT_STATUS = CONNECTED
        HADR_CONNECT_STATUS_TIME = 06/08/2011 13:46:51.561873 (1307566011)
        HEARTBEAT_INTERVAL(seconds) = 30
        HADR_TIMEOUT(seconds) = 120
        TIME_SINCE_LAST_RECV(seconds) = 6
        PEER_WAIT_LIMIT(seconds) = 0
        LOG_HADR_WAIT_CUR(seconds) = 0.000
        LOG_HADR_WAIT_RECENT_AVG(seconds) = 0.006298
        LOG_HADR_WAIT_ACCUMULATED(seconds) = 0.516
        LOG_HADR_WAIT_COUNT = 82
        SOCK_SEND_BUF_REQUESTED,ACTUAL(bytes) = 0, 16384
        SOCK_RECV_BUF_REQUESTED,ACTUAL(bytes) = 0, 87380
            PRIMARY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
            STANDBY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
            HADR_LOG_GAP(bytes) = 0
        STANDBY_REPLY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
        STANDBY_RECV_REPLY_GAP(bytes) = 0
            PRIMARY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
            STANDBY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
            STANDBY_REPLY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
        STANDBY_RECV_BUF_SIZE(pages) = 16
        STANDBY_RECV_BUF_PERCENT = 0
        STANDBY_SPOOL_LIMIT(pages) = 0
        PEER_WINDOW(seconds) = 0
        READS_ON_STANDBY_ENABLED = Y
        STANDBY_REPLY_ONLY_WINDOW_ACTIVE = N

```

Examples: Takeover in HADR multiple standby mode

This set of examples of takeovers (both forced and unforced) in HADR multiple standby mode is based on a three-standby setup. The purpose of these examples is to show how the multiple standby automatic reconfiguration works in a takeover situation.

- “A principal standby takes over gracefully (role switch)” on page 454
- “An auxiliary standby takes over by force (failover)” on page 455
- “An auxiliary standby takes over by force (failover) in a SA MP environment” on page 457

The initial setup for each of the examples is as follows:

- a primary database (host1)
- a principal standby (host2)
- two auxiliary standbys (host3 and host4)

All of the databases are called hadr_db. The primary and principal standby have their synchronization mode set to SYNC and the standbys have theirs set to SUPERASYNC.

The configuration for each database is shown in Table 28.

Table 28. Configuration values for each HADR database

Configuration parameter	Host1	Host2	Host3	Host4
hadr_target_list	host2:40 host3:41 host4:42	host1:10 host3:41 host4:42	host2:40 host1:10 host4:42	host2:40 host1:10 host3:41
hadr_remote_host	host2	host1	host1	host1
hadr_remote_svc	40	10	10	10
hadr_remote_inst	dbinst2	dbinst1	dbinst1	dbinst1

Table 28. Configuration values for each HADR database (continued)

Configuration parameter	Host1	Host2	Host3	Host4
hadr_local_host	host1	host2	host3	host4
hadr_local_svc	10	40	41	42
Configured hadr_syncmode (Refers to the explicitly set synchronization mode, which is used if the database becomes a primary)	SYNC	SYNC	SUPERASYNC	SUPERASYNC
Effective hadr_syncmode (Refers to the synchronization mode that is used if the database is currently a standby)	n/a	SYNC	SUPERASYNC	SUPERASYNC

A principal standby takes over gracefully (role switch)

The DBA performs a takeover on the principal standby by issuing the following command on host2:

```
DB2 TAKEOVER HADR ON DB hadr_db
```

After the takeover is completed successfully, host2 becomes the new primary and host1, which is the first entry in the **hadr_target_list** of host2 (as shown in Table 28 on page 453), becomes its principal standby. Their sync mode is SYNC mode because host2 is configured with an **hadr_syncmode** of SYNC. The auxiliary standby targets, host3 and host4, have their **hadr_remote_host** and **hadr_remote_svc** pointing at the old primary, host1, but are automatically redirected to the new primary, host2. In this redirection, host3 and host4 update (persistently) their **hadr_remote_host**, **hadr_remote_svc**, and **hadr_remote_inst** configuration parameters. They reconnect to host2 as auxiliary standbys, and are told by host2 to use an effective synchronization mode of SUPERASYNC (regardless of what they have locally configured for **hadr_syncmode**). They do not update their settings for **hadr_syncmode** persistently. The configuration for each database is shown in Table 29.

Table 29. Configuration values for each HADR database after a role switch. Rows 3 to 5 in columns 4 and 5 have been bolded to show that they have been auto-reconfigured

Configuration parameter	Host1	Host2	Host3	Host4
hadr_target_list	host2:40 host3:41 host4:42	host1:10 host3:41 host4:42	host2:40 host1:10 host4:42	host2:40 host1:10 host3:41
hadr_remote_host	host2	host1	host2	host2
hadr_remote_svc	40	10	40	40
hadr_remote_inst	dbinst2	dbinst1	dbinst2	dbinst2
hadr_local_host	host1	host2	host3	host4

Table 29. Configuration values for each HADR database after a role switch (continued). Rows 3 to 5 in columns 4 and 5 have been bolded to show that they have been auto-reconfigured

Configuration parameter	Host1	Host2	Host3	Host4
hadr_local_svc	10	40	41	42
Configured hadr_syncmode	SYNC	SYNC	SUPERASYNC	SUPERASYNC
Effective hadr_syncmode	SYNC	n/a	SUPERASYNC	SUPERASYNC

Note: A number of values are not updated for the following reasons

- Because host2 already has its **hadr_remote_host** and **hadr_remote_svc** configuration parameters pointing at its principal standby, host1, these values are not updated on host2.
- Because host1 already has its **hadr_remote_host** and **hadr_remote_svc** configuration parameters pointing at the new primary, these values are not updated on host1.
- Because host1's operational synchronization mode is SYNC and host3 and host4's operational synchronization modes are SUPERASYNC, there is no change for the effective synchronization mode.

An auxiliary standby takes over by force (failover)

A widespread power outage in City A results in the primary (host1) becoming unavailable. Normally, the principal standby (host2) which is in SYNC mode would be the best candidate for taking over and becoming the new primary, but the power outage means that host2 is momentarily unavailable as well. The DBA queries the two auxiliary standbys to determine which one has the most log data:
`db2pd -hadr -db hadr_db | grep 'PRIMARY_LOG_FILE,PAGE,POS|STANDBY_LOG_FILE,PAGE,POS'`

The DBA determines that host3 is the most up to date (although it is still a little behind in log replay) and picks that host as the new primary:

```
DB2 TAKEOVER HADR ON DB hadr_db BY FORCE
```

After the takeover is completed successfully, host3 becomes the new primary. Meanwhile, host2 becomes available again. host3 informs host2 and host4 that it is now the primary. On host3, the values for **hadr_remote_host**, **hadr_remote_svc**, and **hadr_remote_inst** are reconfigured to point to host2, which is the principal standby because it is the first entry in the **hadr_target_list** on host3. On host2, the synchronization mode is reconfigured to SUPERASYNC because that is the setting for **hadr_syncmode** on host3; in addition, the **hadr_remote_host**, **hadr_remote_svc**, and **hadr_remote_inst** are updated (persistently). host4 is automatically redirected to the new primary, host3. In this redirection, host4 updates (persistently) its **hadr_remote_host**, **hadr_remote_svc**, and **hadr_remote_inst** configuration parameters. There is no automatic reconfiguration on host1 until it becomes available again. The configuration for each database is shown in Table 30 on page 456.

Table 30. Configuration values for each HADR database after a failover. Rows 3 to 5 in columns 3 to 5 have been bolded to show that they have been auto-reconfigured

Configuration parameter	Host1 (unavailable)	Host2	Host3	Host4
hadr_target_list	host2:40 host3:41 host4:42	host1:10 host3:41 host4:42	host2:40 host1:10 host4:42	host2:40 host1:10 host3:41
hadr_remote_host	host2	host3	host2	host3
hadr_remote_svc	40	41	40	41
hadr_remote_inst	dbinst2	dbinst3	dbinst2	dbinst3
hadr_local_host	host1	host2	host3	host4
hadr_local_svc	10	40	41	42
Configured hadr_syncmode	SYNC	SYNC	SUPERASYNC	SUPERASYNC
Effective hadr_syncmode	n/a	SUPERASYNC	n/a	SUPERASYNC

After a short period of time, host1 becomes available. The DBA tries to start host1 as a standby, but because host1 has more logs than were propagated to host3, host1 is rejected as part of the initial handshake with the new primary. The DBA takes a backup of the new primary, restores it to host1, and starts HADR on that host:

```
DB2 BACKUP DB hadr_db
DB2 RESTORE DB hadr_db
DB2 START HADR ON DB hadr_db AS STANDBY
```

As is shown in Table 31, host1 is reconfigured.

Table 31. Configuration values for a reintegrated standby. Various rows in column 2 have been bolded to show that they have been auto-reconfigured

Configuration parameter	Host1	Host2	Host3	Host4
hadr_target_list	host2:40 host3:41 host4:42	host1:10 host3:41 host4:42	host2:40 host1:10 host4:42	host2:40 host1:10 host3:41
hadr_remote_host	host3	host3	host2	host3
hadr_remote_svc	41	41	40	41
hadr_remote_inst	dbinst3	dbinst3	dbinst2	dbinst3
hadr_local_host	host1	host2	host3	host4
hadr_local_svc	10	40	41	42
Configured hadr_syncmode	SYNC	SYNC	SUPERASYNC	SUPERASYNC
Effective hadr_syncmode	SUPERASYNC	SUPERASYNC	n/a	SUPERASYNC

If the DBA wants to make host1 the primary again, then all that is required is a failback, which will restore the original configuration shown in Table 28 on page 453.

An auxiliary standby takes over by force (failover) in a SA MP environment

This example is similar to the previous one, but HADR has been deployed with IBM Tivoli System Automation for Multiplatforms (SA MP) to automate failover.

A power failure in City A results in the principal standby (host2) becoming unavailable. Following that, there is an outage on the primary (host1). Normally, SA MP, the cluster manager, would automatically fail over to the principal standby (host2), but the power outage means that one of the auxiliary standbys needs to be the takeover target. Failover cannot be automated to auxiliary standbys, so the DBA must do it manually. However, before doing this, the DBA needs to ensure that TSA is disabled so that if host1 or host2 become available, there is no possibility for a *split brain* situation, in which more than one database is operating independently as a primary. To do this, the DBA issues the following command on host1 and host2 (whenever they become available):

```
db2haicu -disable
```

In addition, the DBA needs to keep host1 offline to eliminate the possibility that the old primary will restart if a client connects to it.

The DBA queries the two auxiliary standbys to determine which one has the most log data:

```
db2pd -hadr -db hadr_db | grep 'STANDBY_LOG_FILE,PAGE,POS'
```

The DBA determines that host3 is the most up to date and picks that host as the new primary.

Then, the DBA issues the force takeover on host3:

```
DB2 TAKEOVER HADR ON DB hadr_db BY FORCE
```

After the takeover is completed successfully, host3 becomes the new primary. Meanwhile, host2 becomes available again. host3 informs host2 and host4 that it is now the primary. On host3, the values for **hadr_remote_host**, **hadr_remote_svc**, and **hadr_remote_inst** are reconfigured to point to host2, which is the principal standby because it is the first entry in the **hadr_target_list** on host3. On host2, the synchronization mode is reconfigured to SUPERASYNC because that is the setting for **hadr_syncmode** on host3; in addition, the **hadr_remote_host**, **hadr_remote_svc**, and **hadr_remote_inst** are updated (persistently). host4 is automatically redirected to the new primary, host3. In this redirection, host4 updates (persistently) its **hadr_remote_host**, **hadr_remote_svc**, and **hadr_remote_inst** configuration parameters. There is no automatic reconfiguration on host1. The configuration for each database is shown in Table 32.

Table 32. Configuration values for each HADR database after a failover. Rows 3 to 5 in columns 3 to 5 have been bolded to show that they have been auto-reconfigured

Configuration parameter	Host1 (unavailable)	Host2	Host3	Host4
hadr_target_list	host2:40 host3:41 host4:42	host1:10 host3:41 host4:42	host2:40 host1:10 host4:42	host2:40 host1:10 host3:41
hadr_remote_host	host2	host3	host2	host3
hadr_remote_svc	40	41	40	41
hadr_remote_inst	dbinst2	dbinst3	dbinst2	dbinst3

Table 32. Configuration values for each HADR database after a failover (continued). Rows 3 to 5 in columns 3 to 5 have been bolded to show that they have been auto-reconfigured

Configuration parameter	Host1 (unavailable)	Host2	Host3	Host4
hadr_local_host	host1	host2	host3	host4
hadr_local_svc	10	40	41	42
Configured hadr_syncmode	SYNC	SYNC	SUPERASYNC	SUPERASYNC
Effective hadr_syncmode	n/a	SUPERASYNC	n/a	SUPERASYNC

HADR reads on standby feature

You can use the reads on standby capability to run read-only operations on the standby database in your High Availability and Disaster Recovery (HADR) solution. Read operations running on a standby do not affect the standby's main role of replaying logs shipped from the primary database.

The reads on standby feature reduces the total cost of ownership of your HADR setup. This expanded role of the standby database allows you to utilize the standby in new ways, such as running some of the workload that would otherwise be running on your primary database. This, in turn frees up the primary for additional workloads.

Read and write clients continue to connect to the primary database; however read clients can also connect to the read-enabled standby, or *active standby*, as long as it is not in the local catchup state or the replay-only window. An active standby's main role is still to replay logs shipped from the primary. As a result, the data on the standby should be virtually identical to the data on the primary. In the event of a failover, any user connections to the standby will be terminated while the standby takes over as the new primary database.

All types of read queries, including scrollable and non-scrollable cursors, are supported on the standby. Read capability is supported in all four HADR synchronization modes (SYNC, NEARSYNC, ASYNC, and SUPERASYNC) and in all HADR states except local catchup.

Enabling reads on standby

You can enable the reads on standby feature on your High Availability and Disaster Recovery (HADR) standby database using the **DB2_HADR_ROS** registry variable.

Before you begin

It is recommended that database configuration parameter **logindexbuild** be set to ON. This will prevent a performance impact from query access plans avoiding any invalid indexes.

It is also recommended that you use a virtual IP when you have reads on standby enabled. Client reroute does not differentiate between writable databases (primary and standard databases) and read-only databases (standby databases). Configuring client reroute between the primary and standby might route applications to the database on which they are not intended to be run.

About this task

You cannot use automatic client reroute (ACR) if you enable reads on standby.

Procedure

1. Set the **DB2_HADR_ROS** registry variable to ON.
2. Set up and initialize the primary and standby databases for HADR. Refer to “Initializing high availability disaster recovery (HADR)” on page 477.

Results

Your standby database is now considered an *active standby*, meaning that it will accept read-only workloads.

What to do next

You can now utilize your standby database as you see fit, such as configuring some of your read-only workload to run on it.

To enable your applications to maintain access to your standby database, follow the steps described in the “Continuous access to Read on Standby databases using Virtual IP addresses” white paper.

Data concurrency on the active standby database

Changes on the HADR primary database may not necessarily be reflected on the HADR active standby database. Uncommitted changes on the primary may not replicate to the standby until the primary flushes, or sends, its logs to disk.

Logs are only guaranteed to be flushed to disk-and, therefore sent to the standby-after they have been committed. Log flushes can also be triggered by undeterministic conditions such as a log buffer full situation. As a result, it is possible for uncommitted changes on the primary to remain in the primary's log buffer for a long time. Because the logger avoids flushing partial pages, this situation may particularly affect small uncommitted changes on the primary.

If your workload running on the standby requires the data to be virtually identical to the data on the primary, you should consider committing your transactions more frequently.

Isolation level on the active standby database

The only isolation level that is supported on an active standby database (an HADR standby database that is read enabled) is Uncommitted Read (UR). If the isolation level requested by an application, statement, or sub-statement is higher than UR, an error will be returned (SQL1773N Reason Code 1).

If you require an isolation level other than UR, consider using the HADR primary instead of the standby for this application. If you simply want to avoid receiving this message, set the **DB2_STANDBY_ISO** registry variable to UR. When **DB2_STANDBY_ISO** is set to UR, the isolation level will be silently coerced to UR. This setting takes precedence over all other isolation settings such as statement isolation and package isolation.

Replay-only window on the active standby database

When an HADR active standby database is replaying DDL log records or maintenance operations, the standby enters the *replay-only window*. When the

standby is in the replay-only window, existing connections to the standby are terminated and new connections to the standby are blocked (SQL1776N).

New connections are allowed on the standby after the replay of all active DDL or maintenance operations has completed.

The only user connections that can remain active on a standby in the replay-only window are connections that are executing **DEACTIVATE DATABASE** or **TAKEOVER** commands. When applications are forced off at the outset of the replay-only window, an error is returned (SQL1224N). Depending on the number of readers connected to the active standby, there may be a slight delay before the DDL log records or maintenance operations are replayed on the standby.

There are a number of DDL statements and maintenance operations that, when run on the HADR primary, will trigger a replay-only window on the standby. The following lists are not exhaustive.

DDL statements

- CREATE, ALTER, or DROP TABLE (except DROP TABLE for DGTT)
- CREATE GLOBAL TEMP TABLE
- TRUNCATE TABLE
- RENAME TABLE
- RENAME TABLESPACE
- CREATE, DROP, or ALTER INDEX
- CREATE or DROP VIEW
- CREATE, ALTER, or DROP TABLESPACE
- CREATE, ALTER, or DROP BUFFER POOL
- CREATE, ALTER, or DROP FUNCTION
- CREATE, ALTER, or DROP PROCEDURE
- CREATE or DROP TRIGGER
- CREATE, ALTER, or DROP TYPE
- CREATE, ALTER, or DROP ALIAS
- CREATE or DROP SCHEMA
- CREATE, ALTER, or DROP METHOD
- CREATE, ALTER, or DROP MODULE
- CREATE, ALTER, or DROP NICKNAME
- CREATE, ALTER, or DROP SEQUENCE
- CREATE, ALTER, or DROP WRAPPER
- CREATE, ALTER, or DROP FUNCTION MAPPING
- CREATE or DROP INDEX EXTENSION
- CREATE or DROP INDEX FOR TEXT
- CREATE or DROP EVENT MONITOR
- CREATE, ALTER, or DROP SECURITY LABEL
- CREATE, ALTER, or DROP SECURITY LABEL COMPONENT
- CREATE, ALTER, or DROP SECURITY POLICY
- CREATE or DROP TRANSFORM
- CREATE, ALTER, or DROP TYPE MAPPING
- CREATE, ALTER, or DROP USER MAPPING

- CREATE or DROP VARIABLE
- CREATE, ALTER, or DROP WORKLOAD
- GRANT USAGE ON WORKLOAD
- REVOKE USAGE ON WORKLOAD
- CREATE, ALTER, or DROP SERVICE CLASS
- CREATE, ALTER, or DROP WORK CLASS SET
- CREATE, ALTER, or DROP WORK ACTION SET
- CREATE, ALTER, or DROP THRESHOLD
- CREATE, ALTER, or DROP HISTOGRAM TEMPLATE
- AUDIT
- CREATE, ALTER, or DROP AUDIT POLICY
- CREATE or DROP ROLE
- CREATE, ALTER, or DROP TRUSTED CONTEXT
- REFRESH TABLE
- SET INTEGRITY

Maintenance operations

- Classic, or offline, reorg
- Inplace, or online, reorg
- Index reorg (indexes all, individual index)
- MDC and ITC reclaim reorg
- Load
- Bind or rebinding
- db2rbind
- Runstats
- Table move
- Auto statistics
- Auto reorg
- Real Time Statistics

Other operation or actions

- Automatic Dictionary Creation for tables with COMPRESS YES attribute
- Asynchronous Index Cleanup on detached table partition
- Implicit rebinding
- Implicit index rebuild
- Manual update of statistics.
- Deferred MDC rollout
- Asynchronous Index cleanup after MDC rollout
- Reuse of a deleted MDC or ITC block on insert into MDC or ITC table
- Asynchronous background processes updating catalog tables SYSJOBS and SYSTASKS for inserting, updating, and deleting tasks

Monitoring the replay-only window

You can monitor the replay-only window using the **db2pd** command with the **-hadr** option (on either the standby or the primary) or the MON_GET_HADR table

function (from the primary). The standby's status, including information about the replay-only window, is sent to the primary on every heartbeat.

There are three pertinent elements to monitor:

- **STANDBY_REPLAY_ONLY_WINDOW_ACTIVE**, which indicates whether DDL or maintenance-operation replay is in progress on the standby. Normally, the value is N, but when the replay-only window is active, the value is Y.
- **STANDBY_REPLAY_ONLY_WINDOW_START**, which indicates the time at which the current replay-only window (if there is one) became active.
- **STANDBY_REPLAY_ONLY_WINDOW_TRAN_COUNT**, which indicates the total number of existing uncommitted DDL or maintenance transactions executed so far in the current replay-only window (if there is one).

To use the table function, issue something similar to the following query on the primary:

```
select STANDBY_ID, STANDBY_REPLAY_ONLY_WINDOW_ACTIVE, STANDBY_REPLAY_ONLY_WINDOW_START,
STANDBY_REPLAY_ONLY_WINDOW_TRAN_COUNT from table (mon_get_hadr(NULL))
```

Here is an example using the **db2pd** command on a standby that is currently in a replay-only window:

```
db2pd -hadr db HADRDB
```

```
Database Member 0 -- Database HADRDB -- Active -- Up 0 days 00:23:17 -- Date 06/08/2011 13:57:23
```

```
HADR_ROLE = STANDBY
REPLAY_TYPE = PHYSICAL
HADR_SYNCMODE = NEARSYNC
STANDBY_ID = 1
LOG_STREAM_ID = 0
HADR_STATE = PEER
PRIMARY_MEMBER_HOST = hostP.ibm.com
PRIMARY_INSTANCE = db2inst
PRIMARY_MEMBER = 0
STANDBY_MEMBER_HOST = hostS1.ibm.com
STANDBY_INSTANCE = db2inst
STANDBY_MEMBER = 0
HADR_CONNECT_STATUS = CONNECTED
HADR_CONNECT_STATUS_TIME = 06/08/2011 13:38:10.199479 (1307565490)
HEARTBEAT_INTERVAL(seconds) = 25
HADR_TIMEOUT(seconds) = 120
TIME_SINCE_LAST_RECV(seconds) = 3
PEER_WAIT_LIMIT(seconds) = 0
LOG_HADR_WAIT_CUR(seconds) = 0.000
LOG_HADR_WAIT_RECENT_AVG(seconds) = 0.006298
LOG_HADR_WAIT_ACCUMULATED(seconds) = 0.516
LOG_HADR_WAIT_COUNT = 82
SOCK_SEND_BUF_REQUESTED,ACTUAL(bytes) = 0, 50772
SOCK_RECV_BUF_REQUESTED,ACTUAL(bytes) = 0, 87616
PRIMARY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
STANDBY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
HADR_LOG_GAP(bytes) = 0
STANDBY_REPLY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
STANDBY_RECV_REPLY_GAP(bytes) = 0
PRIMARY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
STANDBY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
STANDBY_REPLY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
STANDBY_RECV_BUF_SIZE(pages) = 16
STANDBY_RECV_BUF_PERCENT = 0
STANDBY_SPOOL_LIMIT(pages) = 0
PEER_WINDOW(seconds) = 0
READS_ON_STANDBY_ENABLED = Y
```

```
STANDBY_REPLAY_ONLY_WINDOW_ACTIVE = Y
STANDBY_REPLAY_ONLY_WINDOW_START = 06/08/2011 13:50:23
STANDBY_REPLAY_ONLY_WINDOW_TRAN_COUNT = 5
```

Recommendations for minimizing the impact of the replay-only window

Because replay operations on an HADR standby take priority over readers, frequent read-only windows can be disruptive to readers connected to or attempting to connect to the standby. To avoid or minimize this impact, consider the following recommendations:

- Run DDL and maintenance operations during a scheduled maintenance window, preferably at off-peak hours.
- Run DDL operations collectively rather than in multiple groups.
- Run **REORG** or **RUNSTATS** only on the required tables instead of all tables.
- Terminate applications on the active standby using the **FORCE APPLICATION** command with the **ALL** option before running the DDL or maintenance operations on the primary. Monitor the replay-only window to determine when it is inactive, and redeploy the applications on the standby.

HADR delayed replay

HADR delayed replay helps prevent data loss due to errant transactions. To implement HADR delayed replay, set the **hadr_replay_delay** database configuration parameter on the HADR standby database.

Delayed replay intentionally keeps the standby database at a point in time that is earlier than that of the primary database by delaying replay of logs on that standby. If an errant transaction is executed on the primary, you have until the configured time delay has elapsed to take action to prevent the errant transaction from being replayed on the standby. To recover the lost data, you can either copy this data back to the primary, or you can have the standby take over as the new primary database.

Delayed replay works by comparing timestamps in the log stream, which is generated on the primary, and the current time of the standby. As a result, it is important to synchronize the clocks of the primary and standby databases. Transaction commit is replayed on the standby according to the following equation:

(current time on the standby - value of the hadr_replay_delay configuration parameter) >= timestamp of the committed log record

You should set the **hadr_replay_delay** database configuration parameter to a large enough value to allow time to detect and react to errant transactions on the primary.

You can use this feature in either single standby mode or multiple standby mode. In multiple standby mode, typically one or more standbys stays current with the primary for high availability or disaster recovery purposes, and one standby is configured with delayed replay for protection against errant transactions. If you use this feature in single standby mode, you should not enable IBM Tivoli System Automation for Multiplatforms because the takeover will fail.

There are several important restrictions for delayed replay:

- You can set the **hadr_replay_delay** configuration parameter only on a standby database.

- A **TAKEOVER** command on a standby with replay delay enabled will fail. You must first set the **hadr_replay_delay** configuration parameter to 0 and then deactivate and reactivate the standby to pick up the new value, and then issue the **TAKEOVER** command.
 - The delayed replay feature is supported only in SUPERASYNC mode. Because log replay is delayed, a lot of unreplayed log data might accumulate on the standby, filling up receive buffer and spool (if configured). In other synchronization modes, this would cause the primary to be blocked.
- The objective of this feature is to protect against application error. If you want to use this feature and ensure that there is no data loss in the event of a primary failure, consider a multiple standby setup with a more synchronous setting on the principal standby.

Recommendations

Delayed replay and disaster recovery

Consider using a small delay if you are using the standby database for disaster recovery purposes and errant transaction protection.

Delayed replay and the HADR reads on standby feature

Consider using a small delay if you are using the standby database for reads on standby purposes, so that reader sessions can see more up-to-date data. Additionally, because reads on standby runs in “uncommitted read” isolation level, it can see applied, but not yet committed changes that are technically still delayed from replay. These uncommitted transactions can be rolled back in errant transaction recovery procedure when you roll forward the standby to the PIT that you want and then stop.

Delayed replay and log spooling

If you enable delayed replay, it is recommended that you also enable log spooling by setting the **hadr_spool_limit** database configuration parameter. Because of the intentional delay, the replay position can be far behind the log receive position on the standby. Without spooling, log receive can only go beyond replay by the amount of the receive buffer. With spooling, the standby can receive many more logs beyond the replay position, providing more protection against data loss in case of primary failure. Note that in either case, because of the mandatory SUPERASYNC mode, the primary won't be blocked by the delayed replay.

Recovering data by using HADR delayed replay

Using the HADR time-delayed replay feature, you can recover data that was lost because of an errant transaction on the primary database by stopping HADR on a standby before that transaction is replayed.

Before you begin

Delayed replay must have already been enabled for your standby database.

If log replay on the standby, indicated by **STANDBY_REPLAY_LOG_TIME**, has passed the commit time for the errant transaction on the standby, you cannot recover the data using the following procedure. You can determine the **STANDBY_REPLAY_LOG_TIME** by using the **db2pd** command with the **-hadr** parameter or the **MON_GET_HADR** table function.

Restriction: A standby database for which you set the `hadr_replay_delay` configuration parameter cannot take over as a primary; you must first disable delayed replay on that standby.

Procedure

To recover from an errant transaction, perform the following steps on the standby on which you enabled delayed replay:

1. Verify the timing:

- a. Ensure that standby has not yet replayed the transaction. The `STANDBY_REPLAY_LOG_TIME` value must not have reached the errant transaction commit time.
- b. Ensure that the standby has received the relevant logs. The `STANDBY_LOG_TIME` value, which indicates logs received, must have reached a PIT before the errant transaction commit time, but close to the errant transaction commit time. This will be the rollforward PIT used in step 3. If the standby has not yet received enough log files, you can wait until more logs are shipped over, but you run the risk of the replay time reaching the errant transaction time. For example, if the delay is 1 hour, you should stop HADR no later than 50 minutes after the errant transaction time (allowing a 10-minute safety margin), even if log shipping has yet not reached the PIT that you want.

Alternatively, if a shared log archive is available and the logs are already archived, then there is no need to wait. If the logs are not archived yet, the logs can be archived using the `ARCHIVE LOG` command. Otherwise, the user can manually copy complete log files from the primary to the time-delayed standby (the overflow log path is preferred, otherwise, use the log path). For these alternate methods, deactivate the standby first to avoid interference with standby log shipping and replay.

You can determine these times by issuing `db2pd -db dbname -hadr` or by enabling the reads on standby feature on the standby and then issuing the following query, which uses the `MON_GET_HADR` table function:

```
DB2 "select HADR_ROLE, STANDBY_ID, STANDBY_LOG_TIME, STANDBY_REPLAY_LOG_TIME,  
      varchar(PRIMARY_MEMBER_HOST,20) as PRIMARY_MEMBER_HOST,  
      varchar(STANDBY_MEMBER_HOST,20) as STANDBY_MEMBER_HOST  
   from table (mon_get_hadr(NULL))"
```

2. Stop HADR on the standby database:

```
DB2 STOP HADR ON DATABASE dbname
```

3. Roll forward the standby to the PIT that you want and then stop:

```
DB2 ROLLFORWARD DB dbname to time-stamp and STOP
```

4. Use one of the following approaches:

- Restore the lost data on the primary:
 - a. Copy the affected data from the standby and send it back to the primary. If the errant transaction dropped a table, you could export it on the standby and import it to the primary. If the errant transaction deleted rows from a table, you could export the table on the standby and use an import replace operation on the primary.
 - b. Reinitialize the delayed-replay standby because its log stream has diverged from the primary's. No action is needed on any other standbys because they continue to follow the primary and any data repair on the primary is also replicated to them.

- c. Restore the database using a backup image taken on the primary. The image can be one taken at any time.
- d. Remove all log files in standby log path. This step is important. The **ROLLFORWARD...** STOP command in step 3 made the database log stream diverge from the primary. If the files are left alone, the newly restored database would follow that log stream and also diverge from the primary. Alternatively, you can drop the database before the restore for a clean start, but then you will also lose the current configuration including HADR configuration.
- e. Issue the **START HADR** command with the AS STANDBY option on the database. The database should then activate and connect to the primary.
- Have the standby with the intact data become the primary:
 - a. Shut down the old primary to avoid split brain
 - b. On the delayed-replay database, set the **hadr_replay_delay** configuration parameter to 0. Reconfigure the other parameters like **hadr_target_list** if needed. Then run **START HADR** command with the AS PRIMARY BY FORCE options on the database to convert it to the new primary. Use the BY FORCE option because there is no guarantee that the configured principal standby (which could be the old primary) will be able to connect.
 - c. Redirect clients to the new primary.
 - d. The other standbys will be automatically redirected to the new primary. However, if a standby received logs from the old primary beyond the point where old and new primary diverge (the PIT used in step 3), it will be rejected by the new primary. If this happens, reinitialize this standby using the same procedure as reinitializing the old primary.
 - e. Reinitialize the old primary because its log stream has diverged from the new primary's.
 - f. Restore database using a backup image taken on the new primary, or taken on the old primary before the PIT used in step 3.
 - g. Remove all log files in the log path. If you do not do this, the newly restored database will follow the old primary's log stream and diverge from the new primary. Alternatively, you can drop the database before the restore for a clean start, but then you also lose the current configuration including HADR configuration.
 - h. Issue the **START HADR** command with the AS STANDBY option on the database. The database should then activate and connect to the primary.

Performing rolling updates in a DB2 High Availability Disaster Recovery (HADR) environment

Use this procedure in a high availability disaster recovery (HADR) environment when you upgrade software or hardware, update your DB2 database product software, or change database configuration parameters.

This procedure keeps database service available throughout the rolling update process, with only a momentary service interruption when processing is switched from one database to the other. With multiple standbys, you can provide continued HA and DR protection throughout the rolling update process.

Before you begin

Review the system requirements for HADR. See “System requirements for DB2 high availability disaster recovery (HADR)” on page 469.

The HADR pair should be in peer state before starting the rolling update.

If you have two HADR databases (databaseA and database B) set up the following way, perform a role switch on one database so that both primaries are on the same system during the fix pack update:

- The primary for databaseA runs on system1, and the standby runs on system2
- The primary for databaseB runs on system2, and the standby runs on system1

The overall capacity of the databases might be reduced, but it keeps both database online during the procedure.

Note: All DB2 fix pack updates, hardware upgrades, and software upgrades should be implemented in a test environment before being applied to your production system.

About this task

Use this procedure to perform a rolling update on your DB2 database system and update the DB2 database product software from one modification level to another. For example, applying a fix pack to a DB2 database product software. During rolling updates, the modification level or fix pack level of the standby database can be later than that of the primary database while testing the new level. However, you should not keep this configuration for an extended period to reduce the risk of using features that might be incompatible between the levels. The primary and standby databases will not connect to each other if the modification level of the DB2 database product software for the primary database is later than that of the standby database.

A rolling update cannot be used to upgrade from an earlier version to a later version of a DB2 database product software. For example, you cannot use this procedure to upgrade a DB2 database product from Version 9.7 to Version 10.1.

You cannot use this procedure to update the DB2 HADR configuration parameters. Updates to HADR configuration parameters should be made separately. Because HADR requires the parameters on the primary and standby to be the same, this might require both the primary and standby databases to be deactivated and updated at the same time.

Procedure

To perform a rolling update in an HADR environment:

1. Update the standby database by issuing the following steps:
 - a. Use the **DEACTIVATE DATABASE** command to shut down the standby database.
 - b. If necessary, shut down the instance on the standby database.
 - c. Change one or more of the following: the software, the hardware, or the DB2 configuration parameters.

Note: You cannot change any HADR configuration parameters when performing a rolling update.

- d. If necessary, restart the instance on the standby database.
- e. Use the **ACTIVATE DATABASE** command to restart the standby database.
- f. Ensure that HADR enters peer state. Use the MON_GET_HADR table function (on the primary or a read-enabled standby) or the **db2pd** command with the -hadr option to check this.

2. Switch the roles of the primary and standby databases:
 - a. Issue the **TAKEOVER HADR** command on the standby database.
 - b. Direct clients to the new primary database. This can be done using automatic client reroute.

Note: Because the standby database takes over as the primary database, the new primary database is now updated. If you are applying a DB2 fix pack, the **TAKEOVER HADR** command changes the role of the original primary database to standby database. However, the command does not let the new standby database connect to the newly updated primary database. Because the new standby database uses an older version of the DB2 database system, it might not understand the new log records generated by the updated primary database, and it will be shut down. In order for the new standby database to reconnect with the new primary database (that is, for the HADR pair to reform), the new standby database must also be updated.
3. Update the original primary database (which is now the standby database) using the same procedure as in step 1 on page 467. When you have done this, both databases are updated and connected to each other in HADR peer state. The HADR system provides full database service and full high availability protection.
4. Optional: To enable the HADR reads on standby feature during the rolling update perform the following steps to ensure the consistency of the internal DB2 packages on the standby database before read operations are introduced. The binding of internal DB2 packages occurs at first connection time, and can complete successfully only on the primary database.
 - a. Enable the HADR reads on standby feature on the standby database as follows:
 - 1) Set the **DB2_HADR_ROS** registry variable to **ON** on the standby database.
 - 2) Use the **DEACTIVATE DATABASE** command to shut down the standby database.
 - 3) Restart the instance on the standby database.
 - 4) Use the **ACTIVATE DATABASE** command to restart the standby database.
 - 5) Ensure that HADR enters peer state. Use the **MON_GET_HADR** table function (on the primary or a read-enabled standby) or the **db2pd** command with the **-hadr** option to check this.
 - b. Switch the roles of the primary and standby database as follows:
 - 1) Issue the **TAKEOVER HADR** command on the standby database.
 - 2) Direct clients to the new primary database.
 - c. Repeat the same procedure in substep a to enable the HADR reads on standby feature on the new standby database.
5. Optional: If did not perform step 4 and you want to return to your original configuration, switch the roles of the primary and standby database as you did in step 2.
6. Optional: In an HADR environment, run **db2updv10** only on the primary database. After running the **db2updv10** command, you might have to restart the database for changes from **db2updv10** command to take effect. To perform a restart:
 - a. Restart the standby database by deactivating and reactivating it. The standby database is restarted to prevent the disruption of primary database service.
 - 1) Run the following command on the standby database:

```
DEACTIVATE  
db dbname
```

where *dbname* is the name of the standby database.

- 2) Run the following command on the standby database:

```
ACTIVATE  
db dbname
```

where *dbname* is the name of the standby database.

- b. Switch the roles of the primary and standby databases:

- 1) Run the following command on the standby database:

```
TAKOVER  
hadr on db dbname
```

where *dbname* is the name of the standby database.

- 2) Direct clients to the new primary database.

Note: The databases have switched roles. The primary database was previously the standby database and the standby database was previously the primary database.

- c. Restart the standby database (formerly the primary database), using the same method as in Step 1.
- d. Switch the roles of the primary and standby databases to return the database to their original roles. Switch the roles using the same method as in step 2.

High availability disaster recovery (HADR) support

Consider system requirements and feature limitations when designing your high availability database solution.

System requirements for DB2 high availability disaster recovery (HADR)

To achieve optimal performance with high availability disaster recovery (HADR), ensure that your system meets the following requirements for hardware, operating systems, and for the DB2 database system.

Recommendation: For better performance, use the same hardware and software for the system where the primary database resides and for the system where the standby database resides. If the system where the standby database resides has fewer resources than the system where the primary database resides, it is possible that the standby database will be unable to keep up with the transaction load generated by the primary database. This can cause the standby database to fall behind or the performance of the primary database to degrade. In a failover situation, the new primary database should have the resources to service the client applications adequately.

If you enable reads on standby and use the standby database to run some of your read-only workload, ensure that the standby has sufficient resources. An active standby requires additional memory and temporary table space usage to support transactions, sessions, and new threads as well as queries that involve sort and join operations.

Hardware and operating system requirements

Recommendation: Use identical host computers for the HADR primary and standby databases. That is, they should be from the same vendor and have the same architecture.

The operating system on the primary and standby databases should be the same version, including patches. When the rolling update procedure is used to upgrade the operating system, the operating system versions can be different on the primary and standby during the procedure. To minimize risks, plan ahead to have the procedure completed in a short time and try it out first in a test environment.

A TCP/IP interface must be available between the HADR host machines, and a high-speed, high-capacity network is recommended.

DB2 database requirements

The versions of the database systems for the primary and standby databases must be identical; for example, both must be either version 8 or version 9. During rolling updates, the modification level (for example, the fix pack level) of the database system for the standby database can be later than that of the primary database for a short while to test the new level. However, you should not keep this configuration for an extended period of time. The primary and standby databases will not connect to each other if the modification level of the database system for the primary database is later than that of the standby database. In order to use the reads on standby feature, both the primary and the standby databases need to be Version 9.7 Fix Pack 1.

The DB2 database software for both the primary and standby databases must have the same bit size (32 or 64 bit). Table spaces and their containers must be identical on the primary and standby databases. Properties that must be identical include the table space type (DMS or SMS), table space size, container path, container size, and container file type (raw device or file system). The amount of space allocated for log files should also be the same on both the primary and standby databases.

When you issue a table space statement on the primary database, such as CREATE TABLESPACE, ALTER TABLESPACE, or DROP TABLESPACE, it is replayed on the standby database. You must ensure that the devices involved are set up on both of the databases before you issue the table space statement on the primary database.

The primary and standby databases do not require the same database path. If relative container paths are used, the same relative path might map to different absolute container paths on the primary and standby databases.

Storage groups are fully supported by HADR, including replication of the CREATE STOGROUP, ALTER STOGROUP and DROP STOGROUP statements. Similar to table space containers, the storage paths must exist on both primary and standby.

The primary and standby databases must have the same database name. This means that they must be in different instances.

Redirected restore is not supported. That is, HADR does not support redirecting table space containers. However, database directory and log directory changes are supported. Table space containers created by relative paths will be restored to paths relative to the new database directory.

Buffer pool requirements

Since buffer pool operations are also replayed on the standby database, it is important that the primary and standby databases have the same amount of memory. If you are using reads on standby, you will need to configure the buffer pool on the primary so that the active standby can accommodate log replay and read applications.

Installation and storage requirements for high availability disaster recovery (HADR)

To achieve optimal performance with high availability disaster recovery (HADR), ensure that your system meets the following installation and storage requirements.

Installation requirements

For HADR, instance paths should be the same on the primary and the standby databases. Using different instance paths can cause problems in some situations, such as if an SQL stored procedure invokes a user-defined function (UDF) and the path to the UDF object code is expected to be on the same directory for both the primary and standby server.

Storage requirements

Storage groups are fully supported by HADR, including replication of the CREATE STOGROUP, ALTER STOGROUP and DROP STOGROUP statements. Similar to table space containers, the storage path must exist on both primary and standby. Symbolic links can be used to create identical paths. The primary and standby databases can be on the same computer. Even though their database storage starts at the same path, they do not conflict because the actual directories used have instance names embedded in them (since the primary and standby databases must have the same database name, they must be in different instances). The storage path is formulated as `storage_path_name/inst_name/dbpart_name/db_name/tbsp_name/container_name`.

Table spaces and their containers must be identical on the primary and standby databases. Properties that must be identical include: the table space type (DMS or SMS), table space size, container path, container size, and container file type (raw device or file system). Storage groups and their storage paths must be identical. This includes the path names and the amount of space on each that is devoted to each storage group. The amount of space allocated for log files should also be the same on both the primary and standby databases.

When you issue a table space statement on the primary database, such as CREATE TABLESPACE, ALTER TABLESPACE, or DROP TABLESPACE, it is replayed on the standby database. You must ensure that the devices involved are set up on both of the databases before you issue the table space statement on the primary database.

If the table space setup is not identical on the primary and standby databases, log replay on the standby database might encounter errors such as OUT OF SPACE or TABLE SPACE CONTAINER NOT FOUND. Similarly, if the storage groups's storage path setup is not identical on the primary and standby databases, log records associated with the CREATE STOGROUP or ALTER STOGROUP are not be replayed. As a result, the existing storage paths might prematurely run out of space on the standby system and automatic storage table spaces are not be able to increase in size. If any of these situations occurs, the affected table space is put in

rollforward pending state and is ignored in subsequent log replay. If a takeover operation occurs, the table space is not available to applications.

If the problem is noticed on the standby system prior to a takeover then the resolution is to re-establish the standby database while addressing the storage issues. The steps to do this include:

- Deactivating the standby database.
- Dropping the standby database.
- Ensuring the necessary file systems exist with enough free space for the subsequent restore and rollforward.
- Restoring the database at the standby system using a recent backup of the primary database (or, reinitialize using split mirror or flash copy with the **db2inidb** command). Storage group storage paths should not be redefined during the restore. Also, table space containers should not be redirected as part of the restore.
- Restarting HADR on the standby system.

However, if the problem is noticed with the standby database after a takeover has occurred (or if a choice was made to not address the storage issues until this time) then the resolution is based on the type of problem that was encountered.

If the database is enabled for automatic storage and space is not available on the storage paths associated with the standby database then follow these steps:

1. Make space available on the storage paths by extending the file systems, or by removing unnecessary non-DB2 files on them.
2. Perform a table space rollforward to the end of logs.

In the case where the addition or extension of containers as part of log replay could not occur, if the necessary backup images and log file archives are available, you might be able to recover the table space by first issuing the SET TABLESPACE CONTAINERS statement with the IGNORE ROLLFORWARD CONTAINER OPERATIONS option and then issuing the **ROLLFORWARD** command.

The primary and standby databases do not require the same database path. If relative container paths are used, the same relative path might map to different absolute container paths on the primary and standby databases. Consequently, if the primary and standby databases are placed on the same computer, all table space containers must be defined with relative paths so that they map to different paths for primary and standby.

HADR and Network Address Translation (NAT) support

NAT, which is supported in an HADR environment, is usually used for firewall and security because it hides the server's real address.

In an HADR setup, the local and remote host configurations on the primary and standby nodes are cross-checked to ensure they are correct. In a NAT environment, a host is known to itself by a particular IP address but is known to the other hosts by a different IP address. This behavior causes the HADR host cross-check to fail unless you set the **DB2_HADR_NO_IP_CHECK** registry variable to ON. Using this setting causes the host cross-check to be bypassed, enabling the primary and standby to connect in a NAT environment.

If you are not running in a NAT environment, use the default setting of OFF for the **DB2_HADR_NO_IP_CHECK** registry variable. Disabling the cross-check weakens the HADR validation of your configuration.

Considerations for HADR multiple standby mode

In a NAT environment with a multiple standby setup, each standby's settings for **hadr_local_host** and **hadr_local_svc** must still be listed in the primary's **hadr_target_list** or the primary does not accept the connection from that standby.

Normally, in multiple standby mode, on start up, a standby checks that its settings for **hadr_remote_host** and **hadr_remote_svc** are in its **hadr_target_list**, to ensure that on role switch, the old primary can become a new standby. In NAT scenarios, that check fails unless the **DB2_HADR_NO_IP_CHECK** registry variable is ON. Because this check is bypassed, the standby waits until it connects to the primary to check that the primary's **hadr_local_host** and **hadr_local_svc** are in the standby's **hadr_target_list**. The check still ensures role switch can succeed on this pair.

Note: If the **DB2_HADR_NO_IP_CHECK** registry variable is set to ON, the **hadr_remote_host** and **hadr_remote_svc** are not automatically updated.

In a multiple standby setup, **DB2_HADR_NO_IP_CHECK** should be set on all databases that might be making a connection to another database across a NAT boundary. If a database will never cross a NAT boundary to connect to another database (that is, if no such link is configured), then you should not set this registry variable on that database. When **DB2_HADR_NO_IP_CHECK** is set, it prevents a standby from automatically discovering the new primary after a takeover has occurred, and you have to manually reconfigure the standby to have it connect to the new primary.

Restrictions for High Availability Disaster Recovery (HADR)

To achieve optimal performance with High Availability Disaster Recovery (HADR), consider HADR restrictions when designing your high availability DB2 database solution.

The following list is a summary of High Availability Disaster Recovery (HADR) restrictions:

- HADR is not supported in a partitioned database environment.
- HADR is not supported in DB2 pureScale environments.
- The primary and standby databases must have the same operating system version and the same version of the DB2 database system, except for a short time during a rolling upgrade.
- The DB2 database system software on the primary and standby databases must be the same bit size (32 or 64 bit).
- Clients cannot connect to the standby database unless reads on standby are enabled. Reads on standby enables clients to connect to the active standby database and issue read-only queries.
- If reads on standby are enabled, operations on the standby database that write a log record are not permitted; only read clients can connect to the active standby database.
- If reads on standby are enabled, write operations that would modify database contents are not allowed on the standby database. Any asynchronous threads such as real-time statistics collection, Auto Index rebuild and utilities that

attempt to modify database objects will not be supported. Real-time statistics collection and Auto Index rebuild should not be running on the standby database.

- Log files are only archived by the primary database.
- The self tuning memory manager (STMM) can be run only on the current primary database. After the primary database is started or the standby database is converted to a primary database by takeover, the STMM EDU may not start until the first client connection comes in.
- Backup operations are not supported on the standby database.
- The **SET WRITE** command cannot be issued on the standby database
- Non-logged operations, such as changes to database configuration parameters and to the recovery history file, as well as LOB table columns that have the NOT LOGGED option, are not replicated to the standby database.
- Load operations with the **COPY NO** option specified are not supported.
- HADR does not support the use of raw I/O (direct disk access) for database log files. If HADR is started via the **START HADR** command, or the database is activated (restarted) with HADR configured, and raw logs are detected, the associated command will fail.
- Federated server does not fully support HADR in federated two phase commit (F2PC) scenarios. When a HADR database is configured as a federated database, it only supports F2PC with type-1 inbound connections.
- HADR does not support infinite logging.
- The system clock of the HADR primary database must be synchronized with the HADR standby database's system clock.

DB2 High availability disaster recovery (HADR) management

DB2 High availability disaster recovery (HADR) management involves configuring and maintaining the status of your HADR system.

Managing HADR includes such tasks as:

- Cataloging an HADR database.
- “Initializing high availability disaster recovery (HADR)” on page 477
- Checking or altering database configuration parameters related to HADR.
- “Switching database roles in high availability disaster recovery (HADR)” on page 504
- “Performing an HADR failover operation” on page 501
- “Monitoring high availability disaster recovery (HADR) environments” on page 505
- “Stopping DB2 High Availability Disaster Recovery (HADR)” on page 507

You can manage HADR using the following methods:

- Command line processor
- DB2 administrative API
- Task assistants for managing HADR in IBM Data Studio Version 3.1 or later.

Related information:

- Administering databases with task assistants

DB2 High Availability Disaster Recovery (HADR) commands

The DB2 High Availability Disaster Recovery (HADR) feature provides complex logging, failover, and recovery functionality for DB2 high availability database solutions.

Despite the complexity of the functionality HADR provides, there are only a few actions you need to directly command HADR to perform: starting HADR; stopping HADR; and causing the standby database to take over as the primary database.

There are three high availability disaster recover (HADR) commands used to manage HADR:

- **START HADR**
- **STOP HADR**
- **TAKEOVER HADR**

To invoke these commands, use the command line processor or the administrative API.

Issuing the **START HADR** command with either the AS PRIMARY or AS STANDBY option changes the database role to the one specified if the database is not already in that role. This command also activates the database, if it is not already activated.

The **STOP HADR** command changes an HADR database (either primary or standby) into a standard database. Any database configuration parameters related to HADR remain unchanged so that the database can easily be reactivated as an HADR database.

The **TAKEOVER HADR** command, which you can issue on the standby database only, changes the standby database to a primary database. When you do not specify the BY FORCE option, the primary and standby databases switch roles. When you do specify the BY FORCE option, the standby database unilaterally switches to become the primary database. In this case, the standby database attempts to stop transaction processing on the old primary database. However, there is no guarantee that transaction processing will stop. Use the BY FORCE option to force a takeover operation for failover conditions only. To whatever extent possible, ensure that the current primary has definitely failed, or shut it down yourself, prior to issuing the **TAKEOVER HADR** command with the BY FORCE option.

HADR database role switching

A database can be switched between primary and standard roles dynamically and repeatedly. When the database is either online or offline, you can issue both the **START HADR** command with the AS PRIMARY option and the **STOP HADR** command.

You can switch a database between standby and standard roles statically. You can do so repeatedly only if the database remains in rollforward pending state. You can issue the **START HADR** command with the AS STANDBY option to change a standard database to standby while the database is offline and in rollforward pending state. Use the **STOP HADR** command to change a standby database to a standard database while the database is offline. The database remains in rollforward pending state after you issue the **STOP HADR** command. Issuing a subsequent **START HADR** command with the AS STANDBY option returns the database to standby. If you issue the **ROLLFORWARD DATABASE** command with the STOP option after stopping HADR on a standby database, you cannot bring it back to standby. Because the database is out of rollforward pending state, you can use it as a standard database. This is

referred to as taking a snapshot of the standby database. After changing an existing standby database into a standard database, consider creating a new standby database for high availability purposes.

To switch the role of the primary and standby databases, perform a takeover operation without using the BY FORCE option.

To change the standby to primary unilaterally (without changing the primary to standby), use forced takeover. Subsequently, you might be able to reintegrate the old primary as a new standby.

HADR role is persistent. Once an HADR role is established, it remains with the database, even through repeated stopping and restarting of the DB2 instance or deactivation and activation of the DB2 database.

Starting the standby is asynchronous

When you issue the **START HADR** command with the AS STANDBY option, the command returns as soon as the relevant engine dispatchable units (EDUs) are successfully started. The command does not wait for the standby to connect to the primary database. In contrast, the primary database is not considered started until it connects to a standby database (with the exception of when the **START HADR** command is issued on the primary with the BY FORCE option). If the standby database encounters an error, such as the connection being rejected by the primary database, the **START HADR** command with the AS STANDBY option might have already returned successfully. As a result, there is no user prompt to which HADR can return an error indication. The HADR standby will write a message to the DB2 diagnostic log and shut itself down. You should monitor the status of the HADR standby to ensure that it successfully connects with the HADR primary.

Replay errors, which are errors that the standby encounters while replaying log records, can also bring down the standby database. These errors might occur, for example, when there is not enough memory to create a buffer pool, or if the path is not found while creating a table space. You should continuously monitor the status of the standby database.

Do not run HADR commands from a client using a database alias enabled for client reroute

When automatic client reroute is set up, the database server has a predefined alternate server so that client applications can switch between working with either the original database server or the alternative server with only minimal interruption of the work. In such an environment, when a client connects to the database via TCP, the actual connection can go to either the original database or to the alternate database. HADR commands are implemented to identify the target database through regular client connection logic. Consequently, if the target database has an alternative database defined, it is difficult to determine the database on which the command is actually operating. Although an SQL client does not need to know which database it is connecting to, HADR commands must be applied on a specific database. To accommodate this limitation, HADR commands should be issued locally on the server machine so that client reroute is bypassed (client reroute affects only TCP/IP connections).

HADR commands must be run on a server with a valid license

The **START HADR**, **STOP HADR**, and **TAKEOVER HADR** commands require that a valid HADR license has been installed on the server where the command is executed. If the license is not present, these commands will fail and return a command-specific error code (SQL1767N, SQL1769N, or SQL1770N, respectively) along with a reason code of 98. To correct the problem, either install a valid HADR license using **db2licm**, or install a version of the server that contains a valid HADR license as part of its distribution.

Initializing high availability disaster recovery (HADR)

Use this procedure to set up and initialize a DB2 high availability disaster recovery (HADR) primary database and one standby database.

About this task

HADR can be initialized through the command line processor (CLP), or by calling the db2HADRStart API.

Procedure

To use the CLP to initialize HADR on your system for the first time:

1. Determine the host name, host IP address, and the service name or port number for each of the HADR databases.
If a host has multiple network interfaces, ensure that the HADR host name or IP address maps to the intended one. You need to allocate separate HADR ports in /etc/services for each protected database. These cannot be the same as the ports allocated to the instance. The host name can only map to one IP address.

Note: The instance names for the primary and standby databases do not have to be the same.

2. Configure the recommended index logging and re-creation settings on the intended primary, by issuing the following command:

```
"UPDATE DB CFG FOR dbname USING  
    LOGINDEXBUILD ON  
    LOGARCHMETH1      method"
```

3. Create the standby database by restoring a backup image or by initializing a split mirror, based on the existing database that is to be the primary.

In the following example, the **BACKUP DATABASE** and **RESTORE DATABASE** commands are used to initialize a standby database. In this case, an NFS mounted file system is accessible at both sites.

Issue the following command at the primary database:

```
BACKUP DB dbname TO /nfs1/backups/db2/dbname
```

If the database already exists on the standby instance, drop it first for a clean start. Files from the existing database can interfere with HADR operation. For example, left over log files can lead the standby onto a log chain not compatible with the primary. Issue the following command to drop the database:

```
DROP DB dbname
```

Issue the following command at the standby database:

```
RESTORE DB dbname FROM /nfs1/backups/db2/dbname
```

The following example illustrates how to use the **db2inidb** utility to initialize the standby database using a split mirror of the primary database. This procedure is an alternative to the backup and restore procedure illustrated previously.

Issue the following command at the standby database:

```
DB2INIDB dbname AS STANDBY
```

Note:

- a. The database names for the primary and standby databases must be the same.
- b. Do not issue the **ROLLFORWARD DATABASE** command on the standby database after the restore operation or split mirror initialization. The results of using a rollforward operation might differ slightly from replaying the logs using HADR on the standby database. If the databases are not identical, attempts to start the standby will fail.
- c. When creating the standby database using the **RESTORE DATABASE** command, ensure that the standby remains in rollforward-pending or rollforward-in-progress mode. This means that you cannot issue the **ROLLFORWARD DATABASE** command with either the **COMPLETE** option or the **STOP** option. An error will be returned if the **START HADR** command with the **AS STANDBY** option is attempted on the database after rollforward is stopped.
- d. The following **RESTORE DATABASE** command options should be avoided when setting up the standby database: **TABLESPACE**, **INTO**, **REDIRECT**, and **WITHOUT ROLLING FORWARD**.
- e. When setting up the standby database using the **db2inidb** utility, do not use the **SNAPSHOT** or **MIRROR** options. You can specify the **RELOCATE USING** option to change one or more of the following configuration attributes: instance name, log path, and database path. However, you must not change the database name or the table space container paths.

4. On each of the databases, set the **hadr_local_host**, **hadr_local_svc**, and **hadr_syncmode** configuration parameters:

```
"UPDATE DB CFG FOR dbname USING  
    HADR_LOCAL_HOST    hostname  
    HADR_LOCAL_SVC     servicename  
    HADR_SYNCMODE      syncmode"
```

The configuration parameters in this step, step 5, and step 6 on page 479 must be set after the standby databases has been created. If they are set prior to creating the standby database, the settings on the standby database will reflect what is set on the primary database.

Note: This is a generic HADR setup; for more advanced configuration options and settings, see the related links.

5. Optional: Set the **hadr_target_list** configuration parameter on the standby and the primary:

```
UPDATE DB CFG FOR dbname USING  
    HADR_TARGET_LIST   principalhostname:principalservicename1
```

This is an optional, but recommended, step if you are only using one standby database. If you set the **hadr_target_list** parameter, you can add additional

1.

standby database dynamically. You can also take advantage of the autoconfiguration behavior and specify a different synchronization mode on the standby.

6. On each of the databases, set the **hadr_remote_host**, **hadr_remote_svc**, and **hadr_remote_inst** configuration parameters.

On the primary, set the parameters to the corresponding values on the standby by issuing the following command:

```
"UPDATE DB CFG FOR dbname USING  
    HADR_REMOTE_HOST  principalhostname  
    HADR_REMOTE_SVC   principalservicename  
    HADR_REMOTE_INST  principalinstname"
```

On the standby, set the parameters to the corresponding values on the primary by issuing the following command:

```
"UPDATE DB CFG FOR dbname USING  
    HADR_REMOTE_HOST  primaryhostname  
    HADR_REMOTE_SVC   primarieservicename  
    HADR_REMOTE_INST  primaryinstname"
```

If you have configured **hadr_target_list**, the values for these parameters are automatically given the proper values if they were set incorrectly. However, explicitly setting them to the correct values makes correct values available immediately. These values are helpful for the IBM Tivoli System Automation for Multiplatforms (SA MP) software, which might require the **hadr_remote_host** value to construct the resource name.

7. Connect to the standby instance and start HADR on the standby database, as in the following example:

```
START HADR ON DB dbname AS STANDBY
```

Note: Usually, the standby database is started first. If you start the primary database first, this startup procedure will fail if the standby database is not started within the time period specified by the **hadr_timeout** database configuration parameter.

After the standby starts, it enters *local catchup* state in which locally available log files are read and replayed. After it has replayed all local logs, it enters *remote catchup pending* state.

8. Connect to the primary instance and start HADR on the primary database, as in the following example:

```
START HADR ON DB dbname AS PRIMARY
```

9. Use a monitoring interface (the MON_GET_HADR table function or the **db2pd** command with the **-hadr** option) to verify that the HADR pair enters the expected state. After the primary starts, the standby enters *remote catchup* state in which receives log pages from the primary and replays them. After it has replayed all log files that are on the disk of the primary database machine, both databases enter *peer* state (unless you have chosen SUPERASYNC as the synchronization mode).

Initializing a standby database

One strategy for making a database solution highly available is maintaining a primary database to respond to user application requests, and a secondary or standby database that can take over database operations for the primary database if the primary database fails.

Initializing the standby database entails copying the primary database to the standby database.

Procedure

There are several ways to initialize the standby database. For example:

- Use disk mirroring to copy the primary database, and use DB2 database suspended I/O support to split the mirror to create the second database.
- Create a backup image of the primary database and recovery that image to the standby database.
- Use SQL replication to capture data from the primary database and apply that data to the standby database.

What to do next

After initializing the standby database, you must configure your database solution to synchronize the primary database and standby database so the standby database can take over for the primary database if the primary database fails.

Using a split mirror as a standby database

Use the following procedure to create a split mirror of a database for use as a standby database outside of a DB2 pureScale environment.

If a failure occurs on the primary database and it becomes inaccessible, you can use the standby database to take over for the primary database.

About this task

If the primary database was configured for log archiving, the standby database will share the same log archiving configuration. If the log archiving destination is accessible to the standby database, the standby database will automatically retrieve log files from it during rollforward operations. However, once the database is brought out of the rollforward pending state, the standby database will attempt to archive log files to the same location used by the primary database. While the standby database will initially use a different log chain from the primary database, the primary database could eventually use the same log chain value as the standby database. This could cause the primary database to archive log files on top of the log files archived by the standby database, or vice versa , and can affect the recoverability of both databases. You should change the log archiving destination for the standby database to be different from that of the primary database to avoid recoverability issues.

Procedure

To use a split mirror as a standby database:

1. Connect to the primary database using the following command:
`db2 connect to db_name`
2. Suspend the I/O write operations on the primary database using the following command:
`db2 set write suspend for database`

Note: While the database is in suspended state, you should not be running other utilities or tools. You should only be making a copy of the database. You can optionally use the FLUSH BUFFERPOOLS ALL statement before issuing **SET WRITE SUSPEND** to minimize the recovery time of the standby database.

3. Create one or multiple split mirrors from the primary database using appropriate operating system-level and storage-level commands.

Note:

- Ensure that you copy the entire database directory, including the volume directory. You must also copy the log directory and any container directories that exist outside the database directory. To gather this information, refer to the DBPATHS administrative view, which shows all the files and directories of the database that need to be split.
 - If you specified the EXCLUDE LOGS with the **SET WRITE** command, do not include the log files in the copy.
4. Resume the I/O write operations on the primary database using the following command:
- ```
db2 set write resume for database
```
5. Catalog the mirrored database on the secondary system.

**Note:** By default, a mirrored database cannot exist on the same system as the primary database. It must be located on a secondary system that has the same directory structure and uses the same instance name as the primary database. If the mirrored database must exist on the same system as the primary database, you can use the **db2relocatedb** utility or the RELOCATE USING option of the **db2inidb** command to accomplish this.

6. Start the database instance on the secondary system using the following command:

```
db2start
```

7. Initialize the mirrored database on the secondary system by placing it in rollforward pending state using the following command:

```
db2inidb <database_alias> as standby
```

If required, specify the RELOCATE USING option of the **db2inidb** command to relocate the standby database:

```
db2inidb <database_alias> as standby relocate using relocatedbcfg.txt
```

where the **relocatedbcfg.txt** file contains the information required to relocate the database.

**Note:** You can take a full database backup using the split mirror if you have DMS table spaces (database managed space) or automatic storage table spaces. Taking a backup using the split mirror reduces the overhead of taking a backup on the production database. Such backups are considered to be online backups and will contain in-flight transactions, but you cannot include log files from the standby database. When such a backup is restored, you must rollforward to at least the end of the backup before you can issue a **ROLLFORWARD** command with the **STOP** option. Because the backup will not contain any log files, the log files from the primary database that were in use at the time the **SET WRITE SUSPEND** command was issued must be available or the rollforward operation will not be able to reach the end of the backup.

8. Make the archived log files from the primary database available to the standby database either by configuring the log archiving parameters on the standby database or by shipping logs to the standby database.
9. Rollforward the database to the end of the logs or to a point-in-time.
10. Continue retrieving log files and rollforwarding the database through the logs until you reach the end of the logs or the point-in-time required for the standby database.
11. Bring the standby database online by issuing the **ROLLFORWARD** command with the **STOP** option specified.

**Note:**

- The logs from the primary database cannot be applied to the mirrored database after it has been taken out of rollforward pending state.
- If the primary database was configured for log archiving, the standby database will share the same log archiving configuration. If the log archiving destination is accessible to the standby database, the standby database will automatically retrieve log files from it while rollforward is being performed. However, once the database is brought out of rollforward pending state, the standby database will attempt to archive log files to the same location used by the primary database. Although the standby database will initially use a different log chain from the primary database, there is nothing to prevent the primary database from eventually using the same log chain value as the standby database. This may cause the primary database to archive log files on top of the log files archived by the standby database, or vice versa. This could affect the recoverability of both databases. You should change the log archiving destination for the standby database to be different from that of the primary database to avoid these issues.

## Using a split mirror as a standby database in a DB2 pureScale environment

Use the following procedure to create a split mirror of a database for use as a standby database in a DB2 pureScale environment. If a failure occurs on the primary database and it becomes inaccessible, you can use the standby database to take over for the primary database.

### About this task

If the primary database was configured for log archiving, the standby database will share the same log archiving configuration. If the log archiving destination is accessible to the standby database, the standby database will automatically retrieve log files from it during rollforward operations. However, once the database is brought out of the rollforward pending state, the standby database will attempt to archive log files to the same location used by the primary database. While the standby database will initially use a different log chain from the primary database, the primary database could eventually use the same log chain value as the standby database. This could cause the primary database to archive log files on top of the log files archived by the standby database, or vice versa , and can affect the recoverability of both databases. You should change the log archiving destination for the standby database to be different from that of the primary database to avoid recoverability issues.

### Procedure

To use a split mirror as a standby database:

1. Connect to the primary database using the following command:  
`db2 connect to <db_name>`
2. Configure the General Parallel File System (GPFS) on the secondary cluster by extracting and importing the primary cluster's settings. On the primary cluster, run the following GPFS command:  
`mmfsctl <filesystem> syncFSconfig -n <remotenodefile>`

where `<remotenodefile>` is the list of hosts in the secondary cluster.

3. List the cluster manager domain using the following command:

```
db2cluster -cm -list -domain
```

4. Stop the cluster manager on each host in the cluster using the following command:

```
db2cluster -cm -stop -host <host> -force
```

**Note:** The last host which you shut down must be the host from which you are issuing this command.

5. Stop the GPFS cluster on the secondary system using the following command:

```
db2cluster -cfs -stop -all
```

6. Suspend the I/O write operations on the primary database using the following command:

```
db2 set write suspend for database
```

**Note:** While the database is in suspended state, you should not be running other utilities or tools. You should only be making a copy of the database. You can optionally flush all buffer pools before issuing **SET WRITE SUSPEND** to minimize the recovery window. This can be achieved using the FLUSH BUFFERPOOLS ALL statement.

7. Determine which file systems must be suspended and copied using the following command:

```
db2cluster -cfs -list -filesystem
```

8. Suspend each GPFS file system that contains data or log data using the following command:

```
/usr/lpp/mmfs/bin/mmfsc1 <filesystem> suspend
```

where *<filesystem>* represents a file system that contains data or log data.

**Note:** While the GPFS file systems are suspended, both read and write operations are blocked. You should only be performing the split mirror operations during this period to minimize the amount of time that read operations are blocked.

9. Create one or multiple split mirrors from the primary database using appropriate operating system-level and storage-level commands.

**Note:**

- Ensure that you copy the entire database directory, including the volume directory. You must also copy the log directory and any container directories that exist outside the database directory. To gather this information, refer to the DBPATHS administrative view, which shows all the files and directories of the database that need to be split.
- If you specified the EXCLUDE LOGS with the **SET WRITE** command, do not include the log files in the copy.

10. Resume the GPFS file systems that were suspended using the following command for each suspended file system:

```
/usr/lpp/mmfs/bin/mmfsc1 <filesystem> resume
```

where *filesystem* represents a suspended file system that contains data or log data.

11. Resume the I/O write operations on the primary database using the following command:

```
db2 set write resume for database
```

12. Start the GPFS cluster on the secondary system using the following command:

- ```
db2cluster -cfs -start -all
```
13. Start the cluster manager using the following command
`db2cluster -cm -start -domain <domain>`
 14. Catalog the mirrored database on the secondary system.

Note: By default, a mirrored database cannot exist on the same system as the primary database. It must be located on a secondary system that has the same directory structure and uses the same instance name as the primary database. If the mirrored database must exist on the same system as the primary database, you can use the **db2relocatedb** utility or the **RELOCATE USING** option of the **db2inidb** command to accomplish this.

15. Start the database instance on the secondary system using the following command:
`db2start`
16. Initialize the database on the secondary system by placing it in rollforward pending state:
`db2inidb <database_alias> as standby`
If required, specify the **RELOCATE USING** option of the **db2inidb** command to relocate the database:
`db2inidb database_alias as standby relocate using relocatedbcfg.txt`

where `relocatedbcfg.txt` contains the information required to relocate the database.

Note: You can take a full database backup using the split mirror if you have DMS table spaces (database managed space) or automatic storage table spaces. Taking a backup using the split mirror reduces the overhead of taking a backup on the production database. Such backups are considered to be online backups and will contain in-flight transactions, but you cannot include log files from the standby database. When such a backup is restored, you must rollforward to at least the end of the backup before you can issue a **ROLLFORWARD STOP** command. Because the backup will not contain any log files, the log files from the primary database that were in use at the time the **SET WRITE SUSPEND** command was issued must be available or the rollforward operation will not be able to reach the end of the backup.

17. Make the archived log files from the primary database available to the standby database either by configuring the log archiving parameters on the standby database or by shipping logs to the standby database.
18. Rollforward the database to the end of the logs or to a point-in-time.

Note: When executing rollforward operations, you might encounter SQL1273 errors. These errors are expected if some of the log files were not copied from the primary system when the database was split or if one member generates log files faster than other members. SQL1273 is generated in some cases when the rollforward operation must stop to preserve data consistency because the contents of the log files depends on the contents of unavailable log files from other members. If the standby database is configured to retrieve log files archived by the primary database, you can either wait for the primary system to archive the necessary log file or you can use the **ARCHIVE LOG** command on the primary system to force the log file to be archived. Otherwise, you must ship the required log files to the standby database. After the necessary log file is available on the standby database, the rollforward operation can read further ahead in the logs, although SQL1273 might be encountered again if some members are still generating log files faster than other members. For

more information, see the “Disaster recovery and high availability through log shipping in a DB2 pureScale environment” section of the “Backup and restore operations in a DB2 pureScale environment” Information Center topic.

19. Continue the rollforward operation through the logs until you reach the end of the logs or the point-in-time required for the standby database, shipping new log files to the standby database if required.
20. Bring the standby database online by issuing the **ROLLFORWARD DATABASE** command with the ST0P option specified.

Note:

- The logs from the primary database cannot be applied to the mirrored database once it has been taken out of rollforward pending state.
- If the primary database was configured for log archiving, the standby database will share the same log archiving configuration. If the log archiving destination is accessible to the standby database, the standby database will automatically retrieve log files from it while rollforward is being performed. However, once the database is brought out of rollforward pending state, the standby database will attempt to archive log files to the same location used by the primary database. Although the standby database will initially use a different log chain from the primary database, there is nothing to prevent the primary database from eventually using the same log chain value as the standby database. This may cause the primary database to archive log files on top of the log files archived by the standby database, or vice versa. This could affect the recoverability of both databases. You should change the log archiving destination for the standby database to be different from that of the primary database to avoid these issues.

Database configuration for high availability disaster recovery (HADR)

You can use database configuration parameters to help achieve optimal performance with DB2 HADR.

In most cases, you should use the same database configuration parameter settings and database manager configuration parameter settings on the systems where the primary and standby databases are located. If the settings for the configuration parameters on the standby database are different from the settings on the primary, the following problems might occur:

- Error messages might be returned for the standby database while the log files that were shipped from the primary database are being replayed.
- After a takeover operation, the new primary database might be unable to handle the workload, resulting in performance problems or in applications receiving error messages that they did not receive when they were connected to the original primary database.

Changes to the configuration parameters on the primary database are not automatically propagated to the standby database. You must manually make changes on the standby database. For dynamic configuration parameters, changes take effect without the need to shut down and restart the database management system (DBMS) or the database. For non-dynamic configuration parameters, changes take effect after the standby database is restarted.

Following are sections on specific configuration topics for HADR:

- “Size of log files configuration parameter on the standby database” on page 486

- “Database configuration parameter autorestart”
- “Log receive buffer size on a standby database” on page 487
- “Load operations and HADR” on page 487
- “DB2_HADR_PEER_WAIT_LIMIT registry variable” on page 488
- “HADR configuration parameters” on page 489

Size of log files configuration parameter on the standby database

One exception to the configuration parameter behavior that is described in the previous paragraph is the behavior of the **logfilsiz** database configuration parameter. Although the value of this parameter is not replicated to the standby database, to guarantee identical log files on both databases, the setting for the **logfilsiz** configuration parameter on the standby is ignored. Instead, the database creates local log files whose sizes match the size of the log files on the primary database.

After a takeover, the original standby (new primary) uses the **logfilsiz** parameter value that you set on the original primary until you restart the database. At that point, the new primary reverts to using the value that you set locally. In addition, the current log file is truncated and any pre-created log files are resized on the new primary.

If the databases keep switching roles as a result of a non-forced takeover and neither database is deactivated, the log file size that is used is always the one from the original primary database. However, if there is a deactivation and then a restart on the original standby (new primary), the new primary uses the log file size that you configured locally. This log file size continues to be used if the original primary takes over again. Only after a deactivation and restart on the original primary would the log file size revert to the settings on the original primary.

Database configuration parameter autorestart

The recommended configuration for the **autorestart** parameter on HADR systems is ON. If the **autorestart** parameter is set to OFF, and the server fails, your response depends on whether or not you want to restart or fail over to the standby:

- If you want to restart, run the **RESTART DATABASE** command manually. If the restart fails, perform failover.
- If you want to fail over, perform the following steps:
 1. Shut down the old primary to prevent a “split brain”. Do this by either stopping the DB2 instance or powering off the host machine. If the server is not accessible for administration, fence it off from clients by disabling the client/server network.

Note: Deactivating the database is not sufficient because client connections can bring it back online. If it failed in a consistent state, then even if the **autorestart** parameter is set to OFF, this does not prevent client connections from bringing it back online.

2. After shutting down old primary, issue the **TAKEOVER HADR** command with the BY FORCE option on the standby.

Log receive buffer size on a standby database

By default, the log receive buffer size on a standby database is two times the value that you specify for the **logbufsz** configuration parameter on the primary database. This size might not be sufficient. For example, consider what might happen when the HADR synchronization mode is set to ASYNC and the primary and standby databases are in peer state. If the primary database is also experiencing a high transaction load, the log receive buffer on the standby database might fill to capacity, and the log shipping operation from the primary database might stall. To manage these temporary peaks, you can make either of the following configuration changes:

- Increase the size of the log receive buffer on the standby database by modifying the value of the **DB2_HADR_BUF_SIZE** registry variable.
- Enable log spooling on a standby database by setting the **hadr_spool_limit** configuration parameter.

Load operations and HADR

If you issue the **LOAD** command on the primary database with the **COPY YES** parameter, the command executes on the primary database, and the data is replicated to the standby database if the load copy can be accessed through the path or device that is specified by the command. If load copy data cannot be accessed from the standby database, the table space in which the table is stored is marked invalid on the standby database. Any future log records that pertain to this table space are skipped. To ensure that the load operation can access the load copy on the standby database, use a shared location for the output file from the **COPY YES** parameter. Alternatively, you can deactivate the standby database while performing the load on the primary, place a copy of the output file in the standby path, and then activate the standby database.

If you issue the **LOAD** command with the **NONRECOVERABLE** parameter on the primary database, the command executes on the primary database, and the table on the standby database is marked invalid. Any future log records that pertain to this table are skipped. You can issue the **LOAD** command with the **COPY YES** and **REPLACE** parameters to bring the table back, or you can drop the table to recover the space.

Note: You cannot bring a table back using the **LOAD** command with the **COPY YES** and **REPLACE** options if the table has one of the following characteristics:

- The table was created with the NOT LOGGED INITIALLY attribute.
- The table is a multidimensional clustered (MDC) table.
- The table has compression dictionaries.
- The table has XML columns.

Because a load operation with the **COPY NO** parameter is not supported with HADR, the operation is automatically converted to a load operation with the **NONRECOVERABLE** parameter. To enable a load operation with the **COPY NO** parameter to be converted to a load operation with the **COPY YES** parameter, set the **DB2_LOAD_COPY_NO_OVERRIDE** registry variable on the primary database. This registry variable is ignored on the standby database. Ensure that the device or directory that you specify for the primary database can be accessed by the standby database by using the same path, device, or load library.

If you are using the Tivoli Storage Manager (TSM) software to perform a load operation with the **COPY YES** parameter, you might have to set the **vendoropt**

configuration parameter on the primary and standby databases. Depending on how you configured TSM, the values on the primary and standby databases might not be the same. Also, when using TSM to perform a load operation with the **COPY YES** parameter, you must issue the **db2adut1** command with the **GRANT** parameter to give the standby database read access to the files that are loaded.

If table data is replicated by a load operation with the **COPY YES** parameter, the indexes are replicated as follows:

- If you specify the REBUILD indexing mode option with the **LOAD** command and the LOG INDEX BUILD table attribute is set to ON (using the ALTER TABLE statement), or if it is set to NULL and the **logindexbuild** database configuration parameter is set to ON, the primary database includes the rebuilt index object (that is, all of the indexes defined on the table) in the copy file to enable the standby database to replicate the index object. If the index object on the standby database is marked invalid before the load operation, it becomes usable again after the load operation as a result of the index rebuild.
- If you specify the INCREMENTAL indexing mode option with the **LOAD** command and the LOG INDEX BUILD table attribute is set to ON (using the ALTER TABLE statement), or if it is set to NULL and the **logindexbuild** database configuration parameter on the primary database is set to ON, the index object on the standby database is updated only if it is not marked invalid before the load operation. Otherwise, the index is marked invalid on the standby database.

DB2_HADR_PEER_WAIT_LIMIT registry variable

Restriction: In multiple standby mode, none of this section applies to the auxiliary standbys because they are in SUPERASYNC synchronization mode, so they do not ever enter peer state.

If you set the **DB2_HADR_PEER_WAIT_LIMIT** registry variable, the HADR primary database breaks out of peer state if logging on the primary database has been blocked for the specified number of seconds because of log replication to the standby. When this limit is reached, the primary database breaks the connection to the standby database. If you disable the peer window by setting the **hadr_peer_window** configuration parameter to 0, the primary enters the disconnected state, and logging resumes. If you enable the peer window, the primary database enters disconnected peer state, in which logging continues to be blocked. The primary leaves disconnected peer state upon reconnection or peer window expiration. Logging resumes after the primary leaves disconnected peer state.

Note: If you set **DB2_HADR_PEER_WAIT_LIMIT**, use a minimum value of 10 to avoid triggering false alarms.

Honoring peer window transition when a database breaks out of peer state ensures peer window semantics for safe takeover in all cases. If the primary fails during the transition, normal peer window protection still applies: safe takeover from the standby if it is still in disconnected peer state.

On the standby side, after disconnection, the database continues replaying already received logs. After the received logs have been replayed, the standby reconnects to the primary. After replaying the received logs, the standby reconnects to the primary. Upon reconnection, normal state transition follows: first remote catchup state, then peer state.

Relationship to hadr_timeout database configuration parameter

The **hadr_timeout** database configuration parameter does not break the primary out of peer state if the primary keeps receiving heartbeat messages from the standby while blocked. The **hadr_timeout** database configuration parameter specifies a timeout value for the HADR network layer. An HADR database breaks the connection to its partner database if it has not received any message from its partner for the period that is specified by the **hadr_timeout** configuration parameter. The timeout does not control timeout for higher-layer operations such as log shipping and ack (acknowledgement) signals. If log replay on the standby database is stuck on a large operation such as load or reorganization, the HADR component still sends heartbeat messages to the primary database on the normal schedule. In such a scenario, the primary is blocked as long as the standby replay is blocked unless you set the **DB2_HADR_PEER_WAIT_LIMIT** registry variable.

The **DB2_HADR_PEER_WAIT_LIMIT** registry variable unblocks primary logging regardless of connection status. Even if you do not set the **DB2_HADR_PEER_WAIT_LIMIT** registry variable, the primary always breaks out of peer state when a network error is detected or the connection is closed, possibly as result of the **hadr_timeout** configuration parameter.

HADR configuration parameters

Some HADR configuration parameters are static, such as **hadr_local_host** and **hadr_remote_host**. Static parameters are loaded on database startup, and changes are ignored during run time. HADR parameters are also loaded when the **START HADR** command completes. On the primary database, HADR can be started and stopped dynamically, with the database remaining online. Thus, one way to refresh the effective value of an HADR configuration parameter without shutting down the database is to stop and restart HADR. In contrast, the **STOP HADR** brings down the database on the standby, so the standby's parameters cannot be refreshed with database online.

Host name parameters and service and port name parameters (single standby mode) An HADR pair has five interrelated configuration parameters that you should set:

- **hadr_local_host**
- **hadr_remote_host**
- **hadr_local_svc**
- **hadr_remote_svc**
- **hadr_remote_inst**

TCP connections are used for communication between the primary and standby databases. The “local” parameters specify the local address and the “remote” parameters specify the remote address. A primary database listens on its local address for new connections. A standby database that is not connected to a primary database retries connection to its remote address.

The standby database also listens on its local address. In some scenarios, another HADR database can contact the standby database on this address and send it messages.

Unless the **HADR_NO_IP_CHECK** registry variable is set, HADR does the following cross-checks of local and remote addresses on connection:

my local address = your remote address

and

my remote address = your local address

The check is done using the IP address and port number, rather than the literal string in the configuration parameters. You need to set the **HADR_NO_IP_CHECK** registry variable in NAT (Network Address Translation) environment to bypass the check.

You can configure an HADR database to use either IPv4 or IPv6 to locate its partner database. If the host server does not support IPv6, you must use IPv4. If the server supports IPv6, whether the database uses IPv4 or IPv6 depends upon the format of the address that you specify for the **hadr_local_host** and **hadr_remote_host** configuration parameters. The database attempts to resolve the two parameters to the same IP format and use IPv6 when possible. The following table shows how the IP mode is determined for IPv6-enabled servers:

IP mode used for hadr_local_host parameter	IP mode used for hadr_remote_host parameter	IP mode used for HADR communications
IPv4 address	IPv4 address	IPv4
IPv4 address	IPv6 address	Error
IPv4 address	host name, maps to IPv4 only	IPv4
IPv4 address	host name, maps to IPv6 only	Error
IPv4 address	host name, maps to IPv4 and v6	IPv4
IPv6 address	IPv4 address	Error
IPv6 address	IPv6 address	IPv6
IPv6 address	host name, maps to IPv4 only	Error
IPv6 address	host name, maps to IPv6 only	IPv6
IPv6 address	host name, maps to IPv4 and IPv6	IPv6
hostname, maps to IPv4 only	IPv4 address	IPv4
hostname, maps to IPv4 only	IPv6 address	Error
hostname, maps to IPv4 only	hostname, maps to IPv4 only	IPv4
hostname, maps to IPv4 only	hostname, maps to IPv6 only	Error
hostname, maps to IPv4 only	hostname, maps to IPv4 and IPv6	IPv4
hostname, maps to IPv6 only	IPv4 address	Error
hostname, maps to IPv6 only	IPv6 address	IPv6
hostname, maps to IPv6 only	hostname, maps to IPv4 only	Error
hostname, maps to IPv6 only	hostname, maps to IPv6 only	IPv6
hostname, maps to IPv6 only	hostname, maps to IPv4 and IPv6	IPv6
hostname, maps to IPv4 and IPv6	IPv4 address	IPv4

IP mode used for hadr_local_host parameter	IP mode used for hadr_remote_host parameter	IP mode used for HADR communications
hostname, maps to IPv4 and IPv6	IPv6 address	IPv6
hostname, maps to IPv4 and IPv6	hostname, maps to IPv4 only	IPv4
hostname, maps to IPv4 and IPv6	hostname, maps to IPv6 only	IPv6
hostname, maps to IPv4 and IPv6	hostname, maps to IPv4 and IPv6	IPv6

The primary and standby databases can make HADR connections only if they use the same IPv4 or IPv6 format. If one server is IPv6 enabled (but also supports IPv4) and the other server supports IPv4 only, at least one of the **hadr_local_host** and **hadr_remote_host** parameters on the IPv6 server must specify an IPv4 address to force database on this server to use IPv4.

You can set the HADR local service and remote service parameters (**hadr_local_svc** and **hadr_remote_svc**) to either a port number or a service name. The values that you specify must map to ports that are not being used by any other service, including other DB2 components or other HADR databases. In particular, you cannot set either parameter value to the TCP/IP port that is used by the server to await communications from remote clients (the value of the **svccename** database manager configuration parameter) or the next port (the value of the **svccename** parameter + 1).

If the primary and standby databases are on different servers, they can use the same port number or service name; otherwise, they must have different values.

Host name, service or port name, and target list parameters (multiple standby mode)

In multiple standby mode, you should still set the **hadr_local_host**, **hadr_local_svc**, **hadr_remote_host**, **hadr_remote_host**, and **hadr_remote_inst** configuration parameters. If you set those parameters incorrectly, they are automatically updated after the primary connects to the standbys by using the settings of the **hadr_target_list** configuration parameter. This parameter specifies the host and port names of all the standbys. The first standby that you specify in the target list is considered to be the *principal HADR standby database*.

In multiple standby mode, you should still set the **hadr_local_host**, **hadr_local_svc**, **hadr_remote_host**, **hadr_remote_host**, and **hadr_remote_inst** configuration parameters. The **hadr_local_host** and **hadr_local_svc** parameters have the same meaning as in single standby mode. On the primary, set **hadr_remote_host**, **hadr_remote_host**, and **hadr_remote_inst** to indicate its principal standby. A new parameter, **hadr_target_list** is used to list all standbys, with the first entry being the principal standby. On standby, set the “remote” parameters to indicate the primary. In certain conditions, the “remote” parameters (on both the primary and the standby) can be automatically updated. For more information, see the “Automatic reconfiguration of HADR parameters” section in “Database configuration for multiple HADR standby databases” on page 441.

Synchronization mode

In single standby mode, the synchronization mode, which you specify with the **hadr_syncmode** configuration parameter must be identical on the primary and standby databases. The consistency of the value of this configuration parameter is checked when an HADR pair establishes a connection.

In multiple standby mode, the synchronization mode does not have to be the same. All standbys have an *effective synchronization mode* that is determined by the type of standby that they are. The principal standby uses the synchronization mode of the primary, and the auxiliary standbys use SUPERASYNC. All standbys have a *configured synchronization mode*, which is the explicit setting for **hadr_syncmode** and is used if a standby becomes the new primary.

For more detailed information, see “DB2 high availability disaster recovery (HADR) synchronization mode”.

HADR timeout and peer window

The timeout period, which you specify with the **hadr_timeout** configuration parameter, must be identical on the primary and standby databases. The consistency of the values of these configuration parameters is checked when an HADR pair establishes a connection.

With one exception, when the primary database starts, it waits for the longer of the two following periods for a standby to connect:

- For a minimum of 30 seconds
- For the number of seconds that is specified by the **hadr_timeout** database configuration parameter.

If the standby does not connect in the specified time, the startup fails. The one exception to this behavior is when you issue the **START HADR** command with the **BY FORCE** parameter. In this case, the primary database starts without waiting for the standby database to connect to it.

In multiple standby mode, the primary only waits for the principal standby to connect; a connection to an auxiliary standby is optional.

After an HADR pair establishes a connection, they exchange heartbeat messages. The heartbeat interval is computed from factors like the **hadr_timeout** and **hadr_peer_window** configuration parameters. It is reported by the **HEARTBEAT_INTERVAL** field in **MON_GET_HADR** table function. If one database does not receive any message from the other database within the number of seconds that is specified by the **hadr_timeout** configuration parameter, it initiates a disconnect. This behavior means that at most, it takes the number of seconds that is specified by the **hadr_timeout** configuration parameter for an HADR database to detect the failure of either its partner database or the intervening network. If you set the **hadr_timeout** configuration parameter too low, you will receive false alarms and frequent disconnections.

If you have the **hadr_peer_window** configuration parameter set to a nonzero value and the primary loses connection to the standby in peer state, the primary database does not commit transactions until the connection with the standby database is restored or the time value of the **hadr_peer_window** configuration parameter elapses, whichever happens first.

For maximal availability, the default value for the **hadr_peer_window** database configuration parameter is 0. When this parameter is set to 0, as soon as the connection between the primary and the standby is closed, the

primary drops out of peer state to avoid blocking transactions. The connection can close because the standby closed the connection, a network error is detected, or timeout is reached. For increased data consistency, but reduced availability, you can set the **hadr_peer_window** database configuration parameter to a nonzero value.

For more information, see “Setting the **hadr_timeout** and **hadr_peer_window** database configuration parameters”.

The following sample configuration is for the primary and standby databases:

Primary database:

HADR_LOCAL_HOST	host1.ibm.com
HADR_LOCAL_SVC	hadr_service
HADR_REMOTE_HOST	host2.ibm.com
HADR_REMOTE_SVC	hadr_service
HADR_REMOTE_INST	dbinst2
HADR_TIMEOUT	120
HADR_SYNCMODE	NEARSYNC
HADR_PEER_WINDOW	120

Standby database:

HADR_LOCAL_HOST	host2.ibm.com
HADR_LOCAL_SVC	hadr_service
HADR_REMOTE_HOST	host1.ibm.com
HADR_REMOTE_SVC	hadr_service
HADR_REMOTE_INST	dbinst1
HADR_TIMEOUT	120
HADR_SYNCMODE	NEARSYNC
HADR_PEER_WINDOW	120

Setting the **hadr_timeout** and **hadr_peer_window** database configuration parameters

You can configure the **hadr_timeout** and **hadr_peer_window** database configuration parameters for optimal response to a connection failure.

hadr_timeout database configuration parameter

If an HADR database does not receive any communication from its partner database for longer than the length of time specified by the **hadr_timeout** database configuration parameter, then the database concludes that the connection with the partner database is lost. If the database is in peer state when the connection is lost, then it moves into disconnected peer state if the **hadr_peer_window** database configuration parameter is greater than zero, or into remote catchup pending state if **hadr_peer_window** is not greater than zero. The state change applies to both primary and standby databases.

hadr_peer_window database configuration parameter

The **hadr_peer_window** configuration parameter does not replace the **hadr_timeout** configuration parameter. The **hadr_timeout** configuration parameter determines how long an HADR database waits before considering its connection with the partner database as failed. The **hadr_peer_window** configuration parameter determines whether the database goes into disconnected peer state after the connection is lost, and how long the database should remain in that state. HADR breaks the connection as soon as a network error is detected during send, receive, or poll on the TCP socket. HADR polls the socket every 100 milliseconds. This allows it to respond quickly to network errors detected by the OS. Only in the worst case does HADR wait until the timeout to break a bad

connection. In this case, a database application that is running at the time of failure can be blocked for a period of time equal to the sum of the **hadr_timeout** and **hadr_peer_window** database configuration parameters.

Setting the **hadr_timeout** and **hadr_peer_window** database configuration parameters

It is desirable to keep the waiting time that a database application experiences to a minimum. Setting the **hadr_timeout** and **hadr_peer_window** configuration parameters to small values would reduce the time that a database application must wait if a HADR standby databases loses its connection with the primary database. However, there are two other details that should be considered when choosing values to assign to the **hadr_timeout** and **hadr_peer_window** configuration parameters:

- The **hadr_timeout** database configuration parameter should be set to a value that is long enough to avoid false alarms on the HADR connection caused by short, temporary network interruptions. For example, the default value of **hadr_timeout** is 120 seconds, which is a reasonable value on many networks.
- The **hadr_peer_window** database configuration parameter should be set to a value that is long enough to allow the system to perform automated failure responses. If the HA system, for example a cluster manager, detects primary database failure before disconnected peer state ends, a failover to the standby database takes place. Data is not lost in the failover as all data from old primary is replicated to the new primary. If **hadr_peer_window** is too short, HA system may not have enough time to detect the failure and respond.

Note: In HADR multiple standby mode, the principal standby uses the primary's setting for **hadr_peer_window** (the *effective peer window*). The setting for **hadr_peer_window** on any auxiliary standby is meaningless because that type of standby always runs in SUPERASYNC mode.

Log archiving configuration for DB2 high availability disaster recovery (HADR)

To use log archiving with DB2 high availability disaster recovery (HADR), configure both the primary database and the standby database for automatic log retrieval capability from all log archive locations. For multiple standby systems, configure archiving on primary and all standby databases.

Only the current primary database can perform log archiving. If the primary and standby databases are set up with separate archiving locations, logs are archived only to the primary database's archiving location. In the event of a takeover, the standby database becomes the new primary database and any logs archived from that point on are saved to the original standby database's archiving location. In such a configuration, logs are archived to one location or the other, but not both; with the exception that following a takeover, the new primary database might archive a few logs that the original primary database had already archived. In a multiple standby system, the archived log files can be scattered among all databases' (primary and standbys) archive devices. A shared archive is preferred because all files are stored in a single location.

Many operations need to retrieve archived log files. These operations include: database roll forward, the HADR primary database retrieving log files to send to the standby database in remote catch up, and replication programs (such as Q Replication) reading logs. As a result, a shared archive for an HADR system is

preferred, otherwise, the needed files can be distributed on multiple archive devices, and user intervention is needed to locate the needed files and copy them to the requesting database. The recommended copy destination is an archive device. If copying into an archive is not feasible, copy the logs into the overflow log path. As a last resort, copy them into the log path (but you should be aware that there is a risk of damaging the active log files). DB2 does not auto delete user copied files in the overflow and active log path, so you should manually remove the files when they are no longer needed by any HADR standby or any application.

A specific scenario is a takeover in multiple standby mode. After the takeover, the new primary might not have all log files needed by other standbys (because a standby is at an older log position). If the primary cannot find a requested log file, it notifies the standby, which closes the connection and then reconnects in a few seconds to retry. The retry duration is limited to a few minutes. When retry time is exhausted, the standby shuts down. In this case, you should copy the files to the primary to ensure it has files from the first missing file to its current log file. After the files are copied, restart the standby if needed.

The standby database automatically manages log files in its log path. The standby database does not delete a log file from its local log path until it has been notified by the primary database that the primary database has archived it. This behavior provides added protection against the loss of log files. If the primary database fails and its log disk becomes corrupted before a particular log file is archived on the primary database, the standby database does not delete that log file from its own disk because it has not received notification that the primary database successfully archived the log file. If the standby database then takes over as the new primary database, it archives that log file before recycling it. If both the **logarchmeth1** and **logarchmeth2** configuration parameters are in use, the standby database does not recycle a log file until the primary database has archived it using both methods.

In addition to the benefits previously listed, a shared log archive device improves the catchup process by allowing the standby database to directly retrieve older log files from the archive in local catchup state, instead of retrieving those files indirectly through the primary in remote catchup state. However, it is recommended that you not use a serial archive device such as a tape drive for HADR databases. With serial devices, you might experience performance degradation on both the primary and standby databases because of mixed read and write operations. The primary writes to the device when it archives log files and the standby reads from the device to replay logs. This performance impact can occur even if the device is not configured as shared.

Shared log archives on Tivoli Storage Manager

Using a shared log archive with IBM Tivoli Storage Manager (TSM) allows one or more nodes to appear as a single node to the TSM server, which is especially useful in an HADR environment where either machine can be the primary at any one time.

To set up a shared log archive, you need to use proxy nodes which allow the TSM client nodes to perform data protection operations against a centralized name space on the TSM server. The target client node owns the data and agent nodes act on behalf of the target nodes to manage the backup data. The proxy node target is the node name defined on the TSM server to which backup versions of distributed data are associated. The data is managed in a single namespace on the TSM server as if it is entirely the data for this node. The proxy node target name can be a real

node (for example, one of the application hosts) or a virtual node name (that is, with no corresponding physical node). To create a virtual proxy node name, use the following commands on the TSM server:

```
Grant proxynode target=virtual-node-name agent=HADR-primary-name  
Grant proxynode target=virtual-node-name agent=HADR-standby-name
```

Next, you need to set these database configuration parameters on the primary and standby databases to the *virtual-node-name*:

- **vendoropt**
- **logarchopt**

In a multiple standby setup, you need to grade proxynode access to all machines on the TSM server and configure the **vendoropt** and **logarchopt** configuration parameters on all of the standbys.

HADR log spooling

The high availability disaster recovery (HADR) log spooling feature allows transactions on primary to make progress without having to wait for the log replay on the standby.

When this feature is enabled, log data sent by the primary is *spooled*, or written, to disk on the standby, and that log data is later read by log replay.

Log spooling, which is enabled by setting the **hadr_spool_limit** database configuration parameter, is an improvement to the HADR feature. When replay is slow, it is possible that new transactions on the primary can be blocked because it is not able to send log data to the standby system if there is no room in the buffer to receive the data. The log spooling feature means that the standby is not limited by the size of its buffer. When there is an increase in data received that cannot be contained in the buffer, the log replay reads the data from disk. This allows the system to better tolerate either a spike in transaction volume on the primary, or a slow down of log replay (due to the replay of particular type of log records) on the standby.

This feature could potentially lead to a larger gap between the log position of received logs on the standby and the log replay position on the standby, which can lead to longer takeover time. Use the **db2pd** command with the **-hadr** option or the **MON_GET_HADR** table function to monitor this gap by comparing the **STANDBY_LOG_POS** field, which shows receive position, and the **STANDBY_REPLAY_LOG_POS** field. You should consider your spool limit setting carefully because the old standby cannot start up as the new primary and receive transactions until the replay of the spooled logs has finished.

Index logging and high availability disaster recovery (HADR)

You should consider setting the database configuration parameters **logindexbuild** and **indexrec** for high availability disaster recovery (HADR) databases.

Using the **logindexbuild** database configuration parameter

Recommendation: For HADR databases, set the **logindexbuild** database configuration parameter to ON to ensure that complete information is logged for index creation, re-creation, and reorganization. Although this means that index builds might take longer on the primary system and that more log space is required, the indexes will be rebuilt on the standby system during HADR log replay and will be available when a failover takes place. Otherwise, when

replaying an index build or rebuild event, the standby marks the index invalid, because the log records do not contain enough information to populate the new index. If index builds on the primary system are not logged and a failover occurs, any invalid indexes that remain after the failover is complete have to be rebuilt before they can be accessed. While the indexes are being re-created, they cannot be accessed by any applications.

Note: If the LOG INDEX BUILD table attribute is set to its default value of NULL, DB2 uses the value specified for the **logindexbuild** database configuration parameter. If the LOG INDEX BUILD table attribute is set to ON or OFF, the value specified for the **logindexbuild** database configuration parameter is ignored.

You might choose to set the LOG INDEX BUILD table attribute to OFF on one or more tables for either of the following reasons:

- You do not have enough active log space to support logging of the index builds.
- The index data is very large and the table is not accessed often; therefore, it is acceptable for the indexes to be re-created at the end of the takeover operation. In this case, set the **indexrec** configuration parameter to RESTART. Because the table is not frequently accessed, this setting causes the system to re-create the indexes at the end of the takeover operation instead of waiting for the first time the table is accessed after the takeover operation.

If the LOG INDEX BUILD table attribute is set to OFF on one or more tables, any index build operation on those tables might cause the indexes to be re-created any time a takeover operation occurs. Similarly, if the LOG INDEX BUILD table attribute is set to its default value of NULL, and the **logindexbuild** database configuration parameter is set to OFF, any index build operation on a table might cause the indexes on that table to be re-created any time a takeover operation occurs. You can prevent the indexes from being re-created by taking one of the following actions:

- After all invalid indexes are re-created on the new primary database, take a backup of the database and apply it to the standby database. As a result of doing this, the standby database does not have to apply the logs used for re-creating invalid indexes on the primary database, which would mark those indexes as rebuild required on the standby database.
- Set the LOG INDEX BUILD table attribute to ON, or set the LOG INDEX BUILD table attribute to NULL and the **logindexbuild** configuration parameter to ON on the standby database to ensure that the index re-creation will be logged.

Using the **indexrec** database configuration parameter

Recommendation: Set the **indexrec** database configuration parameter to RESTART (the default) on both the primary and standby databases. This causes invalid indexes to be rebuilt after a takeover operation is complete. If any index builds have not been logged, this setting allows DB2 to check for invalid indexes and to rebuild them. This process takes place in the background, and the database is accessible after the takeover operation has completed successfully.

If a transaction accesses a table that has invalid indexes before the indexes have been rebuilt by the background re-create index process, the invalid indexes are rebuilt by the first transaction that accesses it.

High availability disaster recovery (HADR) performance

Configuring different aspects of your database system, including network bandwidth, CPU power, and buffer size, can improve the performance of your DB2 high availability disaster recovery (HADR) databases.

The network is the key part of your HADR setup because network connectivity is required to replicate database changes from the primary to the standby, keeping the two databases in sync.

Recommendations for maximizing network performance:

- Ensure that network bandwidth is greater than the database log generation rate.
- Consider network delays when you choose the HADR synchronization mode. Network delays affect the primary only in SYNC and NEARSYNC modes.

The slowdown in system performance as a result of using SYNC mode can be significantly larger than that of the other synchronization modes. In SYNC mode, the primary database sends log pages to the standby database only after the log pages are successfully written to the primary database log disk. To protect the integrity of the system, the primary database waits for an acknowledgment from the standby before it notifies an application that a transaction was prepared or committed. The standby database sends the acknowledgment only after it writes the received log pages to the standby database disk. The performance overhead equals the time that is needed for writing the log pages on the standby database plus the time that is needed for sending the messages back to the primary.

In NEARSYNC mode, the primary database writes and sends log pages in parallel. The primary then waits for an acknowledgment from the standby. The standby database acknowledges as soon as the log pages are received into its memory. On a fast network, the overhead to the primary database is minimal. The acknowledgment might have already arrived by the time the primary database finishes local log write.

For ASYNC mode, the log write and send are also in parallel; however, in this mode the primary database does not wait for an acknowledgment from the standby. Therefore, network delay is not an issue. Performance overhead is even smaller with ASYNC mode than with NEARSYNC mode.

For SUPERASYNC mode, transactions are never blocked or experience elongated response times because of network interruptions or congestion. New transactions can be processed as soon as previously submitted transactions are written to the primary database. Therefore, network delay is not an issue. The elapsed time for the completion of non-forced takeover operations might be longer than in other modes because the log gap between the primary and the standby databases might be relatively larger.

- Consider tuning the **DB2_HADR_SOSNDBUF** and **DB2_HADR_SORCVBUF** registry variables.

HADR log shipping workload, network bandwidth, and transmission delay are important factors to consider when you are tuning the TCP socket buffer sizes. Two registry variables, **DB2_HADR_SOSNDBUF** and **DB2_HADR_SORCVBUF** allow tuning of the TCP socket send and receive buffer size for HADR connections only. These two variables have the value range of 1024 to 4294967295 and default to the socket buffer size of the operating system, which varies depending on the operating system. It is strongly recommended that you use a minimum value of 16384 (16 K) for your

DB2_HADR_SOSNDBUF and **DB2_HADR_SORCVBUF** settings. Some operating systems automatically round or silently cap the user specified value.

You can use the HADR simulator (a command-line tool that generates a simulated HADR workload) to measure network performance and to experiment with various network tuning options. You can download the simulator at <https://www.ibm.com/developerworks/community/wikis/home/wiki/DB2HADR/page/HADR%20simulator>.

Network congestion

For each log write on the primary, the same log pages are also sent to the standby. Each write operation is called a *flush*. The size of the flush is limited to the log buffer size on the primary database (which is controlled by the database configuration parameter **logbufsz**). The exact size of each flush is nondeterministic. A larger log buffer does not necessarily lead to a larger flush size.

If the standby database is too slow replaying log pages, its log-receiving buffer might fill up, thereby preventing the buffer from receiving more log pages. In SYNC and NEARSYNC modes, if the primary database flushes its log buffer one more time, the data is likely to be buffered in the network pipeline consisting of the primary machine, the network, and the standby database. Because the standby database does not have free buffer to receive the data, it cannot acknowledge, so the primary database becomes blocked while it is waiting for the standby database's acknowledgement.

In ASYNC mode, the primary database continues to send log pages until the pipeline fills up and it cannot send additional log pages. This condition is called *congestion*. Congestion is reported by the **hadr_connect_status** monitor element. For SYNC and NEARSYNC modes, the pipeline can usually absorb a single flush and congestion does not occur. However, the primary database remains blocked waiting for an acknowledgment from the standby database on the flush operation.

Congestion can also occur if the standby database is replaying log records that take a long time to replay, such as database or table reorganization log records.

In SUPERASYNC mode, since the transaction commit operations on the primary database are not affected by the relative slowness of the HADR network or the standby HADR server, the log gap between the primary database and the standby database might continue to increase. It is important to monitor the log gap as it is an indirect measure of the potential number of transactions that might be lost should a true disaster occur on the primary system. In disaster recovery scenarios, any transactions that are committed during the log gap would not be available to the standby database. Therefore, monitor the log gap by using the **hadr_log_gap** monitor element; if it occurs that the log gap is not acceptable, investigate the network interruptions or the relative speed of the standby HADR server and take corrective measures to reduce the log gap.

Recommendations for minimizing network congestion:

- The standby database should be powerful enough to replay the logged operations of the database as fast as they are generated on the primary. Identical primary and standby hardware is recommended.
- Consider tuning the size of the standby database log-receiving buffer by using the **DB2_HADR_BUF_SIZE** registry variable.

A larger buffer can help to reduce congestion, although it might not remove all of the causes of congestion. By default, the size of the standby database

log-receiving buffer is two times the size of the primary database log-writing buffer. The database configuration parameter **logbufsz** specifies the size of the primary database log-writing buffer.

You can determine if the standbys log-receiving buffer is inadequate by using the **db2pd** command with the -hadr option or the MON_GET_HADR table function. If the value for the STANDBY_RECV_BUF_PERCENT field, which indicates the percentage of standby log receiving buffer that is being used, is close to 100, increase the **DB2_HADR_BUF_SIZE** setting.

- Consider setting the **DB2_HADR_PEER_WAIT_LIMIT** registry variable, which allows you to prevent primary database logging from blocking because of a slow or blocked standby database.

When the **DB2_HADR_PEER_WAIT_LIMIT** registry variable is set, the HADR primary database breaks out of the peer state if logging on the primary database is blocked for the specified number of seconds because of log replication to the standby. When this limit is reached, the primary database breaks the connection to the standby database. If the peer window is disabled, the primary enters disconnected state and logging resumes. If the peer window is enabled, the primary database enters disconnected peer state, in which logging continues to be blocked. The primary database leaves disconnected peer state upon re-connection or peer window expiration.

Logging resumes after the primary database leaves disconnected peer state.

Note: If you set **DB2_HADR_PEER_WAIT_LIMIT**, use a minimum value of 10 to avoid triggering false alarms.

Honoring peer window transition when breaking out of peer state ensures peer window semantics for safe takeover in all cases. If the primary fails during the transition, normal peer window protection still applies (safe takeover from standby as long as it is still in disconnected-peer state).

- In most systems, the logging capability is not driven to its limit. Even in SYNC mode, there might not be an observable slow down on the primary database. For example, if the limit of logging is 40 MB per second with HADR enabled, but the system was just running at 30 MB per second before HADR is enabled, then you might not notice any difference in overall system performance.
- To speed up the catchup process, you can use a shared log archive device. However, if the shared device is a serial device such as a tape drive, you might experience performance degradation on both the primary and standby databases because of mixed read and write operations.
- If you are going to use the reads on standby feature, the standby must have the resources to accommodate this additional work.
- If you are going to use the reads on standby feature, configure your buffer pools on the primary, and that information is shipped to the standby through logs.
- If you are going to use the reads on standby feature, Tune the **pckcachesz**, **catalogcache_sz**, **applheapsz**, and **sortheap** configuration parameters on the standby.

Cluster managers and high availability disaster recovery (HADR)

You can implement DB2 High Availability Disaster Recovery (HADR) databases on nodes of a cluster, and use a cluster manager to improve the availability of your database solution.

You can have both the primary database and the standby database managed by the same cluster manager, or you can have the primary database and the standby database managed by different cluster managers.

Set up an HADR pair where the primary and standby databases are serviced by the same cluster manager

This configuration is best suited to environments where the primary and standby databases are located at the same site and where the fastest possible failover is required. These environments would benefit from using HADR to maintain DBMS availability, rather using crash recovery or another recovery method.

You can use the cluster manager to quickly detect a problem and to initiate a takeover operation. Because HADR requires separate storage for the DBMS, the cluster manager should be configured with separate volume control. This configuration prevents the cluster manager from waiting for failover to occur on the volume before using the DBMS on the standby system. You can use the automatic client reroute feature to redirect client applications to the new primary database.

Set up an HADR pair where the primary and standby databases are not serviced by the same cluster manager

This configuration is best suited to environments where the primary and standby databases are located at different sites and where high availability is required for disaster recovery in the event of a complete site failure. There are several ways you can implement this configuration. When an HADR primary or standby database is part of a cluster, there are two possible failover scenarios.

- If a partial site failure occurs and a node to which the DBMS can fail over remains available, you can choose to perform a cluster failover. In this case, the IP address and volume failover is performed using the cluster manager; HADR is not affected.
- If a complete site failure occurs where the primary database is located, you can use HADR to maintain DBMS availability by initiating a takeover operation. If a complete site failure occurs where the standby database is located, you can repair the site or move the standby database to another site.

Performing an HADR failover operation

When you want the current standby database to become the new primary database because the current primary database is not available, you can perform a failover.

About this task

Warning:

This procedure might cause a loss of data. Review the following information before performing this emergency procedure:

- Ensure that the primary database is no longer processing database transactions. If the primary database is still running, but cannot communicate with the standby database, executing a forced takeover operation (issuing the **TAKEOVER HADR** command with the **BY FORCE** option) could result in two primary databases. When there are two primary databases, each database will have different data, and the two databases can no longer be automatically synchronized.

- Deactivate the primary database or stop its instance, if possible. (This might not be possible if the primary system has hung, crashed, or is otherwise inaccessible.) After a takeover operation is performed, if the failed database is later restarted, it will not automatically assume the role of primary database.
- The likelihood and extent of transaction loss depends on your specific configuration and circumstances:
 - If the primary database fails while in peer state or disconnected peer state and the synchronization mode is synchronous (SYNC), the standby database will not lose transactions that were reported committed to an application before the primary database failed.
 - If the primary database fails while in peer state or disconnected peer state and the synchronization mode is near synchronous (NEARSYNC), the standby database can only lose transactions committed by the primary database if both the primary and the standby databases fail at the same time.
 - If the primary database fails while in peer state or disconnected peer state and the synchronization mode is asynchronous (ASYNC), the standby database can lose transactions committed by the primary database if the standby database did not receive all of the log records for the transactions before the takeover operation was performed. The standby database can also lose transactions committed by the primary database if the standby database crashes before it was able to write all the received logs to disk.

Note: Peer window is not allowed in ASYNC mode, therefore the primary database will never enter disconnected peer state in that mode.

- If the primary database fails while in remote catchup state and the synchronization mode is super asynchronous (SUPERASYNC), the standby database can lose transactions committed by the primary database if the standby database did not receive all of the log records for the transactions before the takeover operation was performed. The standby database can also lose transactions committed by the primary database if the standby database crashes before it was able to write all the received logs to disk.

Note: Databases can never be in peer or disconnected peer state in SUPERASYNC mode. Failover (forced takeover) is allowed in remote catchup state only if the synchronization mode is SUPERASYNC.

- If the primary database fails while in remote catchup pending state, transactions that have not been received and processed by the standby database will be lost.

Note: Any log gap shown in the database snapshot will represent the gap at the last time the primary and standby databases were communicating with each other; the primary database might have processed a very large number of transactions since that time.

- Ensure that any application that connects to the new primary (or that is rerouted to the new primary by client reroute), is prepared to handle the following:
 - There is data loss during failover. The new primary does not have all of the transactions committed on the old primary. This can happen even when the **hadr_syncmode** configuration parameter is set to SYNC. Because an HADR standby applies logs sequentially, you can assume that if a transaction in an SQL session is committed on the new primary, all previous transactions in the same session have also been committed on the new primary. The commit sequence of transactions across multiple sessions can be determined only with detailed analysis of the log stream.

- It is possible that a transaction can be issued to the original primary, committed on the original primary and replicated to the new primary (original standby), but not be reported as committed because the original primary crashed before it could report to the client that the transaction was committed. Any application you write should be able to handle that transactions issued to the original primary, but not reported as committed on the original primary, are committed on the new primary (original standby).
- Some operations are not replicated, such as changes to database configuration and to external UDF objects.
- The **TAKEOVER HADR** command can only be issued on the standby database.
- HADR does not interface with the DB2 fault monitor (db2fm) which can be used to automatically restart a failed database. If the fault monitor is enabled, you should be aware of possible fault monitor action on a presumably failed primary database.
- A takeover operation can only take place if the primary and standby databases are in peer state or the standby database is in remote catchup pending state. If the standby database is in any other state, an error will be returned.

Note: You can make a standby database that is in local catchup state available for normal use by converting it to a standard database. To do this, shut the database down by issuing the **DEACTIVATE DATABASE** command, and then issue the **STOP HADR** command. Once HADR has been stopped, you must complete a rollforward operation on the former standby database before it can be used. A database cannot rejoin an HADR pair after it has been converted from a standby database to a standard database. To restart HADR on the two servers, follow the procedure for initializing HADR.

If you have configured a peer window, shut down the primary before the window expires to avoid potential transaction loss in any related failover.

In a failover scenario, a takeover operation can be performed through the command line processor (CLP), or the db2HADRTakeover application programming interface (API).

Procedure

The following procedure shows you how to initiate a failover on the primary or standby database using the CLP:

1. Completely disable the failed primary database. When a database encounters internal errors, normal shutdown commands might not completely shut it down. You might need to use operating system commands to remove resources such as processes, shared memory, or network connections.
2. Issue the **TAKEOVER HADR** command with the **BY FORCE** option on the standby database. In the following example the failover takes place on database LEAFS:
TAKEOVER HADR ON DB LEAFS BY FORCE

The **BY FORCE** option is required because the primary is expected to be offline. If the primary database is not completely disabled, the standby database will still have a connection to the primary and will send a message to the primary database asking it to shutdown. The standby database will still switch to the role of primary database whether or not it receives confirmation from that the primary database has been shutdown.

Switching database roles in high availability disaster recovery (HADR)

During high availability disaster recovery (HADR), use the **TAKEOVER HADR** command to switch the roles of the primary and standby databases.

About this task

- The **TAKEOVER HADR** command can only be issued on the standby database. If the primary database is not connected to the standby database when the command is issued, the takeover operation will fail.
- The **TAKEOVER HADR** command can only be used to switch the roles of the primary and standby databases if the databases are in peer state. If the standby database is in any other state, an error message will be returned.

Procedure

To switch the HADR database roles:

- Use the CLP to initiate a takeover operation on the standby database, issue the **TAKEOVER HADR** command without the **BY FORCE** option on the standby database.

In the following example, the takeover operation takes place on the standby database LEAFS:

```
TAKEOVER HADR ON DB LEAFS
```

A log full error is slightly more likely to occur immediately following a takeover operation. To limit the possibility of such an error, an asynchronous buffer pool flush is automatically started at the end of each takeover. The likelihood of a log full error decreases as the asynchronous buffer pool flush progresses.

Additionally, if your configuration provides a sufficient amount of active log space, a log full error is even more unlikely. If a log full error does occur, the current transaction is aborted and rolled back.

Note: Issuing the **TAKEOVER HADR** command without the **BY FORCE** option will cause any applications currently connected to the HADR primary database to be forced off. This action is designed to work in coordination with automatic client reroute to assist in rerouting clients to the new HADR primary database after a role switch. However, if the forcing off of applications from the primary would be disruptive in your environment, you might want to implement your own procedure to shut down such applications prior to performing a role switch, and then restart them with the new HADR primary database as their target after the role switch is completed.

- Call the db2HADRTakeover application programming interface (API) from an application.
- Open the task assistant for the **TAKEOVER HADR** command in IBM Data Studio.

Reintegrating a database after a takeover operation

If you executed a takeover operation in a high availability disaster recovery (HADR) environment because the primary database failed, you can bring the failed database back online and use it as a standby database or return it to its status as primary database.

Procedure

To reintegrate the failed primary database into the HADR pair as the new standby database:

1. Repair the system where the original primary database resided. This could involve repairing failed hardware or rebooting the crashed operating system.
2. Restart the failed primary database as a standby database. In the following example, database LEAFS is started as a standby database:

```
START HADR ON DB LEAFS AS STANDBY
```

Note: Reintegration will fail if the two copies of the database have incompatible log streams. In particular, HADR requires that the original primary database did not apply any logged operation that was never reflected on the original standby database before it took over as the new primary database. If this did occur, you can restart the original primary database as a standby database by restoring a backup image of the new primary database or by initializing a split mirror.

Successful return of this command does not indicate that reintegration has succeeded; it means only that the database has been started. Reintegration is still in progress. If reintegration subsequently fails, the database will shut itself down. You should monitor standby states using the **GET SNAPSHOT FOR DATABASE** command or the **db2pd** tool to make sure that the standby database stays online and proceeds with the normal state transition. If necessary, you can check the administration notification log file and the **db2diag** log file to find out the status of the database.

What to do next

After the original primary database has rejoined the HADR pair as the standby database, you can choose to perform a failback operation to switch the roles of the databases to enable the original primary database to be once again the primary database. To perform this failback operation, issue the following command on the standby database:

```
TAKEOVER HADR ON DB LEAFS
```

Note:

1. If the HADR databases are not in peer state or the pair is not connected, this command will fail.
2. Open sessions on the primary database are forced closed and inflight transactions are rolled back.
3. When switching the roles of the primary and standby databases, the **BY FORCE** option of the **TAKEOVER HADR** command cannot be specified.

Monitoring high availability disaster recovery (HADR) environments

Monitoring is an integral part of setting up and maintaining your HADR setup. The DB2 monitoring interfaces provide a detailed picture of the configuration and health of your environment.

You can use a number of methods to monitor the status of your HADR databases. There are two preferred ways of monitoring HADR:

- The db2pd command
- The MON_GET_HADR table function

You can also use the following methods, but starting in Version 10.1, they are deprecated, and they might be removed in a future release:

- The **GET SNAPSHOT FOR DATABASE** command

- The db2GetSnapshot API
- The SNAPHADR administrative view
- The SNAP_GET_HADR table function
- Other snapshot administrative views and table functions

db2pd command

This command retrieves information from the DB2 memory sets. You can issue this command from either a primary database or a standby database. If you are using multiple standby mode and you issue this command from a standby, it does not return any information about the other standbys. If you issue this command from the primary, it returns information on all standbys

To view information about high availability disaster recovery for database HADRDB, you could issue the following command:

```
db2pd -db HADRDB -hadr
```

Assuming you issued that command from the primary, you would receive something like the following sample output:

```
Database Member 0 -- Database HADRDB -- Active -- Up 0 days 00:23:17 --
Date 06/08/2011 13:57:23
```

```

          HADR_ROLE = PRIMARY
          REPLAY_TYPE = PHYSICAL
          HADR_SYNCMODE = SYNC
          STANDBY_ID = 1
          LOG_STREAM_ID = 0
          HADR_STATE = PEER
PRIMARY_MEMBER_HOST = hostP.ibm.com
PRIMARY_INSTANCE = db2inst
PRIMARY_MEMBER = 0
STANDBY_MEMBER_HOST = hostS1.ibm.com
STANDBY_INSTANCE = db2inst
STANDBY_MEMBER = 0
HADR_CONNECT_STATUS = CONNECTED
HADR_CONNECT_STATUS_TIME = 06/08/2011 13:38:10.199479 (1307565490)
HEARTBEAT_INTERVAL(seconds) = 25
          HADR_TIMEOUT(seconds) = 100
TIME_SINCE_LAST_RECV(seconds) = 3
          PEER_WAIT_LIMIT(seconds) = 0
LOG_HADR_WAIT_RECENT_AVG(seconds) = 0.006298
LOG_HADR_WAIT_ACCUMULATED(seconds) = 0.516
          LOG_HADR_WAIT_COUNT = 82
SOCK_SEND_BUF_REQUESTED,ACTUAL(bytes) = 0, 50772
SOCK_RECV_BUF_REQUESTED,ACTUAL(bytes) = 0, 87616
          PRIMARY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
          STANDBY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
          HADR_LOG_GAP(bytes) = 0
STANDBY_REPLY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
STANDBY_RECV_REPLY_GAP(bytes) = 0
          PRIMARY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
          STANDBY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
          STANDBY_REPLY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
STANDBY_RECV_BUF_SIZE(pages) = 16
          STANDBY_RECV_BUF_PERCENT = 0
          STANDBY_SPOOL_LIMIT(pages) = 0
          PEER_WINDOW(seconds) = 0
READS_ON_STANDBY_ENABLED = Y
STANDBY_REPLY_ONLY_WINDOW_ACTIVE = N
```

MON_GET_HADR table function

If you issue this query on the primary, it will return information on all standbys. If you want to issue the MON_GET_HADR function against a standby database, be aware of the following points:

- You must enable reads on standby on the standby.

- Even if your HADR setup is in multiple standby mode, the table function does not return any information about any other standbys.

For example, you could issue the following query on the primary database:

```
db2 "select HADR_ROLE, STANDBY_ID, HADR_STATE,
      varchar(PRIMARY_MEMBER_HOST,20) as PRIMARY_MEMBER_HOST,
      varchar(STANDBY_MEMBER_HOST,20) as STANDBY_MEMBER_HOST
   from table (mon_get_hadr(NULL))"
```

Sample output is as follows:

HADR_ROLE	STANDBY_ID	HADR_STATE	PRIMARY_MEMBER_HOST	STANDBY_MEMBER_HOST
PRIMARY		1 PEER	hostP.ibm.com	hostS1.ibm.com

1 record(s) selected.

GET SNAPSHOT FOR DATABASE command

This command collects status information and formats the output. The information that is returned is a snapshot of the database manager operational status at the time that you issued the command. HADR information is displayed in the output under the heading HADR status.

db2GetSnapshot API

This API collects database manager monitor information and writes it to a user-allocated data buffer. The information that is returned is a snapshot of the database manager operational status at the time that the API was called.

SNAPHADR administrative view and SNAP_GET_HADR table function

This administrative view and this table function return information about HADR from a database snapshot, in particular, the HADR logical data group.

Other snapshot administrative views and table functions

You can use the following snapshot administrative views and table functions, which are not HADR specific and return a wider set of information, to query a subsection of the HADR information:

- ADMIN_GET_STORAGE_PATHS
- MON_GET_TRANSACTION_LOG
- SNAPDB
- SNAPDB_MEMORY_POOL
- SNAPDETAILLOG
- SNAP_GET_DB
- SNAP_GET_DB_MEMORY_POOL

Stopping DB2 High Availability Disaster Recovery (HADR)

If you are using the DB2 High Availability Disaster Recovery (HADR) feature, stopping HADR operations to perform maintenance on the primary or standby databases might be necessary. Stop HADR operations only on the database that you are performing maintenance. To stop using HADR completely, stop HADR on both databases.

About this task

Warning: If you want to stop the specified database but you still want it to maintain its role as either an HADR primary or standby database, do not issue the STOP HADR command. If you issue the **STOP HADR** command the database will become a standard database and might require reinitialization in order to resume operations as an HADR database. Instead, issue the **DEACTIVATE DATABASE** command.

If you issue the **STOP HADR** command against a standard database, an error will be returned.

Procedure

To stop HADR operations on the primary or standby database:

- From the CLP, issue the **STOP HADR** command on the database where you want to stop HADR operations.

In the following example, HADR operations are stopped on database SOCKS:

```
STOP HADR ON DATABASE SOCKS
```

If you issue this command against an inactive primary database, the database switches to a standard database and remains offline.

If you issue this command against an inactive standby database the database switches to a standard database, is placed in rollforward pending state, and remains offline.

If you issue this command on an active primary database, logs stop being shipped to the standby database and all HADR engine dispatchable units (EDUs) are shut down on the primary database. The database switches to a standard database and remains online. Transaction processing can continue. You can issue the START HADR command with the AS PRIMARY option to switch the role of the database back to primary database.

If you issue this command on an active standby database, an error message is returned, indicating that you must deactivate the standby database before attempting to convert it to a standard database.

- From an application, call the **db2HADRStop** application programming interface (API).
- From IBM Data Studio, open the task assistant for the **STOP HADR** command.

Chapter 18. Problem-determination tools

Use the problem-determination tools and resources that are provided with your DB2 products to help you understand, isolate, and resolve problems.

DB2 diagnostic (db2diag) log files

The DB2 diagnostic **db2diag** log files are primarily intended for use by IBM Software Support for troubleshooting purposes. The administration notification log is primarily intended for troubleshooting use by database and system administrators. Administration notification log messages are also logged to the **db2diag** log files using a standardized message format.

Overview

With DB2 diagnostic and administration notification messages both logged within the **db2diag** log files, this often makes the **db2diag** log files the first location to examine in order to obtain information about the operation of your databases. Help with the interpretation of the contents of these diagnostic log files is provided in the topics listed in the "Related links" section. If your troubleshooting attempts are unable to resolve your problem and you feel you require assistance, you can contact IBM Software Support (for details, see the "Contacting IBM Software Support" topic). In gathering relevant diagnostic information that will be requested to be sent to IBM Software Support, you can expect to include your **db2diag** log files among other sources of information which includes other relevant logs, storage dumps, and traces.

The **db2diag** log file can exist in two different forms:

Single diagnostic log file

One active diagnostic log file, named db2diag.log, that grows in size indefinitely. This is the default form and it exists whenever the **diagsize** database manager configuration parameter has the value of 0 (the default value for this parameter is 0).

Rotating diagnostic log files

A single active log file (named db2diag.N.log, where N is the file name index that is a continuously growing number starting from 0), although a series of diagnostic log files can be found in the location defined by the **diagpath** configuration parameter, each growing until reaching a limited size, at which time the log file is closed and a new one is created and opened for logging with an incremented file name index (db2diag.N+1.log). It exists whenever the **diagsize** database manager configuration parameter has a nonzero value.

You can choose which of these two forms exist on your system by appropriately setting the **diagsize** database manager configuration parameter.

Configuration

The **db2diag** log files can be configured in size, location, and the types of diagnostic errors recorded by setting the following database manager configuration parameters:

diagsize

The value of **diagsize** decides what form of diagnostic log file will be adopted. If the value is 0, a single diagnostic log file will be adopted. If the value is not 0, rotating diagnostic log files will be adopted, and this nonzero value also specifies the total size of all rotating diagnostic log files and all rotating administration notification log files. The instance must be restarted for the new value of the **diagsize** parameter to take effect. See the "diagsize - Diagnostic log file size configuration parameter" topic for complete details.

diagpath

Diagnostic information can be specified to be written to **db2diag** log files in the location defined by the **diagpath** configuration parameter. See the "diagpath - Diagnostic data directory path configuration parameter" topic for complete details.

alt_diagpath

The **alt_diagpath** database manager configuration parameter provides an alternate diagnostic data directory path for storing diagnostic information. If the database manager fails to write to the path specified by **diagpath**, the path specified by **alt_diagpath** is used to store diagnostic information.

diaglevel

The types of diagnostic errors written to the **db2diag** log files can be specified with the **diaglevel** configuration parameter. See the "diaglevel - Diagnostic error capture level configuration parameter" topic for complete details.

Note: If the **diagsize** configuration parameter is set to a non-zero value, that value specifies the total size of the combination of all rotating administration notification log files and all rotating diagnostic log files contained within the diagnostic data directory. For example, if a system with 4 database partitions has **diagsize** set to 1 GB, the maximum total size of the combined notification and diagnostic logs can reach is 4 GB (4 x 1 GB).

Interpretation of diagnostic log file entries

Use the **db2diag** log files analysis tool (**db2diag**) to filter and format the **db2diag** log files. With the addition of administration notification log messages being logged to the **db2diag** log files using a standardized message format, viewing the **db2diag** log files first is a recommended choice to understand what has been happening to the database.

As an alternative to using **db2diag**, you can use a text editor to view the diagnostic log file on the machine where you suspect a problem to have occurred. The most recent events recorded are the furthest down the file.

Note: The administration notification (*instance_name.nfy*) and diagnostic (*db2diag*) logs grow *continuously* as single log files. When the **diagsize** database manager configuration parameter is set to a nonzero value, both the administration notification and the **db2diag** log files become a series of rotating log files (*instance_name.N.nfy* and *db2diag.N.log*) having a limited total size which is determined by the value of the **diagsize** configuration parameter.

The following example shows the header information for a sample log entry, with all the parts of the log identified.

Note: Not every log entry will contain all of these parts. Only the first several fields (timestamp to TID) and FUNCTION will be present in all the **db2diag** log file records.

```
2007-05-18-14.20.46.973000-240 [1] I27204F655 [2] LEVEL: Info [3]
PID : 3228 [4] TID : 8796 [5] PROC : db2syscs.exe [6]
INSTANCE: DB2MPP [7] NODE : 002 [8] DB : WIN3DB1 [9]
APPHDL : 0-51 [10] APPID: 9.26.54.62.45837.070518182042 [11]
AUTHID : UDBADM [12]
EDUID : 8796 [13] EDUNAME: db2agntp [14] (WIN3DB1) 2
FUNCTION: [15] DB2 UDB, data management, sqldInitDBCB, probe:4820
DATA #1 : [16] String, 26 bytes
Setting ADC Threshold to:
DATA #2 : unsigned integer, 8 bytes
1048576
```

Legend:

1. A timestamp and timezone for the message.

Note: Timestamps in the **db2diag** log files contain a time zone. For example: 2006-02-13-14.34.35.965000-300, where "-300" is the difference between UTC (Coordinated Universal Time, formerly known as GMT) and local time at the application server in minutes. Thus -300 represents UTC - 5 hours, for example, EST (Eastern Standard Time).

2. The record ID field. The recordID of the **db2diag** log files specifies the file offset at which the current message is being logged (for example, "27204") and the message length (for example, "655") for the platform where the DB2 diagnostic log was created.
3. The diagnostic level of the message. The levels are Info, Warning, Error, Severe, Critical, and Event.
4. The process ID
5. The thread ID
6. The process name
7. The name of the instance generating the message.
8. For multi-partition systems, the database partition generating the message. (In a non-partitioned database, the value is "000".)
9. The database name
10. The application handle. This value aligns with that used in **db2pd** output and lock dump files. It consists of the coordinator partition number followed by the coordinator index number, separated by a dash.
11. Identification of the application for which the process is working. In this example, the process generating the message is working on behalf of an application with the ID 9.26.54.62.45837.070518182042.

A TCP/IP-generated application ID is composed of three sections

1. **IP address:** It is represented as a 32-bit number displayed as a maximum of 8 hexadecimal characters.
2. **Port number:** It is represented as 4 hexadecimal characters.
3. **A unique identifier** for the instance of this application.

Note: When the hexadecimal versions of the IP address or port number begin with 0 through to 9, they are changed to G through to P. For example, "0" is mapped to "G", "1" is mapped to "H", and so on. The IP

address, AC10150C.NA04.006D07064947 is interpreted as follows: The IP address remains AC10150C, which translates to 172.16.21.12. The port number is NA04. The first character is "N", which maps to "7". Therefore, the hexadecimal form of the port number is 7A04, which translates to 31236 in decimal form.

This value is the same as the *appl_id* monitor element data. For detailed information about how to interpret this value, see the documentation for the *appl_id* monitor element.

To identify more about a particular application ID, either:

- Use the **LIST APPLICATIONS** command on a DB2 server or **LIST DCS APPLICATIONS** on a DB2 Connect™ gateway to view a list of application IDs. From this list, you can determine information about the client experiencing the error, such as its database partition name and its TCP/IP address.
- Use the **GET SNAPSHOT FOR APPLICATION** command to view a list of application IDs.
- Use the **db2pd -applications -db <dbname>** command.

- 12 The authorization identifier.
13 The engine dispatchable unit identifier.
14 The name of the engine dispatchable unit.
15. The product name ("DB2"), component name ("data management"), and function name ("sqlInitDBCB") that is writing the message (as well as the probe point ("4820") within the function).
16. The information returned by a called function. There may be multiple data fields returned.

Now that you have seen a sample **db2diag** log file entry, here is a list of all of the possible fields:

```
<timestamp><timezone>      <recordID>          LEVEL: <level> (<source>)
PID    : <pid>              TID   : <tid>           PROC  : <procName>
INSTANCE: <instance>        NODE  : <node>          DB    : <database>
APPHDL : <appHandle>       APPID: <appID>
AUTHID : <authID>
EDUID  : <eduID>           EDUNAME: <engine dispatchable unit name>
FUNCTION: <prodName>, <compName>, <funcName>, probe:<probeNum>
MESSAGE : <messageID> <msgText>
CALLED  : <prodName>, <compName>, <funcName>  OSERR: <errorName> (<errno>)
RETCODE : <type>=<retCode> <errorDesc>
ARG #N : <typeTitle>, <typeName>, <size> bytes
... argument ...
DATA #N : <typeTitle>, <typeName>, <size> bytes
... data ...
```

The fields which were not already explained in the example, are:

- - <source> Indicates the origin of the logged error. (You can find it at the end of the first line in the sample.) The possible values are:
 - origin - message is logged by the function where error originated (inception point)
 - OS - error has been produced by the operating system
 - received - error has been received from another process (client/server)
 - sent - error has been sent to another process (client/server)

- MESSAGE Contains the message being logged. It consists of:
 - <messageID> - message number, for example, ECF=0x9000004A or DIA8604C
 - <msgText> - error description

When the CALLED field is also present, <msgText> is an impact of the error returned by the CALLED function on the function logging a message (as specified in the FUNCTION field)
- CALLED This is the function that returned an error. It consists of:
 - <prodName> - The product name: "OS", "DB2", "DB2 Tools" or "DB2 Common"
 - <compName> - The component name ('-' in case of a system call)
 - <funcName> - The called function name
- OSERR This is the operating system error returned by the CALLED system call. (You can find it at the end of the same line as CALLED.) It consists of:
 - <errorName> - the system specific error name
 - <errno> - the operating system error number
- ARG This section lists the arguments of a function call that returned an error. It consists of:
 - <N> - The position of an argument in a call to the "called" function
 - <typeTitle> - The label associated with the Nth argument typename
 - <typeName> - The name of the type of argument being logged
 - <size> - The size of argument to be logged
- DATA This contains any extra data dumped by the logging function. It consists of:
 - <N> - The sequential number of data object being dumped
 - <typeTitle> - The label of data being dumped
 - <typeName> - The name of the type of data field being logged, for example, PD_TYPE_UINT32, PD_TYPE_STRING
 - <size> - The size of a data object

Interpreting the informational record of the db2diag log files

The first message in the **db2diag** log files should always be an informational record.

An example of an informational record is as follows:

```
2006-02-09-18.07.31.059000-300 I1H917          LEVEL: Event
PID     : 3140           TID : 2864           PROC : db2start.exe
INSTANCE: DB2             NODE : 000
FUNCTION: DB2 UDB, RAS/PD component, _pdlogInt, probe:120
START   : New Diagnostic Log file
DATA #1 : Build Level, 124 bytes
Instance "DB2" uses "32" bits and DB2 code release "SQL09010"
with level identifier "01010107".
Informational tokens are "DB2 v9.1.0.190", "s060121", "", Fix Pack "0".
DATA #2 : System Info, 1564 bytes
System: WIN32_NT MYSRVR Service Pack 2 5.1 x86 Family 15, model 2, stepping 4
CPU: total:1 online:1 Cores per socket:1 Threading degree per core:1
Physical Memory(MB): total:1024 free:617 available:617
Virtual Memory(MB): total:2462 free:2830
Swap    Memory(MB): total:1438 free:2213
Information in this record is only valid at the time when this file was created
(see this record's time stamp)
```

The Informational record is output for **db2start** on every logical partition. This results in multiple informational records: one per logical partition. Since the informational record contains memory values which are different on every partition, this information might be useful.

Setting the error capture level of the diagnostic log files

The DB2 diagnostic (**db2diag**) log files are files that contain text information logged by DB2 database systems. This information is used for troubleshooting and much of it is primarily intended for IBM Software Support.

About this task

The types of diagnostic errors that are recorded in the **db2diag** log files are determined by the **diaglevel** database manager configuration parameter setting.

Procedure

- To check the current setting, issue the command **GET DBM CFG**.

Look for the following variable:

Diagnostic error capture level (DIAGLEVEL) = 3

- To change the value dynamically, use the **UPDATE DBM CFG** command.

To change a database manager configuration parameter online:

```
db2 attach to instance-name  
db2 update dbm cfg using parameter-name value  
db2 detach
```

For example:

DB2 UPDATE DBM CFG USING DIAGLEVEL X

where X is the notification level you want. If you are diagnosing a problem that can be reproduced, IBM Software Support personnel might suggest that you use **diaglevel 4** while performing troubleshooting.

First occurrence data capture information

First occurrence data capture (FODC) collects diagnostic information about a DB2 instance, host or member when a problem occurs. FODC reduces the need to reproduce a problem to obtain diagnostic information, because diagnostic information can be collected as the problem occurs.

FODC can be invoked manually with the **db2fodc** command when you observe a problem or invoked automatically whenever a predetermined scenario or symptom is detected. After the diagnostic information has been collected, it is used to help determine the potential causes of the problem. In some cases, you might be able to determine the problem cause yourself, or involvement from IBM support personnel will be required.

Once execution of the **db2fodc** command has finished, the **db2support** tool must be executed to collect the resulting diagnostic files and prepare the FODC package to be submitted to IBM Support. The **db2support** command will collect the contents of all FODC package directories found or specified with the **-fodcpath** parameter. This is done to avoid additional requests, from IBM Support for diagnostic information.

Collecting diagnosis information based on common outage problems

Diagnostic information can be gathered automatically in a first occurrence data collection (FODC) package as the problem that affects an instance, host, or member is occurring. The information in the FODC package can also be collected manually.

Automatic collection of diagnostic information

The database manager invokes the **db2fodc** command for automatic First Occurrence Data Capture (FODC), which in turn invokes one of the DB2 call-out scripts (COS).

To correlate the outage with the DB2 diagnostic logs and the other troubleshooting files, a diagnostic message is written to both the administration notification and the **db2diag** log files. The FODC package directory name includes the FODC_ prefix, the outage type, the timestamp when the FODC directory was created, and the member or partition number where the problem occurred. The FODC package description file is placed in the new FODC package directory.

Table 33. Automatic FODC types and packages

Package	Description	Script executed
FODC_Trap_timestamp_memberNumber	An instance wide trap has occurred	db2cos_trap (.bat)
FODC_Panic_timestamp_memberNumber	Engine detected an incoherence and decided not to continue	db2cos_trap (.bat)
FODC_BadPage_timestamp_memberNumber	A Bad Page has been detected	db2cos_datacorruption (.bat)
FODC_DBMarkedBad_timestamp_memberNumber	A database has been marked bad due to an error	db2cos (.bat)
FODC_IndexError_timestamp_PID_EDUID_memberNumber	An EDU wide index error occurred.	db2cos_indexerror_short (.bat) or db2cos_indexerror_long (.bat)
FODC_Member_timestamp_memberNumber	A member or partition has failed or has received a kill signal	db2cos_member (.bat)

Manual collection of diagnostic information

You use the **db2fodc** command manually when you suspect a problem is occurring. Problem scenarios that you can collect diagnostic data for include apparent system hangs, performance issues, or when an upgrade operation or instance creation did not complete as expected. When the **db2fodc** command is run manually, a new FODC package directory is created. The FODC package directory name includes the FODC_ prefix, the problem scenario, the timestamp when the FODC directory was created, and the member(s) or partition number(s) where FODC was performed.

Table 34. Manual FODC types and packages

Package	Description	Script executed
FODC_Clp_timestamp_member	User invoked db2fodc -clp to collect environment and configuration related information, used to troubleshoot problems related to instance creation.	db2cos_clp script (.bat)
FODC_Connections_timestamp_member	User invoked db2fodc -connections to collect connection-related diagnostic data, used to diagnose problems such as sudden spikes in the number of applications in the executing or compiling state or new database connections being denied.	db2cos_threshold script (.bat)
FODC_Cpu_timestamp_member	User invoked db2fodc -cpu to collect processor-related performance and diagnostic data, used to diagnose problems such as high processor utilization rates, a high number of running processes, or high processor wait times.	db2cos_threshold script (.bat)
FODC_Hang_timestamp_memberList	User invoked db2fodc -hang to collect data for hang troubleshooting (or severe performance)	db2cos_hang (.bat)
FODC_Memory_timestamp_member	User invoked db2fodc -memory to collect memory-related diagnostic data, used to diagnose problems such as no free memory available, swap space being used at a high rate, excessive paging or a suspected a memory leak.	db2cos_threshold script (.bat)
FODC_Perf_timestamp_memberList	User invoked db2fodc -perf to collect data for performance troubleshooting	db2cos_perf (.bat)
FODC_Preupgrade_timestamp_member	User invoked db2fodc -preupgrade to collect performance related information before a critical upgrade or update such as upgrading an instance or updating to the next fix pack	db2cos_preupgrade (.bat)

Table 34. Manual FODC types and packages (continued)

Package	Description	Script executed
Scripts located in FODC_IndexError_ timestamp_PID_EDUID_ memberList	<p>User could issue db2fodc -indexerror <i>FODC_IndexError_directory</i> [basic full] (default is basic) to invoke the db2dart commands in the script(s).</p> <p>On partitioned database environments, use db2_a11 "<<+node#< db2fodc -indexerror <i>FODC_IndexError_directory</i> [basic full]"</p> <p>The <i>node#</i> is the last number in the <i>FODC_IndexError_directory</i> directory name. An absolute path is required when using db2fodc -indexerror with the db2_a11 command.</p>	db2cos_indexerror_long (.bat) or db2cos_indexerror_short (.bat)

First occurrence data capture configuration

First occurrence data capture configuration (FODC) behaviour, including the path used to store the FODC package, is controlled by the *DB2FODC* registry variable, which can be set persistently with the **db2set** command or changed dynamically (in-memory only) through the **db2pdcfg** command. FODC behavior can also be customized by updating the call-out scripts (COS) invoked during FODC.

Each partition or member in the instance has its own FODC settings, and you can control how FODC takes place at the partition or member level. If FODC settings exist both at the member or partition level and at the instance level, the member or partition level settings override the instance level settings. For manual FODC, settings can also be overridden by command line parameters you specify, such as the **-fodcpath** parameter. In partitioned or DB2 pureScale database environments, if you specify a list of members or partitions for manual FODC, the settings for the first member or partition specified are used.

Persistent settings made with the **db2set** command do not become effective until the instance is recycled; dynamic settings made with the **db2pdcfg** command are effective immediately and remain effective in memory until the instance is recycled.

To help you control how FODC packages are handled, several *DB2FODC* registry variable settings are available, but not all settings are available on all platforms. You can control the following behaviors through the *DB2FODC* registry variable:

- Where the generated FODC packages are stored (with the *FODCPATH* setting)
- Whether core dump files are generated or not (with the *DUMPCORE* setting)
- How big core dump files can become (with the *CORELIMIT* setting)
- Where the generated core files are stored (with the *DUMPDIR* setting)

FODC by default invokes a **db2cos** call-out script to collect diagnostic information when the database manager cannot continue processing due to a panic, trap, segmentation violation or exception. To help you control the call-out script that is

invoked during FODC, several COS parameter settings are available. You can control the following behaviors through the COS parameter of the DB2FODC registry variable:

- Whether the **db2cos** script is invoked when the database manager cannot continue processing (with the *ON* and *OFF* setting; the default is *ON*)
- How often the **db2cos** script checks the size of the output files generated (with the *COS_SLEEP* setting)
- How long FODC should wait for the **db2cos** script to finish (with the *COS_TIMEOUT* setting)
- How often the **db2cos** script is invoked during a database manager trap (with the *COS_COUNT* setting)
- Whether the **db2cos** script is enabled when the *SQLO_SIG_DUMP* signal is received (with the *COS_SQLO_SIG_DUMP* setting)

FODC package directory settings (**FODCPATH**)

FODC packages can result in the generation of large volumes of diagnostic data that require space to store and can impose a significant processor usage on the system. You can control what directory path FODC sends diagnostic data to, so that you can pick a directory path with sufficient free space available.

The following order is used to determine what FODC path to use:

Automatic FODC

FODCPATH registry variable setting

The **FODCPATH** parameter for the **DB2FODC** registry variable can be set at the member or partition level and at the instance level. FODC uses the **FODCPATH** parameter setting for each partition or member, if set. If a partition or member level setting does not exist, the instance level setting is used.

No FODC path settings

By default, if you do not specify any **FODCPATH** setting at either the member or instance level, FODC sends diagnostic information to the current diagnostic directory path (**diagpath** or **alt_diagpath**).

Manual FODC

db2fodc -fodcpath command parameter option

When manually invoking the **db2fodc** command, you can indicate the location where the FODC package directory is created by specifying the **-fodcpath** parameter option together with the command. If you specify the **-fodcpath** parameter with a valid path name, the FODC package directory is created in that path.

FODCPATH registry variable setting

If you do not specify the **-fodcpath** parameter with the **db2fodc** command, and you specified a list of partitions or members, the **db2fodc** command uses the **FODCPATH** parameter setting for the **DB2FODC** registry variable of the first partition or member from the list specified. If the value for that **FODCPATH** parameter is not set, **db2fodc** uses the instance level **FODCPATH** setting. If you do not specify the **-fodcpath** parameter and do not specify a list of partitions or members, the **db2fodc** command first tries to use the **FODCPATH** parameter setting for the current partition or member; if not set, the instance level setting is used.

No FODC path settings

By default, if you do not specify any FODC path, first occurrence data capture sends diagnostic information to the current diagnostic directory path (**diagpath** or **alt_diagpath**).

Assume that you have a partitioned database environment with 3 members or partitions (0, 1, and 2). The following example shows how to set the FODC path persistently at the instance level for all 3 partitions or members using the **db2set** command:

```
db2set DB2FODC=FODCPATH=/home/hote149/juntang/FODC
```

FODC path settings can also be performed persistently at the member level for each member, overriding the instance level setting. To make these settings effective, the instance must be recycled. For example, to change the FODC path on member 0, issue the following command:

```
db2set DB2FODC=FODCPATH=/home/hote149/juntang/FODC/FODC0 -i juntang 0
```

If you now want to change the FODC path dynamically on member 1 and member 2, you use the following **db2pdcfg** commands. These settings are effective immediately and remain in memory until the instance is recycled.

```
db2pdcfg -fodc FODCPATH=/home/hote149/juntang/FODC/FODC1 -member 1
```

```
db2pdcfg -fodc FODCPATH=/home/hote149/juntang/FODC/FODC2 -member 2
```

If you want to know what the current FODC settings are for each member or partition in a system, you can use the **db2pdcfg -fodc -member all** command (in the example, output is abridged and only the FODC path output is shown):

```
Database Member 0  
FODC package path (FODCPATH)= /home/hote149/juntang/FODC/FODC0/
```

```
Database Member 1  
FODC package path (FODCPATH)= /home/hote149/juntang/FODC/FODC1/
```

```
Database Member 2  
FODC package path (FODCPATH)= /home/hote149/juntang/FODC/FODC2/
```

Customized data collection

The behavior of data collection by **db2fodc -hang** and **db2fodc -perf** is also controlled by parameters defined in the TOOL OPTIONS section of the DB2 call-out script that is invoked during FODC. These parameters can be customized by changing the script that is executed during FODC.

To customize the data collection on UNIX systems, copy the script placed in `/bin/db2cos_symptom` to `/adm/db2cos_symptom`, where `symptom` is either `hang` or `perf`. Once in this new directory, modify the script as you like. On Windows systems, simply modify the default script `\bin\db2cos_symptom.bat`. On UNIX systems, **db2fodc** first tries to execute the script in `/adm/db2cos_symptom`, and, if it is not found, executes the original script in `/bin/db2cos_symptom`. On Windows systems, the script `\bin\db2cos_symptom.bat` is always executed.

Data collected as part of FODC

First occurrence data capture (FODC) results in the creation of a FODC package directory and subdirectories where diagnostic information is collected. The parent package directory, subdirectories and files that get collected are collectively known as a FODC package.

Files containing diagnostic information that are collected by FODC

FODC collects diagnostic information from a number of sources. The exact diagnostic information captured by FODC depends on the type of problem encountered and might include:

Administration notification log (*instance_name.nfy*)

- Operating system: All
- Default location:
 - Linux and UNIX: Located in the directory specified by the **diagpath** database manager configuration parameter.
 - Windows: Use the Event Viewer Tool (**Start > Control Panel > Administrative Tools > Event Viewer**)
- Created automatically when the instance is created.
- When significant events occur, DB2 writes information to the administration notification log. The information is intended for use by database and system administrators. The type of message recorded in this file is determined by the **notifylevel** configuration parameter.

Note: When the **diagsize** database manager configuration parameter is set to a nonzero value, the single administration notification log file behavior (*instance_name.nfy*) will be changed to a rotating log behavior (*instance_name.N.nfy*).

DB2 diagnostic log (db2diag log file)

- Operating system: All
- Default location: Located in the directory identified by the **diagpath** database manager configuration parameter.
- Created automatically when the instance is created.
- This text file contains diagnostic information about error and warnings encountered by the instance. This information is used for troubleshooting and is intended for technicians at IBM Software Support. The type of message recorded in this file is determined by the **diaglevel** database manager configuration parameter.

Note: When the **diagsize** database manager configuration parameter is set to a nonzero value, the single diagnostic log file behavior (a single db2diag.log file) will be changed to a rotating log behavior (db2diag.N.log).

DB2 administration server (DAS) diagnostic log (db2dasdiag.log)

- Operating system: All
- Default location:
 - Linux and UNIX: Located in DASHOME/das/dump, where DASHOME is the home directory of the DAS owner
 - Windows: Located in "dump" folder, in the DAS home directory. For example: C:\Program Files\IBM\SQLLIB\DB2DAS00\dump
- Created automatically when the DAS is created.
- This text file contains diagnostic information about errors and warnings encountered by the DAS.

DB2 event log (db2eventlog.xxx, where xxx is the database partition number)

- Operating system: All

- Default location: Located in the directory specified by the **diagpath** database manager configuration parameter
- Created automatically when the instance is created.
- The DB2 event log file is a circular log of infrastructure-level events occurring in the database manager. The file is fixed in size, and acts as circular buffer for the specific events that are logged as the instance runs. Every time you stop the instance, the previous event log will be replaced, not appended. If the instance traps, a db2eventlog.XXX.crash file is also generated. These files are intended for use by IBM Software Support.

DB2 callout script (db2cos) output files

- Operating system: All
- Default location: Located in the directory specified by the **diagpath** database manager configuration parameter
- If db2cos scripts are executed as a consequence of an FODC outage, db2cos output files will be placed under the FODC directory that was created in the location specified by the **diagpath** database manager configuration parameter.
- Created automatically when a panic, trap or segmentation violation occurs. Can also be created during specific problem scenarios, as specified using the **db2pdcfg** command.
- The default db2cos script will invoke **db2pd** commands to collect information in an unlatched manner. The contents of the db2cos output files will vary depending on the commands contained in the db2cos script, such as operating system commands and other DB2 diagnosing tools. For more details on the tools that are executed with the db2cos script, open the script file in a text editor.
- The db2cos script is shipped under the **bin/** directory. On UNIX, this directory is read-only. To create your own modifiable version of this script, copy the db2cos script to the **adm/** directory. You are free to modify this version of the script. If the script is found in the **adm/** directory, it is that version that is run. Otherwise, the default version in the **bin/** directory is run.

Dump files

- Operating system: All
- Default location: Located in the directory specified by the **diagpath** database manager configuration parameter
- If these files are dumped during an FODC outage, they will be placed under the FODC directory.
- Created automatically when particular problem scenarios arise.
- For some error conditions, extra information is logged in binary files named after the failing process ID. These files are intended for use by IBM Software Support.

Trap files

- Operating system: All
- Default location: Located in the directory specified by the **diagpath** database manager configuration parameter
- If these files are dumped during an FODC outage, they will be placed under the FODC directory.

- Created automatically when the instance ends abnormally. Can also be created at will using the **db2pd** command.
- The database manager generates a trap file if it cannot continue processing due to a trap, segmentation violation, or exception.

Core files

- Operating system: Linux and UNIX
- Default location: Located in the directory specified by the **diagpath** database manager configuration parameter
- If these files are dumped during an FODC outage, they will be placed under the FODC directory.
- Created by the operating system when the DB2 instance terminates abnormally.
- Among other things, the core image will include most or all of the memory allocations of DB2, which may be required for problem descriptions.

FODC package path and contents

FODC creates the FODC package directory in the FODC path specified. You specify the FODC path through the **FODCPATH** registry variable setting or the **db2fodc -fodcpath** command parameter option. If you do not specify any FODC path, first occurrence data capture sends diagnostic information to the current diagnostic directory path (**diagpath** or **alt_diagpath**). A **db2diag** log file diagnostic message is logged to identify the directory name used for FODC. The capture of diagnostic information can generate a significant volume of diagnostic data, depending on what parameters are specified, and enough space must be available in the directory path where FODC stores diagnostic information. To avoid a scenario where FODC fills all the available space in the file system and impacts your data server, it is recommended that you specify a FODC path where FODC can store the diagnostic data.

For automatic FODC, a package is collected for the member or partition where the problem is occurring; if the problem is occurring on multiple members, multiple packages are collected in separate FODC package directories. The FODC package directory follows the naming convention

FODC_outageType_timestamp_member_number, where *outageType* is the problem symptom, *timestamp* is the time of FODC invocation, and *member_number* is the member or partition number where the problem occurred. For example, when a trap occurs on member 1, FODC might automatically create a package named like **FODC_Trap_ 2010-11-17-20.58.30.695243_0001**.

For manual FODC, a package is collected for the member(s) or partition(s) you specify. The naming convention for the FODC package directory is **FODC_manualOutageType_timestamp_memberList**, where *manualOutageType* is the problem symptom, *timestamp* is the time of FODC invocation, and *memberList* is a list of the members or partitions where the problem occurred. For example, the manually issued command **db2fodc -hang -basic -member 1,2,3 -db sample** creates a manual FODC package for members 1,2 and 3, and might be named like **FODC_hang_ 2010-11-17-20.58.30.695243_0001.0002.0003**.

One or more of the following subdirectories is created under the FODC package directory:

- DB2CONFIG containing DB2 configuration output and files

- DB2PD containing **db2pd** output or output files
- DB2SNAPS containing DB2 snapshots
- DB2TRACE containing DB2 traces
- OS CONFIG containing operating system configuration files
- OSSNAPS containing operating system monitor information
- OSTRACE containing operating system traces

Not all of these directories might exist, depending on your FODC configuration and the outage type for which the **db2fodc** command is run.

FODC sends the following diagnostic information to the FODC package directory:

db2fodc -clp collects the following information:

- Operating system information
- Instance and database configuration information

db2fodc -hang collects the following information:

db2fodc -hang collects the following info:

- Basic operating system information. The problem could be due to OS level, patches, and so on.
- Basic DB2 configuration information.
- Operating system monitor information: vmstat, netstat, iostat, and so on.
 - 2 iterations at least: with timestamps saved
- Partial call stacks: DB2 stack traces of top CPU agents.
- Operating system trace: trace on AIX.
- Diagnostic information collected by **db2pd**.
- DB2 trace.
- Full DB2 call stacks.
- Second round of DB2 configuration information.
 - Including second DB2 trace collection.
- Snapshot information: **db2 get snapshot for** database, applications, tables, and so on.
 - Information will be collected per node in case of multiple logical nodes.

db2fodc -perf monitors the system possibly collecting the following information:

- Snapshots
- Stacktraces
- Virtual Memory (Vmstat)
- Input/Output information (Iostat)
- traces
- Some other information depending on the case. See the script for more details.

db2fodc -indexerror collects the following information:

- Basic Mode
 - db2cos_indexerror_short(.bat) script is run. See script for additional details.

- If applicable **db2dart** commands exist in the script, the **db2dart /DD**, **db2dart /DI**, or both data formatting actions are run with number of pages limited to 100.
- Full Mode
 - db2cos_indexerror_short(.bat) and db2cos_indexerror_long(.bat) scripts are run. See scripts for additional details.
 - If applicable **db2dart** commands exist in the script db2cos_indexerror_short(.bat), the **db2dart /DD**, **db2dart /DI**, or both data formatting actions are run with number of pages limited to 100.
 - If applicable **db2dart** commands exist in the db2cos_indexerror_long(.bat) script, the **db2dart /T** command is run. This command requires the database be offline.

db2fodc -preupgrade collects the following information:

- Operating system information
- Instance and database configuration information, such as output of the **db2level** command, environment variables, output of the **db2 get dbm cfg** command, and the db2nodes.cfg file
- System catalog data and statistics, such as optimizer information collected by the **db2support -d dbname -c -s -cl 0** command
- Operating system monitoring data, such as output of the **netstat -v** and **ps -elf** commands
- System files
- Package information, as returned by the DB2 LIST PACKAGES FOR SCHEMA *schema-name* SHOW DETAIL command for all schema names
- Any FODC_Preupgrade directories found in db2dump/. These directories contain information such as performance data, top dynamic SQL queries, and explain plans
- The logfile from the **db2ckupgrade** command in /tmp/db2ckupgrade.log.*processID*, if it exists
- Output from the **db2prereqcheck** command

The following diagnostic information is also included when you specify the members on which to collect:

- Snapshots (after turning on all monitor switches)
- The **db2pd** command output for the -everything, -agents, -applications, -mempools, and -fcm parameters
- The dynamic SQL statements used most frequently
- The query plans for SQL statements
- The explain plans for static packages

Manual **db2fodc** command invocation results in the creation of a log file named db2fodc_*symptom*.log in the FODC_*symptom* directory, where *symptom* is one of the collection types, such as hang or perf. Inside this file, the **db2fodc** command also stores status information and metadata describing the FODC package inside the FODC subdirectory. This file contains information about the type of FODC, the timestamp of the start and end of data collection, and other information useful for the analysis of the FODC package.

Automatic FODC data generation

When an outage occurs and automatic first occurrence data capture (FODC) is enabled, data is collected based on symptoms. The data collected is specific to what is needed to diagnose the outage.

One or many messages, including those defined as "critical" are used to mark the origin of an outage.

Trap files contain information such as:

- The amount of free virtual storage
- Values associated with the product's configuration parameters and registry variables at the time the trap occurred
- Estimated amount of memory used by the DB2 product at the time of the trap
- Information that provides a context for the outage

The raw stack dump might be included in an ASCII trap file.

Dump files that are specific to components within the database manager are stored in the appropriate FODC package directory.

Monitor and audit facilities using First Occurrence Data Capture (FODC)

If you find you are required to investigate monitor or audit facility problems, there are logs that contain information about the probable cause for the difficulties you may be experiencing.

DB2 audit log ("db2audit.log")

- Operating system: All
- Default location:
 - Windows: Located in the \$DB2PATH\instance_name\security directory
 - Linux and UNIX: Located in the \$HOME\sql1lib\security directory, where \$HOME is the instance owner's home directory
- Created when the **db2audit** facility is started.
- Contains audit records generated by the DB2 audit facility for a series of predefined database events.

DB2 governor log ("mylog.x", where x is the number of database partitions on which the governor is running)

- Operating system: All
- Default location:
 - Windows: Located in the \$DB2PATH\instance_name\log directory
 - Linux and UNIX: Located in the \$HOME\sql1ib\log directory, where \$HOME is the instance owner's home directory
- Created when using the governor utility. The base of the log file name is specified in the **db2gov** command.
- Records information about actions performed by the governor daemon (for example, forcing an application, reading the governor configuration file, starting or ending the utility) as well as errors and warnings.

Event monitor file (for example, "00000000.evt")

- Operating system: All

- Default location: When you create a file event monitor, all of the event records are written to the directory specified in the CREATE EVENT MONITOR statement.
- Generated by the event monitor when events occur.
- Contains event records that are associated with the event monitor.

db2ckbkp command

Use the db2ckbkp command to test the integrity of a backup image and to determine whether it can be restored. Also, use this command to display the metadata stored in the backup header and information about the objects stored in the backup image such as storage paths and table spaces.

db2cklog command

Use the db2cklog command to check the validity of one or a range of archive log files to determine whether to use the log files during the rollforward recovery of a database or table space.

Checking archive log files with the db2cklog tool

Checking your archive log files ensures that known good log files are available in case a rollforward recovery becomes necessary and that the recovery operation does not fail because of a problem with a log file. Also, if the validation fails, you can determine what options are available and decide the course of action.

Before you begin

You need to have read permission on the archive log files, so that the **db2cklog** tool can read the log files and perform its checks. Only log files that are closed, such as archive log files, can be validated successfully. If you run the tool on a log file that is still active, the tool cannot check that file accurately and you will receive a warning to let you know that the file is still active.

About this task

The **db2cklog** tool reads either single log files or a range of numbered log files and performs checks on the internal validity of the files. Log files that pass validation without any error messages or warnings are known good files and you can use them during a rollforward recovery operation. If an archive log file fails validation with an error message or if a warning is returned, then you must not use that log file during rollforward recovery. An archive log file that fails validation cannot be repaired and you should follow the response outlined in this task for what to do next.

Checking your archive log files is useful in the following scenarios:

- Immediately before a rollforward operation is started: If it becomes necessary to perform a rollforward recovery operation, you can first run the **db2cklog** tool against all the archive log files that are required to perform the rollforward operation to ensure that the log files are valid. Running the **db2cklog** tool against the log files beforehand helps avoid a situation where a recovery operation fails partway through because of a problem with a log file, necessitating a follow-on recovery operation.
- Every time a log file is closed and copied to the log archive directory: As part of your day-to-day operations, archive log files can be checked as an added

precaution to make sure that known good log files are always available. With this preventive measure, you know right away whether you need to locate a copy of a log file or if a full database backup to establish a new recovery point is needed. This helps to reduce any delay in the event that a rollforward recovery becomes necessary.

Procedure

To check your archive log files, you issue the **db2cklog** command from the command line and include the log file or files you want checked. Note that you do not specify full log file names with the **db2cklog** command but only the numeric identifiers that are part of the log file names. The numeric identifier of the S0000001.LOG log file is 1, for example; to check that log file, you specify db2cklog 1. If the archive log files are not in the current directory, include the relative or absolute path to the log files with the optional **ARCHLOGPATH** parameter.

1. If you want to check the validity of a single archive log file, you specify the numeric identifier of that log file as *log-file-number1* with the command. For example, to check the validity of the S0000000.LOG log file in the /home/amytang/tests directory, you issue the command db2cklog 0 ARCHLOGPATH /home/amytang/tests.
2. If you want to check the validity of a range of archive log files, you include the first and last numeric identifier of that range with the command (from *log-file-number1* to *log-file-number2*). All log files in the range are checked, unless the upper end of the range specified with *log-file-number2* is numerically lower than the beginning of the range (specified with *log-file-number1*). In that case, only *log-file-number1* is checked. For example, to check the validity of the log files ranging from S0000000.LOG to S0000005.LOG in the /home/nrichers/tests directory, you issue the command db2cklog 0 TO 5 ARCHLOGPATH /home/nrichers/tests

Results

The **db2cklog** tool will return a return code of zero for any file that passes validation. If a range of numbered archive log files is specified, the **db2cklog** tool will read each file in sequence, perform its checks and issue a return code for each file. The tool stops at the first error it encounters, even if a range of log files was specified and there are additional files the tool has not yet checked. The DBT message that is returned when an error is found can provide you with some more information about why an archive log file failed validation, but you cannot fix an invalid log file. If you receive a DBT warning message that a log file might still be active but know for certain that the file is an archive log file, then you should treat the archive log file as invalid and follow the response for what to do next outlined in this task.

Example

The following example shows the typical output of the **db2cklog** command as it parses a log file, in this case S0000002.LOG. This file passes validation with a return code of zero.

```
$ db2cklog 2
```

D B 2 C K L O G

DB2 Check Log File tool
I B M

The db2cklog tool is a utility can be used to test the integrity of an archive log file and to determine whether or not the log file can be used in the rollforward database command.

```
=====
"db2cklog": Processing log file header of "S0000002.LOG"
"db2cklog": Processing log pages of "S0000002.LOG" (total log pages: "316840")
    ==> page "1" ...
    ==> page "25001" ...
    ==> page "50001" ...
    ==> page "75001" ...
    ==> page "100001" ...
    ==> page "125001" ...
    ==> page "150001" ...
    ==> page "175001" ...
    ==> page "200001" ...
    ==> page "225001" ...
    ==> page "250001" ...
    ==> page "275001" ...
    ==> page "300001" ...
"db2cklog": Finished processing log file "S0000002.LOG". Return code: "0".
=====
```

What to do next

If an archive log file fails validation, your response depends on whether or not you have a copy of the log file that can pass validation by the **db2cklog** tool. If you are not sure whether you have a copy of the log file, check the setting for the **logarchmeth2** configuration parameter, which determines whether your database server archives a secondary copy of each log file. If you are validating logs as they are being archive and log mirroring is also configured on your data server, you might still be able to locate a copy of the log file in the log mirror path, as your data server does not recycle log files immediately after archiving.

- If you have a copy of the archive log file, use the **db2cklog** command against that copy. If the copy of the log file passes validation, replace the log file that cannot be read with the valid copy of the log file.
- If you have only one copy of the archive log file and that copy cannot be validated, the log file is beyond repair and cannot be used for rollforward recovery purposes. In this case, you must make a full database backup as soon as possible to establish a new, more recent recovery point that does not depend on the unusable log file for rollforward recovery.

db2ls command

The db2ls command lists where DB2 products are installed on your system and the DB2 product level. It can also list all or specific DB2 products and features for a particular installation path.

Listing DB2 database products installed on your system (Linux and UNIX)

On Linux and UNIX operating systems, use the db2ls command to list the DB2 database products and features installed on your system.

Before you begin

At least one DB2 Version 9 (or later) database product must already be installed by a root user for a symbolic link to the **db2ls** command to be available in the /usr/local/bin directory.

About this task

With the ability to install multiple copies of DB2 database products on your system and the flexibility to install DB2 database products and features in the path of your choice, you need a tool to help you keep track of what is installed and where it is installed. On supported Linux and UNIX operating systems, the **db2ls** command lists the DB2 products and features installed on your system, including the DB2 HTML documentation.

The **db2ls** command can be found both in the installation media and in a DB2 install copy on the system. The **db2ls** command can be run from either location. The **db2ls** command can be run from the installation media for all products except IBM Data Server Driver Package.

The **db2ls** command can be used to list:

- Where DB2 database products are installed on your system and list the DB2 database product level
- All or specific DB2 database products and features in a particular installation path

Restrictions

The output that the **db2ls** command lists is different depending on the ID used:

- When the **db2ls** command is run with root authority, only root DB2 installations are queried.
- When the **db2ls** command is run with a non-root ID, root DB2 installations and the non-root installation owned by matching non-root ID are queried. DB2 installations owned by other non-root IDs are not queried.

The **db2ls** command is the only method to query a DB2 database product. You *cannot* query DB2 database products using Linux or UNIX operating system native utilities, such as **pkginfo**, **rpm**, **SMIT**, or **swlist**. Any existing scripts containing a native installation utility that you use to query and interface with DB2 installations must change.

You *cannot* use the **db2ls** command on Windows operating systems.

Procedure

- To list the path where DB2 database products are installed on your system and list the DB2 database product level, enter:
db2ls

The command lists the following information for each DB2 database product installed on your system:

- Installation path
- Level
- Fix pack
- Special Install Number. This column is used by IBM DB2 Support.
- Installation date. This column shows when the DB2 database product was last modified.
- Installer UID. This column shows the UID with which the DB2 database product was installed.
- To list information about DB2 database products or features in a particular installation path the **q** parameter must be specified:

```
db2ls -q -p -b baseInstallDirectory
```

where:

- **q** specifies that you are querying a product or feature. This parameter is mandatory.
- **p** specifies that the listing displays products rather than listing the features.
- **b** specifies the installation directory of the product or feature. This parameter is mandatory if you are not running the command from the installation directory.

Results

Depending on the parameters provided, the command lists the following information:

- Installation path. This is specified only once, not for each feature.
- The following information is displayed:
 - Response file ID for the installed feature, or if the **p** option is specified, the response file ID for the installed product. For example, **ENTERPRISE_SERVER_EDITION**.
 - Feature name, or if the **p** option is specified, product name.
 - Product version, release, modification level, fix pack level (VRMF). For example, **10.1.0.0**
 - Fix pack, if applicable. For example, if Fix Pack 1 is installed, the value displayed is 1. This includes interim fix packs, such as Fix Pack 1a.
- If any of the product's VRMF information do not match, a warning message displays at the end of the output listing. The message suggests the fix pack to apply.

db2mtrk command

You can use the db2mtrk command to generate a complete report of memory status, for instances, databases, agents, and applications. The command output includes memory pool allocation information such as current size, maximum size, and type of function for which memory is used.

Buffer pools memory allocation

You can use the db2mtrk command to view the amount of database memory that has been allocated to buffer pools. The bufferpool heaps are always fully allocated so the memory tracker reports the same values for the current and maximum sizes

of these heaps. If a bufferpool size is set to automatic then the current and maximum size of the bufferpool heap will be adjusted over time based on workload and available memory.

Example 1

To report the database and instance memory usage every 10 seconds, issue the following command: `db2mtrk -v -i -d -r 10`.

Example 2

In addition to database and instance memory usage, to report detailed application memory usage grouped by application ID, issue the following command: `db2mtrk -a -v -i -d`.

db2pd command

You can use the db2pd command for monitoring and troubleshooting because it can return quick and immediate information from the DB2 memory sets.

Overview

The tool collects information without acquiring any latches or using any engine resources. It is therefore possible (and expected) to retrieve information that is changing while **db2pd** is collecting information; hence the data might not be completely accurate. If changing memory pointers are encountered, a signal handler is used to prevent **db2pd** from ending abnormally. This can result in messages such as "Changing data structure forced command termination" to appear in the output. Nonetheless, the tool can be helpful for troubleshooting. Two benefits to collecting information without latching include faster retrieval and no competition for engine resources.

If you want to capture information about the database management system when a specific SQLCODE, ZRC code or ECF code occurs, this can be accomplished using the **db2pdcfg -catch** command. When the errors are caught, the db2cos (callout script) is launched. The db2cos script can be dynamically altered to run any **db2pd** command, operating system command, or any other command needed to resolve the problems. The template db2cos script file is located in sqllib/bin on UNIX and Linux. On the Windows operating system, db2cos is located in the \$DB2PATH\bin directory.

When adding a new node, you can monitor the progress of the operation on the database partition server, that is adding the node, using the **db2pd -addnode** command with the optional **oldviewapps** and **detail** parameters for more detailed information.

If you require a list of event monitors that are currently active or have been, for some reason, deactivated, run the **db2pd -gfw** command. This command also returns statistics and information about the targets, into which event monitors write data, for each fast writer EDU.

Examples

The following list is a collection of examples in which the **db2pd** command can be used to expedite troubleshooting:

- Example 1: Diagnosing a lockwait

- Example 2: Using the **-locks** parameter to capture all the locks being waited on
- Example 3: Using the **-apinfo** parameter to capture detailed runtime information about the lock owner and the lock waiter
- Example 4: Using the callout scripts when considering a locking problem
- Example 5: Mapping an application to a dynamic SQL statement
- Example 6: Monitoring memory usage
- Example 7: Determine which application is using up your table space
- Example 8: Monitoring recovery
- Example 9: Determining the amount of resources a transaction is using
- Example 10: Monitoring log usage
- Example 11: Viewing the sysplex list
- Example 12: Generating stack traces
- Example 13: Viewing memory statistics for a database partition
- Example 14: Monitoring the progress of index reorganization
- Example 15: Displaying the top EDUs by processor time consumption and displaying EDU stack information
- Example 16: Displaying agent event metrics

The results text show in the examples is an extract of the the **db2cmd** command output for better readability.

Example 1: Diagnosing a lockwait

If you run **db2pd -db databaseName -locks -transactions -applications -dynamic**, the results are similar to the following ones:

Locks:

TranHdl	Lockname	Type	Mode	Sts	Owner	Dur	HldCnt	Att	ReleaseFlg
3	00020002000000040000000052	Row	..X	G	3	1	0	0x0000	0x40000000
2	00020002000000040000000052	Row	..X	W*	2	1	0	0x0000	0x40000000

For the database that you specified using the **-db** database name option, the first results show the locks for that database. The results show that TranHdl 2 is waiting on a lock held by TranHdl 3.

Transactions:

AppHndl	[nod-index]	TranHdl	Locks	State	Tflag	Tflag2	...
11	[000-00011]	2	4	READ	0x00000000	0x00000000	...
12	[000-00012]	3	4	WRITE	0x00000000	0x00000000	...

We can see that TranHdl 2 is associated with AppHndl 11 and TranHdl 3 is associated with AppHndl 12.

Applications:

AppHndl	NumAgents	CoorPid	Status	C-AnchID	C-StmtUID	L-AnchID	L-StmtUID	Appid
12	1	1073336	UOW-Waiting	0	0	17	1	...5602
11	1	1040570	UOW-Executing	17	1	94	1	...5601

We can see that AppHndl 12 last ran dynamic statement 17, 1. AppHndl 11 is currently running dynamic statement 17, 1 and last ran statement 94, 1.

Dynamic SQL Statements:

AnchID	StmtUID	NumEnv	NumVar	NumRef	NumExe	Text
17	1	1	1	2	2	update pdtest set c1 = 5
94	1	1	1	2	2	set lock mode to wait 1

We can see that the text column shows the SQL statements that are associated with the lock timeout.

Example 2: Using the **-wlocks** parameter to capture all the locks being waited on

If you run **db2pd -wlocks -db pdtest**, results similar to the following ones are generated. They show that the first application (AppHandl 47) is performing an insert on a table and that the second application (AppHandl 46) is performing a select on that table:

```
venus@boson:/home/venus =>db2pd -wlocks -db pdtest

Database Partition 0 -- Database PDTEST -- Active -- Up 0 days 00:01:22

Locks being waited on :
AppHandl TranHdl Lockname          Type Mode Conv Sts CoorEDU AppName AuthID AppID
47      8      00020004000000000840000652 Row  ..X      G   5160  db2bp  VENUS ...13730
46      2      00020004000000000840000652 Row  .NS     W   5913  db2bp  VENUS ...13658
```

Example 3: Using the **-apinfo** parameter to capture detailed runtime information about the lock owner and the lock waiter

The following sample output was generated under the same conditions as those for Example 2:

```
venus@boson:/home/venus =>db2pd -apinfo 47 -db pdtest

Database Partition 0 -- Database PDTEST -- Active -- Up 0 days 00:01:30

Application :
  Address :          0x0780000001676480
  AppHandl [nod-index] : 47      [000-00047]
  Application PID :    876558
  Application Node Name : boson
  IP Address:        n/a
  Connection Start Time : (1197063450)Fri Dec 7 16:37:30 2007
  Client User ID :    venus
  System Auth ID :    VENUS
  Coordinator EDU ID : 5160
  Coordinator Partition : 0
  Number of Agents :   1
  Locks timeout value : 4294967294 seconds
  Locks Escalation :   No
  Workload ID :        1
  Workload Occurrence ID : 2
  Trusted Context :   n/a
  Connection Trust Type : non trusted
  Role Inherited :   n/a
  Application Status : UOW-Waiting
  Application Name :  db2bp
  Application ID :    *LOCAL.venus.071207213730

  ClientUserID :       n/a
  ClientWrkstnName :  n/a
  ClientApplName :    n/a
  ClientAccntng :     n/a

List of inactive statements of current UOW :
  UOW-ID :           2
  Activity ID :      1
  Package Schema :  NULLID
  Package Name :    SQLC2G13
  Package Version :
  Section Number :  203
  SQL Type :        Dynamic
  Isolation :       CS
  Statement Type :  DML, Insert/Update/Delete
  Statement :        insert into pdtest values 99
```

```

venus@boson:/home/venus =>db2pd -apinfo 46 -db pdtest

Database Partition 0 -- Database PDTEST -- Active -- Up 0 days 00:01:39

Application :
  Address :          0x0780000000D77A60
  AppHandle [nod-index] : 46           [000-00046]
  Application PID :    881102
  Application Node Name : boson
  IP Address:          n/a
  Connection Start Time : (1197063418)Fri Dec 7 16:36:58 2007
  Client User ID :     venus
  System Auth ID :     VENUS
  Coordinator EDU ID : 5913
  Coordinator Partition : 0
  Number of Agents :   1
  Locks timeout value : 4294967294 seconds
  Locks Escalation :   No
  Workload ID :        1
  Workload Occurrence ID : 1
  Trusted Context :   n/a
  Connection Trust Type : non trusted
  Role Inherited :   n/a
  Application Status : Lock-wait
  Application Name :  db2bp
  Application ID :    *LOCAL.venus.071207213658

  ClientUserID :       n/a
  ClientWrkstnName :  n/a
  ClientApplName :    n/a
  ClientAcntng :      n/a

List of active statements :
  *UOW-ID :            3
  Activity ID :        1
  Package Schema :    NULLID
  Package Name :       SQLC2G13
  Package Version :   :
  Section Number :    201
  SQL Type :          Dynamic
  Isolation :         CS
  Statement Type :    DML, Select (blockable)
  Statement :          select * from pdtest

```

Example 4: Using the callout scripts when considering a locking problem

To use the callout scripts, find the db2cos output files. The location of the files is controlled by the database manager configuration parameter **diagpath**. The contents of the output files will differ depending on what commands you enter in the db2cos script file. An example of the output provided when the db2cos script file contains a **db2pd -db sample -locks** command is as follows:

```

Lock Timeout Caught
Thu Feb 17 01:40:04 EST 2006
Instance DB2
Database: SAMPLE
Partition Number: 0
PID: 940
TID: 2136
Function: sqlplnfd
Component: lock manager
Probe: 999
Timestamp: 2006-02-17-01.40.04.106000
AppID: *LOCAL.DB2...
AppHandle:
...
Database Partition 0 -- Database SAMPLE -- Active -- Up 0 days 00:06:53

```

```

Locks:
Address   TranHdl Lockname          Type Mode Sts Owner Dur HldCnt Att Rlse
0x402C6B30 3      0002000300000004000000052 Row ..X W* 3     1   0     0   0x40

```

In the output, W* indicates the lock that experienced the timeout. In this case, a lockwait has occurred. A lock timeout can also occur when a lock is being converted to a higher mode. This is indicated by C* in the output.

You can map the results to a transaction, an application, an agent, or even an SQL statement with the output provided by other **db2pd** commands in the db2cos file. You can narrow down the output or use other commands to collect the information that you need. For example, you can use the **db2pd -locks wait** parameters to print only locks with a wait status. You can also use the **-app** and **-agent** parameters.

Example 5: Mapping an application to a dynamic SQL statement

The command **db2pd -applications -dynamic** reports the current and last anchor ID and statement unique ID for dynamic SQL statements. This allows direct mapping from an application to a dynamic SQL statement.

Applications:

Address	AppHandle	[nod-index]	NumAgents	CoorPid	Status
0x00000002006D2120	780	[000-00780]	1	10615	UOW-Executing

C-AnchID	C-StmtUID	L-AnchID	L-StmtUID	Appid
163	1	110	1	*LOCAL.burford.050202200412

Dynamic SQL Statements:

Address	AnchID	StmtUID	NumEnv	NumVar	NumRef	NumExe	Text
0x0000000220A02760	163	1	2	2	1		CREATE VIEW MYVIEW
0x0000000220A0B460	110	1	2	2	2	1	CREATE VIEW YOURVIEW

Example 6: Monitoring memory usage

The **db2pd -memblock** command can be useful when you are trying to understand memory usage, as shown in the following sample output:

All memory blocks in DBMS set.

Address	PoolID	PoolName	BlockAge	Size(Bytes)	I	LOC	File
0x07800000000740068	62	resynch	2	112	1	1746	1583816485
0x07800000000725688	62	resynch	1	108864	1	127	1599127346
0x078000000001F4348	57	ostrack	6	5160048	1	3047	698130716
0x078000000001B5608	57	ostrack	5	240048	1	3034	698130716
0x078000000001A0068	57	ostrack	1	80	1	2970	698130716
0x078000000001A00E8	57	ostrack	2	240	1	2983	698130716
0x078000000001A0208	57	ostrack	3	80	1	2999	698130716
0x078000000001A0288	57	ostrack	4	80	1	3009	698130716
0x07800000000700068	70	apmh	1	360	1	1024	3878879032
0x078000000007001E8	70	apmh	2	48	1	914	1937674139
0x07800000000700248	70	apmh	3	32	1	1000	1937674139
...							

This is followed by the sorted 'per-pool' output:

Memory blocks sorted by size for ostrack pool:					
PoolID	PoolName	TotalSize(Bytes)	TotalCount	LOC	File
57	ostrack	5160048	1	3047	698130716
57	ostrack	240048	1	3034	698130716
57	ostrack	240	1	2983	698130716
57	ostrack	80	1	2999	698130716
57	ostrack	80	1	2970	698130716
57	ostrack	80	1	3009	698130716

Total size for ostrack pool: 5400576 bytes

Memory blocks sorted by size for apmh pool:

PoolID	PoolName	TotalSize(Bytes)	TotalCount	LOC	File
70	apmh	40200	2	121	2986298236
70	apmh	10016	1	308	1586829889
70	apmh	6096	2	4014	1312473490
70	apmh	2516	1	294	1586829889
70	apmh	496	1	2192	1953793439
70	apmh	360	1	1024	3878879032
70	apmh	176	1	1608	1953793439
70	apmh	152	1	2623	1583816485
70	apmh	48	1	914	1937674139
70	apmh	32	1	1000	1937674139

Total size for apmh pool: 60092 bytes

...

The final section of output sorts the consumers of memory for the entire memory set:

All memory consumers in DBMS memory set:

PoolID	PoolName	TotalSize(Bytes)	%Bytes	TotalCount	%Count	LOC	File
57	ostrack	5160048	71.90	1	0.07	3047	698130716
50	sqlch	778496	10.85	1	0.07	202	2576467555
50	sqlch	271784	3.79	1	0.07	260	2576467555
57	ostrack	240048	3.34	1	0.07	3034	698130716
50	sqlch	144464	2.01	1	0.07	217	2576467555
62	resynch	108864	1.52	1	0.07	127	1599127346
72	eduah	108048	1.51	1	0.07	174	4210081592
69	krcbh	73640	1.03	5	0.36	547	4210081592
50	sqlch	43752	0.61	1	0.07	274	2576467555
70	apmh	40200	0.56	2	0.14	121	2986298236
69	krcbh	32992	0.46	1	0.07	838	698130716
50	sqlch	31000	0.43	31	2.20	633	3966224537
50	sqlch	25456	0.35	31	2.20	930	3966224537
52	kerh	15376	0.21	1	0.07	157	1193352763
50	sqlch	14697	0.20	1	0.07	345	2576467555
...							

You can also report memory blocks for private memory on UNIX and Linux operating systems. For example, if you run **db2pd -memb pid=159770**, results similar to the following ones are generated:

All memory blocks in Private set.

PoolID	PoolName	BlockAge	Size(Bytes)	I	LOC	File
88	private	1	2488	1	172	4283993058
88	private	2	1608	1	172	4283993058
88	private	3	4928	1	172	4283993058
88	private	4	7336	1	172	4283993058
88	private	5	32	1	172	4283993058
88	private	6	6728	1	172	4283993058
88	private	7	168	1	172	4283993058
88	private	8	24	1	172	4283993058
88	private	9	408	1	172	4283993058
88	private	10	1072	1	172	4283993058
88	private	11	3464	1	172	4283993058
88	private	12	80	1	172	4283993058
88	private	13	480	1	1534	862348285
88	private	14	480	1	1939	862348285
88	private	80	65551	1	1779	4231792244

Total set size: 94847 bytes

Memory blocks sorted by size:

PoolID	PoolName	TotalSize(Bytes)	TotalCount	LOC	File
88	private	65551	1	1779	4231792244

```

88      private   28336          12        172  4283993058
88      private    480           1        1939  862348285
88      private    480           1        1534  862348285
Total set size: 94847 bytes

```

Example 7: Determine which application is using up your table space

Using **db2pd -tcbstats** command, you can identify the number of inserts for a table. The following example shows sample information for a user-defined global temporary table called TEMP1:

```

TCB Table Information:
TbspaceID TableID PartID ... TableName SchemaNm ObjClass DataSize LfSize LobSize XMLSize
3          2       n/a     ... TEMP1   SESSION Temp      966      0      0      0

TCB Table Stats:
TableName Scans UDI PgReorgs ... Reads FscrUpdates Inserts Updates Deletes OvFlReads OvFlCrtes
TEMP1      0      0      0      ... 0      0      43968  0      0      0      0      0

```

You can then obtain the information for table space 3 by using the **db2pd -tablespaces** command. Sample output is as follows:

```

Tablespace 3 Configuration:
Type Content PageSz ExtentSz Auto Prefetch BufID FSC NumCntrs MaxStripe LastConsecPg Name
DMS UsrTmp 4096   32      Yes 32      1   On 1      0      31      TEMPSPACE2

Tablespace 3 Statistics:
TotalPgs UsablePgs UsedPgs PndFreePgs FreePgs HWM State      MinRecTime NQuiescers
5000     4960      1088   0      3872  1088 0x00000000 0      0

Tablespace 3 Autoresize Statistics:
AS AR InitSize IncSize IIP MaxSize LastResize LRF
No No 0          0      No 0      None      No

Containers:
ContainNum Type TotalPgs UseablePgs StripeSet Container
0         File  5000     4960      0      /home/db2inst1/tempspace2a

```

The **MinRecTime** column returns a value that is a UNIX time stamp in a Coordinated Universal Time (UTC) timezone format. To convert the value to a GMT time zone format, you can use the DB2 time stamp function. For example, if **MinRecTime** returns a value of 1369626329, to convert this value to a GMT format run the following statement:

```
db2 "values timestamp('1970-01-01-00.00.00') + 1369626329 seconds"
```

The query will return a GMT value of 2013-05-27-03.45.29.000000.

The **FreePgs** column shows that space is filling up. As the free pages value decreases, there is less space available. Notice also that the value for **FreePgs** plus the value for **UsedPgs** equals the value of **UsablePgs**.

Once this is known, you can identify the dynamic SQL statement that is using the table TEMP1 by running the **db2pd -db sample -dyn**:

```
Database Partition 0 -- Database SAMPLE -- Active -- Up 0 days 00:13:06
```

```

Dynamic Cache:
Current Memory Used          1022197
Total Heap Size               1271398
Cache Overflow Flag           0
Number of References          237
Number of Statement Inserts   32
Number of Statement Deletes   13
Number of Variation Inserts   21
Number of Statements           19

```

Dynamic SQL Statements:

```

AnchID StmtUID NumEnv NumVar NumRef NumExe Text
78      1      2      2      3      2      declare global temporary table temp1 ...
253    1      1      1     24     24      insert into session.temp1 values('TEST')

```

Finally, you can map the information from the preceding output to the applications output to identify the application by running **db2pd -db sample -app**.

Applications:

AppHandle [nod-index]	NumAgents	CoorPid	Status	C-AnchID	C-StmtUID
501 [000-00501]	1	11246	UOW-Waiting	0	0
L-AnchID L-StmtUID Appid					
253 1					*LOCAL.db2inst1.050202160426

You can use the anchor ID (AnchID) value that identified the dynamic SQL statement to identify the associated application. The results show that the last anchor ID (L-AnchID) value is the same as the anchor ID (AnchID) value. You use the results from one run of **db2pd** in the next run of **db2pd**.

The output from **db2pd -agent** shows the number of rows read (in the Rowsread column) and rows written (in the Rowswrtn column) by the application. These values give you an idea of what the application has completed and what the application still has to complete, as shown in the following sample output:

AppHandle [nod-index]	AgentPid	Priority	Type	DBName		
501 [000-00501]	11246	0	Coord	SAMPLE		
State ClientPid Userid ClientNm Rowsread Rowswrtn LkTm0t						
Inst-Active 26377 db2inst1 db2bp 22 9588 NotSet						

You can map the values for AppHandle and AgentPid resulting from running the **db2pd -agent** command to the corresponding values for AppHandle and CoorPid resulting from running the **db2pd -app** command.

The steps are slightly different if you suspect that an internal temporary table is filling up the table space. You still use **db2pd -tcbspace** to identify tables with large numbers of inserts, however. Following is sample information for an implicit temporary table:

TCB Table Information:									
TbspaceID	TableID	PartID	MasterTbs	MasterTab	TableName	SchemaNm	ObjClass	DataSize	...
1	2	n/a	1	2	TEMP (00001,00002)	<30>	<JMC Temp	2470	...
1	3	n/a	1	3	TEMP (00001,00003)	<31>	<JMC Temp	2367	...
1	4	n/a	1	4	TEMP (00001,00004)	<30>	<JMC Temp	1872	...

TCB Table Stats:									
TableName	Scans	UDI	PgReorgs	NoChgUpdts	Reads	FscrUpdates	Inserts	...	
TEMP (00001,00002)	0	0	0	0	0	0	43219	...	
TEMP (00001,00003)	0	0	0	0	0	0	42485	...	
TEMP (00001,00004)	0	0	0	0	0	0	0	...	

In this example, there are a large number of inserts for tables with the naming convention TEMP (TbspaceID, TableID). These are implicit temporary tables. The values in the SchemaNm column have a naming convention of the value for AppHandle concatenated with the value for SchemaNm, which makes it possible to identify the application doing the work.

You can then map that information to the output from **db2pd -tablespaces** to see the used space for table space 1. Take note of the relationship between the UsedPgs and UsablePgs values in the table space statistics in the following output:

Tablespace Configuration:												
Id	Type	Content	PageSz	ExtentSz	Auto	Prefetch	...	FSC	NumCntrs	MaxStripe	LastConsecPg	Name
1	SMS	SysTmp	4096	32	Yes	320	...	On	10	0	31	TEMPSPACE1

Tablespace Statistics:												
Tablespace	UsedPgs	UsablePgs	AllocPgs	FreePgs	Extents	AllocExtents	FreeExtents	AllocSize	FreeSize	Alloced	AllocedExtents	AllocedSize
TEMPSPACE1	43219	42485	0	0	10	10	0	40960	40960	On	10	40960

```

Id TotalPgs  UsablePgs UsedPgs    PndFreePgs FreePgs    HWM     State      MinRecTime NQuiescers
1   6516      6516      6516       0           0          0         0x00000000 0          0

Tablespace Autoresize Statistics:
Address      Id AS AR InitSize   IncSize   IIP MaxSize   LastResize LRF
0x07800000203FB5A0 1 No No 0        0        No 0        None      No

Containers:
...

```

You can then identify application handles 30 and 31 (because you saw them in the **-tcbstats** output) by using the command **db2pd -app:**

```

Applications:
AppHandle NumAgents CoorPid Status      C-AnchID C-StmtUID L-AnchID L-StmtUID Appid
31        1          4784182 UOW-Waiting 0          0          107      1          ...4142
30        1          8966270 UOW-Executing 107      1          107      1          ...4013

```

Finally, map the information from the preceding output to the Dynamic SQL output obtained by running the **db2pd -dyn** command:

```

Dynamic SQL Statements:
AnchID StmtUID NumEnv NumVar NumRef NumExe Text
107      1          1          1        43      43      select c1, c2 from test group by c1,c2

```

Example 8: Monitoring recovery

If you run the command **db2pd -recovery**, the output shows several counters that you can use to verify that recovery is progressing, as shown in the following sample output. The Current Log and Current LSO values provide the log position. The CompletedWork value is the number of bytes completed thus far.

```

Recovery:
Recovery Status      0x00000401
Current Log          S0000005.LOG
Current LSN          0000001F07BC
Current LSO          000002551BEA
Job Type             ROLLFORWARD RECOVERY
Job ID               7
Job Start Time       (1107380474) Wed Feb 2 16:41:14 2005
Job Description       Database Rollforward Recovery
Invoker Type         User
Total Phases         2
Current Phase        1

Progress:
Address      PhaseNum Description StartTime      CompletedWork TotalWork
0x0000000200667160 1      Forward   Wed Feb 2 16:41:14 2005 2268098 bytes Unknown
0x0000000200667258 2      Backward  NotStarted      0 bytes      Unknown

```

Example 9: Determining the amount of resources a transaction is using

If you run the command **db2pd -transactions**, the output shows the number of locks, the first log sequence number (LSN), the last LSN, the first LSO, the last LSO, the log space used, and the space reserved, as shown in the following sample output. This can be useful for understanding the behavior of a transaction.

```

Transactions:
Address      AppHandle [nod-index] TranHdl Locks State Tflag
0x000000022026D980 797      [000-00797] 2      108    WRITE  0x00000000
0x000000022026E600 806      [000-00806] 3      157    WRITE  0x00000000
0x000000022026F280 807      [000-00807] 4      90     WRITE  0x00000000

Tflag2      Firstlsn      Lastlsn      Firstlso      Lastlso
0x00000000 0x0000001A4212 0x00000001C2022 0x000001072262 0x0000010B2C8C
0x00000000 0x000000107320 0x00000001S3462 0x000001057574 0x0000010B3340
0x00000000 0x0000001BC00C 0x00000001X2F03 0x00000107CF0C 0x0000010B2FDE
LogSpace    SpaceReserved  TID          AxRegCnt    GXID

```

```

4518      95450      0x000000000451 1      0
6576      139670     0x0000000003E0 1      0
3762      79266      0x000000000472 1      0

```

Example 10: Monitoring log usage

The command **db2pd -logs** is useful for monitoring log usage for a database. By using the Pages Written value, as shown in the following sample output, you can determine whether the log usage is increasing:

```

Logs:
Current Log Number      2
Pages Written           846
Method 1 Archive Status Success
Method 1 Next Log to Archive 2
Method 1 First Failure   n/a
Method 2 Archive Status   Success
Method 2 Next Log to Archive 2
Method 2 First Failure   n/a

Address          StartLSN       StartLSO       State    Size  Pages  Filename
0x000000023001BF58 0x00000022F032 0x000001B58000 0x00000000 1000 1000  S0000002.LOG
0x000000023001BE98 0x000000000000 0x000001F40000 0x00000000 1000 1000  S0000003.LOG
0x0000000230008F58 0x000000000000 0x000002328000 0x00000000 1000 1000  S0000004.LOG

```

You can identify two types of problems by using this output:

- If the most recent log archive fails, Archive Status is set to a value of Failure. If there is an ongoing archive failure, preventing logs from being archived at all, Archive Status is set to a value of First Failure.
- If log archiving is proceeding very slowly, the Next Log to Archive value is lower than the Current Log Number value. If archiving is very slow, space for active logs might run out, which in turn might prevent any data changes from occurring in the database.

Note: S0000003.LOG and S0000004.LOG do not contain any log records yet and therefore the StartLSN is 0x0

Example 11: Viewing the sysplex list

Without the **db2pd -sysplex** command showing the following sample output, the only other way to report the sysplex list is by using a DB2 trace.

```

Sysplex List:
Alias:        HOST
Location Name: HOST1
Count:        1

IP Address    Port     Priority  Connections Status      PRDID
1.2.34.56     400      1          0            0

```

Example 12: Generating stack traces

You can use the **db2pd -stack all** command for Windows operating systems or the **-stack** command for UNIX operating systems to produce stack traces for all processes in the current database partition. You might want to use this command iteratively when you suspect that a process or thread is looping or hanging.

You can obtain the current call stack for a particular engine dispatchable unit (EDU) by issuing the command **db2pd -stack eduid**, as shown in the following example:

```
Attempting to dump stack trace for eduid 137.  
See current DIAGPATH for trapfile.
```

If the call stacks for all of the DB2 processes are desired, use the command **db2pd -stack all**, for example (on Windows operating systems):

```
Attempting to dump all stack traces for instance.  
See current DIAGPATH for trapfiles.
```

If you are using a partitioned database environment with multiple physical nodes, you can obtain the information from all of the partitions by using the command **db2_a11 "; db2pd -stack all"**. If the partitions are all logical partitions on the same machine, however, a faster method is to use **db2pd -a11dbp -stacks**.

You can also redirect the output of the **db2pd -stacks** command for **db2sysc** processes to a specific directory path with the **dumpdir** parameter. The output can be redirected for a specific duration only with the **timeout** parameter. For example, to redirect the output of stack traces for all EDUs in **db2sysc** processes to /home/waleed/mydir for 30 seconds, issue the following command:

```
db2pd -a11dbp -stack all dumpdir=/home/waleed/mydir timeout=30
```

Example 13: Viewing memory statistics for a database partition

The **db2pd -dbptnmem** command shows how much memory the DB2 server is currently consuming and, at a high level, which areas of the server are using that memory.

The following example shows the output from running **db2pd -dbptnmem** on an AIX machine:

```
Database Partition Memory Controller Statistics
```

```
Controller Automatic: Y  
Memory Limit: 122931408 KB  
Current usage: 651008 KB  
HWM usage: 651008 KB  
Cached memory: 231296 KB
```

The descriptions of these data fields and columns are as follows:

Controller Automatic

Indicates the memory controller setting. It shows the value "Y" if the **instance_memory** configuration parameter is set to AUTOMATIC. This means that database manager automatically determines the upper boundary of memory consumption.

Memory Limit

If an instance memory limit is enforced, the value of the **instance_memory** configuration parameter is the upper bound limit of DB2 server memory that can be consumed.

Current usage

The amount of memory the server is currently consuming.

HWM usage

The high water mark (HWM) or peak memory usage that has been consumed since the activation of the database partition (when the **db2start** command was run).

Cached memory

The amount of the current usage that is not currently being used but is cached for performance reasons for future memory requests.

Following is the continuation of the sample output from running **db2pd -dbptnmem** on an AIX operating system:

Individual Memory Consumers:

Name	Mem Used (KB)	HWM Used (KB)	Cached (KB)
<hr/>			
APPL-DBONE	160000	160000	159616
DBMS-name	38528	38528	3776
FMP_RESOURCES	22528	22528	0
PRIVATE	13120	13120	740
FCM_RESOURCES	10048	10048	0
LCL-p606416	128	128	0
DB-DBONE	406656	406656	67200

All registered “consumers” of memory within the DB2 server are listed with the amount of the total memory they are consuming. The column descriptions are as follows:

Name A short, distinguishing name of a consumer of memory, such as the following ones:

APPL-*dbname*

Application memory consumed for database *dbname*

DBMS-*name*

Global database manager memory requirements

FMP_RESOURCES

Memory required to communicate with **db2fmps**

PRIVATE

Miscellaneous private memory requirements

FCM_RESOURCES

Fast Communication Manager resources

LCL-*pid*

The memory segment used to communicate with local applications

DB-*dbname*

Database memory consumed for database *dbname*

Mem Used (KB)

The amount of memory that is currently allotted to the consumer

HWM Used (KB)

The high-water mark (HWM) of the memory, or the peak memory, that the consumer has used

Cached (KB)

Of the Mem Used (KB), the amount of memory that is not currently being used but is immediately available for future memory allocations

Example 14: Monitoring the progress of index reorganization

In DB2 Version 9.8 Fix Pack 3 and later fix packs, the progress report of an index reorganization has the following characteristics:

- The **db2pd -reorgs index** command reports index reorg progress for partitioned indexes (Fix Pack 1 introduced support for only non-partitioned indexes).

- The **db2pd -reorgs index** command supports the monitoring of index reorg at the partition level (that is, during reorganization of a single partition).
- The reorg progress for non-partitioned and partitioned indexes is reported in separate outputs. One output shows the reorg progress for non-partitioned indexes, and the following outputs show the reorg progress for partitioned indexes on each table partition; the index reorg statistics of only one partition is reported in each output.
- Non-partitioned indexes are processed first, followed by partitioned indexes in serial fashion.
- The **db2pd -reorgs index** command displays the following additional information fields in the output for partitioned indexes:
 - MaxPartition - Total number of partitions for the table being processed. For partition-level reorg, MaxPartition will always have a value of 1 since only a single partition is being reorganized.
 - PartitionID - The data partition identifier for the partition being processed.

The following example shows an output obtained using the **db2pd -reorgs index** command which reports the index reorg progress for a range-partitioned table with 2 partitions.

Note: The first output reports the Index Reorg Stats of the non-partitioned indexes. The following outputs report the Index Reorg Stats of the partitioned indexes on each partition.

```

Index Reorg Stats:
Retrieval Time: 02/08/2010 23:04:21
TbspaceID: -6      TableID: -32768
Schema: ZORAN    TableName: BIGRPT
Access: Allow none
Status: Completed
Start Time: 02/08/2010 23:03:55   End Time: 02/08/2010 23:04:04
Total Duration: 00:00:08
Prev Index Duration: -
Cur Index Start: -
Cur Index: 0          Max Index: 2           Index ID: 0
Cur Phase: 0          ( - )     Max Phase: 0
Cur Count: 0          Max Count: 0
Total Row Count: 750000

Retrieval Time: 02/08/2010 23:04:21
TbspaceID: 2      TableID: 5
Schema: ZORAN    TableName: BIGRPT
PartitionID: 0      MaxPartition: 2
Access: Allow none
Status: Completed
Start Time: 02/08/2010 23:04:04   End Time: 02/08/2010 23:04:08
Total Duration: 00:00:04
Prev Index Duration: -
Cur Index Start: -
Cur Index: 0          Max Index: 2           Index ID: 0
Cur Phase: 0          ( - )     Max Phase: 0
Cur Count: 0          Max Count: 0
Total Row Count: 375000

Retrieval Time: 02/08/2010 23:04:21
TbspaceID: 2      TableID: 6
Schema: ZORAN    TableName: BIGRPT
PartitionID: 1      MaxPartition: 2
Access: Allow none
Status: Completed
Start Time: 02/08/2010 23:04:08   End Time: 02/08/2010 23:04:12
Total Duration: 00:00:04
Prev Index Duration: -

```

```

Cur Index Start: -
Cur Index: 0           Max Index: 2           Index ID: 0
Cur Phase: 0           ( - )   Max Phase: 0
Cur Count: 0           Max Count: 0
Total Row Count: 375000

```

Example 15: Displaying the top EDUs by processor time consumption and displaying EDU stack information

If you issue the **db2pd** command with the **-edus** parameter option, the output lists all engine dispatchable units (EDUs). Output for EDUs can be returned at the level of granularity you specify, such as at the instance level or at the member. On Linux and UNIX operating systems only, you can also specify the **interval** parameter suboption so that two snapshots of all EDUs are taken, separated by an interval you specify. When the **interval** parameter is specified, two additional columns in the output indicate the delta of processor user time (USR DELTA column) and the delta of processor system time (SYS DELTA column) across the interval.

In the following example, the deltas for processor user time and processor system time are given across a five-second interval:

```

$ db2pd -edus interval=5

Database Partition 0 -- Active -- Up 0 days 00:53:29 -- Date 06/04/2010 03:34:59

List of all EDUs for database partition 0

db2sysc PID: 1249522
db2wdog PID: 2068678

```

EDU ID	TID	Kernel TID	EDU Name	USR	SYS	USR DELTA	SYS DELTA
6957	6957	13889683	db2agntdp (SAMPLE) 0	58.238506	0.820466	1.160726	0.014721
6700	6700	11542589	db2agent (SAMPLE) 0	52.856696	0.754420	1.114821	0.015007
5675	5675	4559055	db2agntdp (SAMPLE) 0	60.386779	0.854234	0.609233	0.014304
3088	3088	13951225	db2agntdp (SAMPLE) 0	80.073489	2.249843	0.499766	0.006247
3615	3615	2887875	db2loggw (SAMPLE) 0	0.939891	0.410493	0.011694	0.004204
4900	4900	6344925	db2pfchr (SAMPLE) 0	1.748413	0.014378	0.014343	0.000103
7986	7986	13701145	db2agntdp (SAMPLE) 0	1.410225	0.025900	0.003636	0.000074
2571	2571	8503329	db2ipccm 0	0.251349	0.083787	0.002551	0.000857
7729	7729	14168193	db2agntdp (SAMPLE) 0	1.717323	0.029477	0.000998	0.000038
7472	7472	11853991	db2agnta (SAMPLE) 0	1.860115	0.032926	0.000860	0.000012
3358	3358	2347127	db2loggr (SAMPLE) 0	0.151042	0.184726	0.000387	0.000458
515	515	13820091	db2aiothr 0	0.405538	0.312007	0.000189	0.000178
7215	7215	2539753	db2agntdp (SAMPLE) 0	1.165350	0.019466	0.000291	0.000008
6185	6185	2322517	db2wlmd (SAMPLE) 0	0.061674	0.034093	0.000169	0.000100
6442	6442	2756793	db2evmli (DB2DETAILEDLOCK) 0	0.072142	0.052436	0.000092	0.000063
4129	4129	15900799	db2glock (SAMPLE) 0	0.013239	0.000741	0.000064	0.000001
2	2	11739383	db2alarm 0	0.036904	0.028367	0.000009	0.000009
4386	4386	13361367	db2dlock (SAMPLE) 0	0.015653	0.001281	0.000014	0.000003
1029	1029	15040579	db2fcms 0	0.041929	0.016598	0.000010	0.000004
5414	5414	14471309	db2pfchr (SAMPLE) 0	0.000093	0.000002	0.000000	0.000000
258	258	13656311	db2sysc 0	8.369967	0.263539	0.000000	0.000000
5157	5157	7934145	db2pfchr (SAMPLE) 0	0.027598	0.000177	0.000000	0.000000
1543	1543	2670647	db2fcmr 0	0.004191	0.000079	0.000000	0.000000
1286	1286	8417339	db2extev 0	0.000312	0.000043	0.000000	0.000000
2314	2314	14360813	db2licc 0	0.000371	0.000051	0.000000	0.000000
5928	5928	3137537	db2taskd (SAMPLE) 0	0.004903	0.000572	0.000000	0.000000
3872	3872	2310357	db2lfr (SAMPLE) 0	0.000126	0.000007	0.000000	0.000000
4643	4643	11694287	db2pclnr (SAMPLE) 0	0.000094	0.000002	0.000000	0.000000
1800	1800	5800175	db2extev 0	0.001212	0.002137	0.000000	0.000000
772	772	7925817	db2thcln 0	0.000429	0.000072	0.000000	0.000000
2057	2057	6868993	db2pdbc 0	0.002423	0.001603	0.000000	0.000000
2828	2828	10866809	db2resync 0	0.016764	0.003098	0.000000	0.000000

To provide information only about the EDUs that are the top consumers of processor time and to reduce the amount of output returned, you can further include the **top** parameter option. In the following example, only the top five

EDUs are returned, across an interval of 5 seconds. Stack information is also returned, and can be found stored separately in the directory path specified by DUMPDIR, which defaults to **diagpath**.

```
$ db2pd -edus interval=5 top=5 stacks

Database Partition 0 -- Active -- Up 0 days 00:54:00 -- Date 06/04/2010 03:35:30

List of all EDUs for database partition 0

db2sysc PID: 1249522
db2wdog PID: 2068678

EDU ID TID Kernel TID EDU Name          USR      SYS    USR DELTA   SYS DELTA
=====
3358  3358  2347127  db2loggr (SAMPLE) 0  0.154906  0.189223  0.001087  0.001363
3615  3615  2887875  db2loggw (SAMPLE) 0  0.962744  0.419617  0.001779  0.000481
515   515   13820091  db2aiothr 0        0.408039  0.314045  0.000658  0.000543
258   258   13656311  db2sysc 0        8.371388  0.264812  0.000653  0.000474
6700  6700  11542589  db2agent (SAMPLE) 0  54.814420  0.783323  0.000455  0.000310

$ ls -ltr
total 552
drwxrwxr-t  2 vbmithun build          256 05-31 09:59 events/
drwxrwxr-t  2 vbmithun build          256 06-04 03:17 stmmlog/
-rw-r--r--  1 vbmithun build          46413 06-04 03:35 1249522.3358.000.stack.txt
-rw-r--r--  1 vbmithun build          22819 06-04 03:35 1249522.3615.000.stack.txt
-rw-r--r--  1 vbmithun build          20387 06-04 03:35 1249522.515.000.stack.txt
-rw-r--r--  1 vbmithun build          50426 06-04 03:35 1249522.258.000.stack.txt
-rw-r--r--  1 vbmithun build          314596 06-04 03:35 1249522.6700.000.stack.txt
-rw-r--r--  1 vbmithun build          94913 06-04 03:35 1249522.000.processObj.txt
```

Example 16: Displaying agent event metrics

The **db2pd** command supports returning event metrics for agents. If you need to determine whether an agent changed state during a specific period of time, use the event option together with the **-agents** parameter. The AGENT_STATE_LAST_UPDATE_TIME(Tick Value) column that is returned shows the last time that the event being processed by the agent was changed. Together with a previously obtained value for AGENT_STATE_LAST_UPDATE_TIME(Tick Value), you can determine whether an agent has moved on to a new task or whether it continues to process the same task over an extended period of time.

```
db2pd -agents event
Database Partition 0 -- Active -- Up 0 days 03:18:52 -- Date 06/27/2011 11:47:10

Agents:
Current agents: 12
Idle agents: 0
Active coord agents: 10
Active agents total: 10
Pooled coord agents: 2
Pooled agents total: 2

AGENT_STATE_LAST_UPDATE_TIME(Tick Value) EVENT_STATE EVENT_TYPE EVENT_OBJECT EVENT_OBJECT_NAME
2011-06-27-14.44.38.859785(...968075) IDLE      WAIT      REQUEST      n/a
```

Troubleshooting scripts

You may have internal tools or scripts that are based on the processes running in the database engine. These tools or scripts may no longer work because all agents, prefetchers, and page cleaners are now considered threads in a single, multi-threaded process.

Your internal tools and scripts will have to be modified to account for a threaded process. For example, you may have scripts that start the **ps** command to list the process names; and then perform tasks against certain agent processes. Your scripts must be rewritten.

The problem determination database command **db2pd** will have a new option **-edu** (short for “engine dispatchable unit”) to list all agent names along with their thread IDs. The **db2pd -stack** command continues to work with the threaded engine to dump individual EDU stacks or to dump all EDU stacks for the current node.

db2val command

The db2val command verifies the basic functions of a DB2 copy by checking the state of installation files, instance setup, and local database connections.

Validating your DB2 copy

Use the db2val command to determine whether your DB2 copy is functioning properly.

About this task

The **db2val** tool verifies the core function of a DB2 copy by validating installation files, instances, database creation, connections to that database, and the state of partitioned database environments. This validation can be helpful if you have manually deployed a DB2 copy on Linux and UNIX operating systems using tar.gz files. The **db2val** command can quickly ensure that all the configuration has been correctly done and ensure that the DB2 copy is what you expect it to be. You can specify instances and databases or you can run **db2val** against all of the instances. The **db2val** command can be found in the *DB2-install-path\bin* and *sql1ib/bin* directories.

Example

For example, to validate all the instances for the DB2 copy, run the following command:

```
db2val -a
```

For complete **db2val** command details and further example, refer to the “db2val - DB2 copy validation tool command” topic.

db2dart command

The **db2dart** command can be used to verify the architectural correctness of databases and the objects within them. It can also be used to display the contents of database control files in order to extract data from tables that might otherwise be inaccessible.

To display all of the possible options, issue the **db2dart** command without any parameters. Some options that require parameters, such as the table space ID, are prompted for if they are not explicitly specified on the command line.

By default, the **db2dart** utility will create a report file with the name *databaseName.RPT*. For single-partition database partition environments, the file is created in the current directory. For multiple-partition database partition

environments, the file is created under a subdirectory in the diagnostic directory. The subdirectory is called DART####, where #### is the database partition number.

In a DB2 pureScale environment, some metadata files (such as bufferpool configuration files) exist for each member and are validated or updated on a per-member basis (rather than per-database).

The **db2dart** utility accesses the data and metadata in a database by reading them directly from disk. Because of that, you should never run the tool against a database that still has active connections. If there are connections, the tool will not know about pages in the buffer pool or control structures in memory, for example, and might report false errors as a result. Similarly, if you run **db2dart** against a database that requires crash recovery or that has not completed rollforward recovery, similar inconsistencies might result due to the inconsistent nature of the data on disk.

Comparison of INSPECT and db2dart

The **INSPECT** command inspects a database for architectural integrity, checking the pages of the database for page consistency. The **INSPECT** command checks that the structures of table objects and structures of table spaces are valid. Cross object validation conducts an online index to data consistency check. The **db2dart** command examines databases for architectural correctness and reports any encountered errors.

The **INSPECT** command is similar to the **db2dart** command in that it allows you to check databases, table spaces, and tables. A significant difference between the two commands is that the database needs to be deactivated before you run **db2dart**, whereas **INSPECT** requires a database connection and can be run while there are simultaneous active connections to the database.

If you do not deactivate the database, **db2dart** will yield unreliable results.

The following tables list the differences between the tests that are performed by the **db2dart** and **INSPECT** commands.

Table 35. Feature comparison of db2dart and INSPECT for table spaces

Tests performed	db2dart	INSPECT
SMS table spaces		
Check table space files	YES	NO
Validate contents of internal page header fields	YES	YES
DMS table spaces		
Check for extent maps pointed at by more than one object	YES	NO
Check every extent map page for consistency bit errors	NO	YES
Check every space map page for consistency bit errors	NO	YES
Validate contents of internal page header fields	YES	YES

Table 35. Feature comparison of db2dart and INSPECT for table spaces (continued)

Tests performed	db2dart	INSPECT
Verify that extent maps agree with table space maps	YES	NO

Table 36. Feature comparison of db2dart and INSPECT for data objects

Tests performed	db2dart	INSPECT
Check data objects for consistency bit errors	YES	YES
Check the contents of special control rows	YES	NO
Check the length and position of variable length columns	YES	NO
Check the LONG VARCHAR, LONG VARGRAPHIC, and large object (LOB) descriptors in table rows	YES	NO
Check the summary total pages, used pages and free space percentage	NO	YES
Validate contents of internal page header fields	YES	YES
Verify each row record type and its length	YES	YES
Verify that rows are not overlapping	YES	YES

Table 37. Feature comparison of db2dart and INSPECT for index objects

Tests performed	db2dart	INSPECT
Check for consistency bit errors	YES	YES
Check the location and length of the index key and whether there is overlapping	YES	YES
Check the ordering of keys in the index	YES	NO
Determine the summary total pages and used pages	NO	YES
Validate contents of internal page header fields	YES	YES
Verify the uniqueness of unique keys	YES	NO
Check for the existence of the data row for a given index entry	NO	YES
Verify each key to a data value	NO	YES

Table 38. Feature comparison of db2dart and INSPECT for block map objects

Tests performed	db2dart	INSPECT
Check for consistency bit errors	YES	YES
Determine the summary total pages and used pages	NO	YES
Validate contents of internal page header fields	YES	YES

Table 39. Feature comparison of db2dart and INSPECT for long field and LOB objects

Tests performed	db2dart	INSPECT
Check the allocation structures	YES	YES
Determine the summary total pages and used pages (for LOB objects only)	NO	YES

In addition, the following actions can be performed using the **db2dart** command:

- Format and dump data pages
- Format and dump index pages
- Format data rows to delimited ASCII
- Mark an index invalid

The **INSPECT** command cannot be used to perform those actions.

Part 4. Performance and scalability

The performance improvement process is an iterative approach to monitoring and tuning aspects of performance. Depending on the results of this performance monitoring, you will adjust the configuration of the database server and make changes to the applications that use the database server. Scalability refers to the ability of a database to grow and continue to exhibit the same operating characteristics and response times.

Chapter 19. SQL and XQuery compiler

The SQL and XQuery compiler performs several steps to produce an access plan that can be executed.

The *query graph model* is an internal, in-memory database that represents the query as it is processed through these steps, which are shown in Figure 70 on page 554. Note that some steps occur only for queries that will run against a federated database.

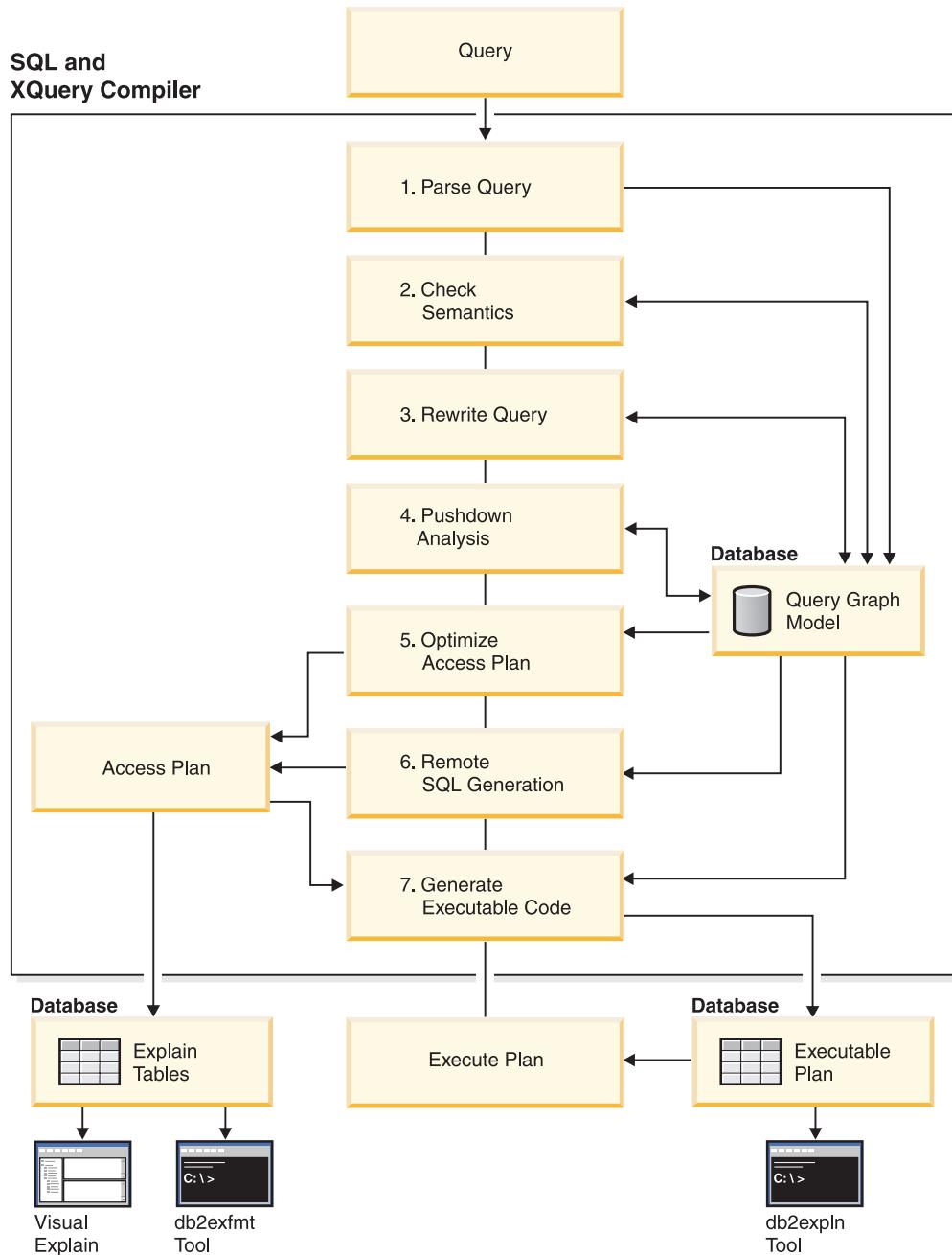


Figure 70. Steps performed by the SQL and XQuery compiler

1. Parse query

The SQL and XQuery compiler analyzes the query to validate the syntax. If any syntax errors are detected, the query compiler stops processing and returns an appropriate error to the application that submitted the query. When parsing is complete, an internal representation of the query is created and stored in the query graph model.

2. Check semantics

The compiler ensures that there are no inconsistencies among parts of the statement. For example, the compiler verifies that a column specified for the YEAR scalar function has been defined with a datetime data type.

The compiler also adds behavioral semantics to the query graph model, including the effects of referential constraints, table check constraints, triggers, and views. The query graph model contains all of the semantics for the query, including query blocks, subqueries, correlations, derived tables, expressions, data types, data type conversions, code page conversions, and distribution keys.

3. Rewrite query

The compiler uses the global semantics that are stored in the query graph model to transform the query into a form that can be optimized more easily. It then stores the result in the query graph model.

For example, the compiler might move a predicate, altering the level at which it is applied, thereby potentially improving query performance. This type of operation movement is called *general predicate pushdown*. In a partitioned database environment, the following query operations are more computationally intensive:

- Aggregation
- Redistribution of rows
- Correlated subqueries, which are subqueries that contain a reference to a column in a table that is outside of the subquery

For some queries in a partitioned database environment, decorrelation might occur as part of rewriting the query.

4. Pushdown analysis (federated databases only)

The major task in this step is to recommend to the optimizer whether an operation can be remotely evaluated or *pushed down* at a data source. This type of pushdown activity is specific to data source queries and represents an extension to general predicate pushdown operations.

5. Optimize access plan

Using the query graph model as input, the optimizer portion of the compiler generates many alternative execution plans for satisfying the query. To estimate the execution cost of each of these plans, the optimizer uses statistics for tables, indexes, columns and functions. It then chooses the plan with the smallest estimated execution cost. The optimizer uses the query graph model to analyze the query semantics and to obtain information about a wide variety of factors, including indexes, base tables, derived tables, subqueries, correlations, and recursion.

The optimizer can also consider another type of pushdown operation, *aggregation and sort*, which can improve performance by pushing the evaluation of these operations to the Data Management Services (DMS) component.

The optimizer also considers whether there are buffer pools of different sizes when determining page size selection. It considers whether the database is partitioned, or whether intraquery parallelism in a symmetric multiprocessor (SMP) environment is an option. This information is used by the optimizer to help select the best access plan for the query.

The output of this step is an access plan, and details about this access plan are captured in the explain tables. The information that is used to generate an access plan can be captured by an explain snapshot.

6. Remote SQL generation (federated databases only)

The final plan that is selected by the optimizer might consist of a set of steps that operate on a remote data source. The remote SQL generation step creates an efficient SQL statement for operations that are performed by each data source, based on the SQL dialect at that data source.

7. Generate executable code

In the final step, the compiler uses the access plan and the query graph model to create an executable access plan, or section, for the query. This code generation step uses information from the query graph model to avoid repetitive execution of expressions that need to be computed only once. This type of optimization is possible for code page conversions and when host variables are used.

To enable query optimization or reoptimization of static or dynamic SQL or XQuery statements that have host variables, special registers, or parameter markers, bind the package with the REOPT bind option. The access path for a statement belonging to such a package, and containing host variables, special registers, or parameter markers, will be optimized using the values of these variables rather than default estimates that are chosen by the compiler. This optimization takes place at query execution time when the values are available.

Information about access plans for static SQL and XQuery statements is stored in the system catalog tables. When a package is executed, the database manager uses the information that is stored in the system catalog to determine how to access the data and provide results for the query. This information is used by the **db2expln** tool.

Note: Execute the **RUNSTATS** command at appropriate intervals against tables that change often. The optimizer needs up-to-date statistical information about tables and their data to create the most efficient access plans. Rebind your application to take advantage of updated statistics. If **RUNSTATS** is not executed, or the optimizer assumes that this command was executed against empty or nearly empty tables, it might use default values or attempt to derive certain statistics based on the number of file pages that are used to store the table on disk. See also “Automatic statistics collection”.

Query rewriting methods and examples

During the query rewrite stage, the query compiler transforms SQL and XQuery statements into forms that can be optimized more easily; this can improve the possible access plans. Rewriting queries is particularly important for very complex queries, including those queries that have many subqueries or many joins. Query generator tools often create these types of very complex queries.

To influence the number of query rewrite rules that are applied to an SQL or XQuery statement, change the optimization class. To see some of the results of the query rewrite process, use the explain facility.

Queries might be rewritten in any one of the following ways:

- Operation merging

To construct a query so that it has the fewest number of operations, especially SELECT operations, the SQL and XQuery compiler rewrites queries to merge query operations. The following examples illustrate some of the operations that can be merged:

- Example - View merges

A SELECT statement that uses views can restrict the join order of the table and can also introduce redundant joining of tables. If the views are merged during query rewrite, these restrictions can be lifted.

- Example - Subquery to join transforms

If a SELECT statement contains a subquery, selection of order processing of the tables might be restricted.

- Example - Redundant join elimination

During query rewrite, redundant joins can be removed to simplify the SELECT statement.

- Example - Shared aggregation

When a query uses different functions, rewriting can reduce the number of calculations that need to be done.

- Operation movement

To construct a query with the minimum number of operations and predicates, the compiler rewrites the query to move query operations. The following examples illustrate some of the operations that can be moved:

- Example - DISTINCT elimination

During query rewrite, the optimizer can move the point at which the DISTINCT operation is performed, to reduce the cost of this operation. In some cases, the DISTINCT operation can be removed completely.

- Example - General predicate pushdown

During query rewrite, the optimizer can change the order in which predicates are applied, so that more selective predicates are applied to the query as early as possible.

- Example - Decorrelation

In a partitioned database environment, the movement of result sets among database partitions is costly. Reducing the size of what must be broadcast to other database partitions, or reducing the number of broadcasts, or both, is an objective of the query rewriting process.

- Predicate translation

The SQL and XQuery compiler rewrites queries to translate existing predicates into more optimal forms. The following examples illustrate some of the predicates that might be translated:

- Example - Addition of implied predicates

During query rewrite, predicates can be added to a query to enable the optimizer to consider additional table joins when selecting the best access plan for the query.

- Example - OR to IN transformations

During query rewrite, an OR predicate can be translated into an IN predicate for a more efficient access plan. The SQL and XQuery compiler can also translate an IN predicate into an OR predicate if this transformation would create a more efficient access plan.

Compiler rewrite example: Operation merging

A SELECT statement that uses views can restrict the join order of the table and can also introduce redundant joining of tables. If the views are merged during query rewrite, these restrictions can be lifted.

Suppose you have access to the following two views that are based on the EMPLOYEE table: one showing employees that have a high level of education and the other showing employees that earn more than \$35,000 per year:

```
create view emp_education (empno, firstnme, lastname, edlevel) as
  select empno, firstnme, lastname, edlevel
    from employee
   where edlevel > 17
```

```

create view emp_salaries (empno, firstname, lastname, salary) as
  select empno, firstnme, lastname, salary
    from employee
   where salary > 35000

```

The following user-written query lists those employees who have a high level of education and who earn more than \$35,000 per year:

```

select e1.empno, e1.firstname, e1.lastname, e1.edlevel, e2.salary
  from emp_education e1, emp_salaries e2
 where e1.empno = e2.empno

```

During query rewrite, these two views could be merged to create the following query:

```

select e1.empno, e1.firstname, e1.lastname, e1.edlevel, e2.salary
  from employee e1, employee e2
 where
   e1.empno = e2.empno and
   e1.edlevel > 17 and
   e2.salary > 35000

```

By merging the SELECT statements from the two views with the user-written SELECT statement, the optimizer can consider more choices when selecting an access plan. In addition, if the two views that have been merged use the same base table, additional rewriting might be performed.

Example - Subquery to join transformations

The SQL and XQuery compiler will take a query containing a subquery, such as:

```

select empno, firstname, lastname, phoneno
  from employee
 where workdept in
 (select deptno
   from department
  where deptname = 'OPERATIONS')

```

and convert it to a join query of the form:

```

select distinct empno, firstname, lastname, phoneno
  from employee emp, department dept
 where
   emp.workdept = dept.deptno and
   dept.deptname = 'OPERATIONS'

```

A join is generally much more efficient to execute than a subquery.

Example - Redundant join elimination

Queries sometimes have unnecessary joins.

```

select e1.empno, e1.firstname, e1.lastname, e1.edlevel, e2.salary
  from employee e1,
       employee e2
 where e1.empno = e2.empno
   and e1.edlevel > 17
   and e2.salary > 35000

```

The SQL and XQuery compiler can eliminate the join and simplify the query to:

```

select empno, firstname, lastname, edlevel, salary
  from employee
 where
   edlevel > 17 and
   salary > 35000

```

The following example assumes that a referential constraint exists between the EMPLOYEE and DEPARTMENT tables on the department number. First, a view is created.

```
create view peplview as
  select firstname, lastname, salary, deptno, deptname, mgrno
    from employee e department d
      where e.workdept = d.deptno
```

Then a query such as the following:

```
select lastname, salary
  from peplview
```

becomes:

```
select lastname, salary
  from employee
  where workdept not null
```

Note that in this situation, even if you know that the query can be rewritten, you might not be able to do so because you do not have access to the underlying tables. You might only have access to the view (shown previously). Therefore, this type of optimization has to be performed by the database manager.

Redundancy in referential integrity joins likely occurs when:

- Views are defined with joins
- Queries are automatically generated

Example - Shared aggregation

Using multiple functions within a query can generate several calculations that take time. Reducing the number of required calculations improves the plan. The compiler takes a query that uses multiple functions, such as the following:

```
select sum(salary+bonus+comm) as osum,
      avg(salary+bonus+comm) as oavg,
      count(*) as ocount
    from employee
```

and transforms it:

```
select osum, osum/ocount ocount
  from (
    select sum(salary+bonus+comm) as osum,
           count(*) as ocount
      from employee
  ) as shared_agg
```

This rewrite halves the required number of sums and counts.

Compiler rewrite example: Operation movement

During query rewrite, the optimizer can move the point at which the DISTINCT operation is performed, to reduce the cost of this operation. In some cases, the DISTINCT operation can be removed completely.

For example, if the EMPNO column of the EMPLOYEE table were defined as the primary key, the following query:

```
select distinct empno, firstname, lastname
  from employee
```

could be rewritten by removing the DISTINCT clause:

```
select empno, firstnme, lastname  
  from employee
```

In this example, because the primary key is being selected, the compiler knows that each returned row is unique. In this case, the DISTINCT keyword is redundant. If the query is not rewritten, the optimizer must build a plan with necessary processing (such as a sort) to ensure that the column values are unique.

Example - General predicate pushdown

Altering the level at which a predicate is normally applied can result in improved performance. For example, the following view provides a list of all employees in department D11:

```
create view d11_employee  
(empno, firstnme, lastname, phoneno, salary, bonus, comm) as  
  select empno, firstnme, lastname, phoneno, salary, bonus, comm  
    from employee  
   where workdept = 'D11'
```

The following query against this view is not as efficient as it could be:

```
select firstnme, phoneno  
  from d11_employee  
 where lastname = 'BROWN'
```

During query rewrite, the compiler pushes the `lastname = 'BROWN'` predicate down into the `D11_EMPLOYEE` view. This allows the predicate to be applied sooner and potentially more efficiently. The actual query that could be executed in this example is as follows:

```
select firstnme, phoneno  
  from employee  
 where  
   lastname = 'BROWN' and  
   workdept = 'D11'
```

Predicate pushdown is not limited to views. Other situations in which predicates can be pushed down include UNION, GROUP BY, and derived tables (nested table expressions or common table expressions).

Example - Decorrelation

In a partitioned database environment, the compiler can rewrite the following query, which is designed to find all of the employees who are working on programming projects and who are underpaid.

```
select p.projno, e.empno, e.lastname, e.firstname,  
      e.salary+e.bonus+e.comm as compensation  
  from employee e, project p  
 where  
   p.empno = e.empno and  
   p.projname like '%PROGRAMMING%' and  
   e.salary+e.bonus+e.comm <  
     (select avg(e1.salary+e1.bonus+e1.comm)  
      from employee e1, project p1  
     where  
      p1.projname like '%PROGRAMMING%' and  
      p1.projno = p.projno and  
      e1.empno = p1.empno)
```

Because this query is correlated, and because both PROJECT and EMPLOYEE are unlikely to be partitioned on PROJNO, the broadcasting of each project to each database partition is possible. In addition, the subquery would have to be evaluated many times.

The compiler can rewrite the query as follows:

- Determine the distinct list of employees working on programming projects and call it DIST_PROJS. It must be distinct to ensure that aggregation is done only once for each project:

```
with dist_projs(projno, empno) as
  (select distinct projno, empno
   from project p1
   where p1.projname like '%PROGRAMMING%')
```

- Join DIST_PROJS with the EMPLOYEE table to get the average compensation per project, AVG_PER_PROJ:

```
avg_per_proj(projno, avg_comp) as
  (select p2.projno, avg(e1.salary+e1.bonus+e1.comm)
   from employee e1, dist_projs p2
   where e1.empno = p2.empno
   group by p2.projno)
```

- The rewritten query is:

```
select p.projno, e.empno, e.lastname, e.firstname,
       e.salary+e.bonus+e.comm as compensation
    from project p, employee e, avg_per_prog a
   where
     p.empno = e.empno and
     p.projname like '%PROGRAMMING%' and
     p.projno = a.projno and
     e.salary+e.bonus+e.comm < a.avg_comp
```

This query computes the avg_comp per project (avg_per_proj). The result can then be broadcast to all database partitions that contain the EMPLOYEE table.

Compiler rewrite example: Operation movement - Predicate pushdown for combined SQL/XQuery statements

One fundamental technique for the optimization of relational SQL queries is to move predicates in the WHERE clause of an enclosing query block into an enclosed lower query block (for example, a view), thereby enabling early data filtering and potentially better index usage.

This is even more important in partitioned database environments, because early filtering potentially reduces the amount of data that must be shipped between database partitions.

Similar techniques can be used to move predicates or XPath filters inside of an XQuery. The basic strategy is always to move filtering expressions as close to the data source as possible. This optimization technique is called *predicate pushdown* in SQL and *extraction pushdown* (for filters and XPath extractions) in XQuery.

Because the data models employed by SQL and XQuery are different, you must move predicates, filters, or extractions across the boundary between the two languages. Data mapping and casting rules have to be considered when transforming an SQL predicate into a semantically equivalent filter and pushing it down into the XPath extraction. The following examples address the pushdown of relation predicates into XQuery query blocks.

Consider the following two XML documents containing customer information:

Document 1

```
<customer>
  <name>John</name>
  <lastname>Doe</lastname>
  <date_of_birth>
    1976-10-10
  </date_of_birth>
  <address>
    <zip>95141.0</zip>
  </address>
  <volume>80000.0</volume>
</customer>
<customer>
  <name>Jane</name>
  <lastname>Doe</lastname>
  <date_of_birth>
    1975-01-01
  </date_of_birth>
  <address>
    <zip>95141.4</zip>
  </address>
  <volume>50000.00</volume>
</customer>
```

Document 2

```
<customer>
  <name>Michael</name>
  <lastname>Miller </lastname>
  <date_of_birth>
    1975-01-01
  </date_of_birth>
  <address>
    <zip>95142.0</zip>
  </address>
  <volume>100000.00</volume>
</customer>
<customer>
  <name>Michaela</name>
  <lastname>Miller</lastname>
  <date_of_birth>
    1980-12-23
  </date_of_birth>
  <address>
    <zip>95140.5</zip>
  </address>
  <volume>100000</volume>
</customer>
```

Example - Pushing INTEGER predicates

Consider the following query:

```
select temp.name, temp.zip
  from xmltable('db2-fn:xmlcolumn("T.XMLDOC")'
    columns name varchar(20) path 'customer/name',
    zip integer path 'customer/zip'
  ) as temp
 where zip = 95141
```

To use possible indexes on T.XMLDOC and to filter out records that are not needed early on, the `zip = 95141` predicate will be internally converted into the following equivalent XPATH filtering expression:

```
T.XMLCOL/customer/zip[. >= 95141.0 and . < 95142.0]
```

Because schema information for XML fragments is not used by the compiler, it cannot be assumed that ZIP contains integers only. It is possible that there are other numeric values with a fractional part and a corresponding double XML index on this specific XPath extraction. The XML2SQL cast would handle this transformation by truncating the fractional part before casting the value to INTEGER. This behavior must be reflected in the pushdown procedure, and the predicate must be changed to remain semantically correct.

Example - Pushing DECIMAL(x,y) predicates

Consider the following query:

```
select temp.name, temp.volume
  from xmltable('db2-fn:xmlcolumn("T.XMLDOC")'
    columns name varchar(20) path 'customer/name',
    volume decimal(10,2) path 'customer/volume'
  ) as temp
 where volume = 100000.00
```

To use possible DECIMAL or DOUBLE indexes on T.XMLDOC and to filter out records that are not needed early on, similar to the handling for the INTEGER

type, the `volume = 100000.00` predicate will be internally converted into the following XPATH range filtering expression:

```
T.XMLCOL/customer/volume[.>=100000.00 and .<100000.01]
```

Example - Pushing VARCHAR(*n*) predicates

Consider the following query:

```
select temp.name, temp.lastname
  from xmstable('db2-fn:xmlcolumn("T.XMLDOC")'
    columns name varchar(20) path 'customer/name',
    lastname varchar(20) path 'customer/lastname'
  ) as temp
 where lastname = 'Miller'
```

To use possible indexes on T.XMLDOC and to filter out records that are not needed early on, the `lastname = 'Miller'` predicate will be internally converted into an equivalent XPATH filtering expression. A high-level representation of this expression is:

```
T.XMLCOL/customer/lastname[. >= rtrim("Miller") and . <
RangeUpperBound("Miller", 20)]
```

Trailing blanks are treated differently in SQL than in XPath or XQuery. The original SQL predicate will not distinguish between the two customers whose last name is "Miller", even if one of them (Michael) has a trailing blank. Consequently, both customers are returned, which would not be the case if an unchanged predicate were pushed down.

The solution is to transform the predicate into a range filter.

- The first boundary is created by truncating all trailing blanks from the comparison value, using the RTRIM() function.
- The second boundary is created by looking up all possible strings that are greater than or equal to "Miller", so that all strings that begin with "Miller" are located. Therefore, the original string is replaced with an upperbound string that represents this second boundary.

Compiler rewrite example: Predicate translation

During query rewrite, predicates can be added to a query to enable the optimizer to consider additional table joins when selecting the best access plan for the query.

The following query returns a list of the managers whose departments report to department E01, and the projects for which those managers are responsible:

```
select dept.deptname dept.mgrno, emp.lastname, proj.projname
  from department dept, employee emp, project proj
 where
  dept.admrdept = 'E01' and
  dept.mgrno = emp.empno and
  emp.empno = proj.respemp
```

This query can be rewritten with the following implied predicate, known as a *predicate for transitive closure*:

```
dept.mgrno = proj.respemp
```

The optimizer can now consider additional joins when it tries to select the best access plan for the query.

During the query rewrite stage, additional local predicates are derived on the basis of the transitivity that is implied by equality predicates. For example, the following query returns the names of the departments whose department number is greater than E00, and the employees who work in those departments.

```
select empno, lastname, firstname, deptno, deptname
  from employee emp, department dept
 where
   emp.workdept = dept.deptno and
   dept.deptno > 'E00'
```

This query can be rewritten with the following implied predicate:

```
emp.workdept > 'E00'
```

This rewrite reduces the number of rows that need to be joined.

Example - OR to IN transformations

Suppose that an OR clause connects two or more simple equality predicates on the same column, as in the following example:

```
select *
  from employee
 where
   deptno = 'D11' or
   deptno = 'D21' or
   deptno = 'E21'
```

If there is no index on the DEPTNO column, using an IN predicate in place of OR causes the query to be processed more efficiently:

```
select *
  from employee
 where deptno in ('D11', 'D21', 'E21')
```

In some cases, the database manager might convert an IN predicate into a set of OR clauses so that index ORing can be performed.

Access plan optimization

Access plans can be optimized in an attempt to improve query performance. The degree of improvement depends on the type of optimization chosen. Optimizing access plans is one of the best ways to ensure that the query compiler behaves the way you expect and design it to.

Optimization classes

When you compile an SQL or XQuery statement, you can specify an optimization class that determines how the optimizer chooses the most efficient access plan for that statement.

The optimization classes differ in the number and type of optimization strategies that are considered during the compilation of a query. Although you can specify optimization techniques individually to improve runtime performance for the query, the more optimization techniques that you specify, the more time and system resources query compilation will require.

You can specify one of the following optimization classes when you compile an SQL or XQuery statement.

- 0** This class directs the optimizer to use minimal optimization when generating an access plan, and has the following characteristics:
- Frequent-value statistics are not considered by the optimizer.
 - Only basic query rewrite rules are applied.
 - Greedy join enumeration is used.
 - Only nested loop join and index scan access methods are enabled.
 - List prefetch is not used in generated access methods.
 - The star-join strategy is not considered.
- This class should only be used in circumstances that require the lowest possible query compilation overhead. Query optimization class 0 is appropriate for an application that consists entirely of very simple dynamic SQL or XQuery statements that access well-indexed tables.
- 1** This optimization class has the following characteristics:
- Frequent-value statistics are not considered by the optimizer.
 - Only a subset of query rewrite rules are applied.
 - Greedy join enumeration is used.
 - List prefetch is not used in generated access methods.
- Optimization class 1 is similar to class 0, except that merge scan joins and table scans are also available.
- 2** This class directs the optimizer to use a degree of optimization that is significantly higher than class 1, while keeping compilation costs for complex queries significantly lower than class 3 or higher. This optimization class has the following characteristics:
- All available statistics, including frequent-value and quantile statistics, are used.
 - All query rewrite rules (including materialized query table routing) are applied, except computationally intensive rules that are applicable only in very rare cases.
 - Greedy join enumeration is used.
 - A wide range of access methods is considered, including list prefetch and materialized query table routing.
 - The star-join strategy is considered, if applicable.
- Optimization class 2 is similar to class 5, except that it uses greedy join enumeration instead of dynamic programming join enumeration. This class has the most optimization of all classes that use the greedy join enumeration algorithm, which considers fewer alternatives for complex queries, and therefore consumes less compilation time than class 3 or higher. Class 2 is recommended for very complex queries in a decision support or online analytic processing (OLAP) environment. In such environments, a specific query is not likely to be repeated in exactly the same way, so that an access plan is unlikely to remain in the cache until the next occurrence of the query.
- 3** This class represents a moderate amount of optimization, and comes closest to matching the query optimization characteristics of DB2 for z/OS. This optimization class has the following characteristics:
- Frequent-value statistics are used, if available.
 - Most query rewrite rules are applied, including subquery-to-join transformations.

- Dynamic programming join enumeration is used, with:
 - Limited use of composite inner tables
 - Limited use of Cartesian products for star schemas involving lookup tables
- A wide range of access methods is considered, including list prefetch, index ANDing, and star joins.

This class is suitable for a broad range of applications, and improves access plans for queries with four or more joins.

- 5** This class directs the optimizer to use a significant amount of optimization to generate an access plan, and has the following characteristics:
- All available statistics, including frequent-value and quantile statistics, are used.
 - All query rewrite rules (including materialized query table routing) are applied, except computationally intensive rules that are applicable only in very rare cases.
 - Dynamic programming join enumeration is used, with:
 - Limited use of composite inner tables
 - Limited use of Cartesian products for star schemas involving lookup tables
 - A wide range of access methods is considered, including list prefetch, index ANDing, and materialized query table routing.

Optimization class 5 (the default) is an excellent choice for a mixed environment with both transaction processing and complex queries. This optimization class is designed to apply the most valuable query transformations and other query optimization techniques in an efficient manner.

If the optimizer detects that additional resources and processing time for complex dynamic SQL or XQuery statements are not warranted, optimization is reduced. The extent of the reduction depends on the machine size and the number of predicates. When the optimizer reduces the amount of query optimization, it continues to apply all of the query rewrite rules that would normally be applied. However, it uses greedy join enumeration and it reduces the number of access plan combinations that are considered.

- 7** This class directs the optimizer to use a significant amount of optimization to generate an access plan. It is similar to optimization class 5, except that in this case, the optimizer never considers reducing the amount of query optimization for complex dynamic SQL or XQuery statements.
- 9** This class directs the optimizer to use all available optimization techniques. These include:
- All available statistics
 - All query rewrite rules
 - All possibilities for join enumeration, including Cartesian products and unlimited composite inners
 - All access methods

This class increases the number of possible access plans that are considered by the optimizer. You might use this class to determine whether more comprehensive optimization would generate a better access plan for very

complex or very long-running queries that use large tables. Use explain and performance measurements to verify that a better plan has actually been found.

Choosing an optimization class

Setting the optimization class can provide some of the advantages of explicitly specifying optimization techniques.

This is true, particularly when:

- Managing very small databases or very simple dynamic queries
- Accommodating memory limitations on your database server at compile time
- Reducing query compilation time; for example, during statement preparation

Most statements can be adequately optimized with a reasonable amount of resource by using the default optimization class 5. Query compilation time and resource consumption are primarily influenced by the complexity of a query; in particular, by the number of joins and subqueries. However, compilation time and resource consumption are also affected by the amount of optimization that is performed.

Query optimization classes 1, 2, 3, 5, and 7 are all suitable for general use. Consider class 0 only if you require further reductions in query compilation time, and the SQL and XQuery statements are very simple.

Tip: To analyze a long-running query, run the query with **db2batch** to determine how much time is spent compiling and executing the query. If compilation time is excessive, reduce the optimization class. If execution time is a problem, consider a higher optimization class.

When you select an optimization class, consider the following general guidelines:

- Start by using the default query optimization class 5.
- When choosing a class other than the default, try class 1, 2, or 3 first. Classes 0, 1, and 2 use the greedy join enumeration algorithm.
- Use optimization class 1 or 2 if you have many tables with many join predicates on the same column, and if compilation time is a concern.
- Use a low optimization class (0 or 1) for queries that have very short run times of less than one second. Such queries tend to:
 - Access a single table or only a few tables
 - Fetch a single row or only a few rows
 - Use fully qualified and unique indexes
 - Be involved in online transaction processing (OLTP)
- Use a higher optimization class (3, 5, or 7) for queries that have longer run times of more than 30 seconds.
- Class 3 or higher uses the dynamic programming join enumeration algorithm, which considers many more alternative plans, and might incur significantly more compilation time than classes 0, 1, or 2, especially as the number of tables increases.
- Use optimization class 9 only if you have extraordinary optimization requirements for a query.

Complex queries might require different amounts of optimization to select the best access plan. Consider using higher optimization classes for queries that have:

- Access to large tables
- A large number of views
- A large number of predicates
- Many subqueries
- Many joins
- Many set operators, such as UNION or INTERSECT
- Many qualifying rows
- GROUP BY and HAVING operations
- Nested table expressions

Decision support queries or month-end reporting queries against fully normalized databases are good examples of complex queries for which at least the default query optimization class should be used.

Use higher query optimization classes for SQL and XQuery statements that were produced by a query generator. Many query generators create inefficient queries. Poorly written queries require additional optimization to select a good access plan. Using query optimization class 2 or higher can improve such queries.

For SAP applications, always use optimization class 5. This optimization class enables many DB2 features optimized for SAP, such as setting the **DB2_REDUCED_OPTIMIZATION** registry variable.

In a federated database, the optimization class does not apply to the remote optimizer.

Setting the optimization class

When you specify an optimization level, consider whether a query uses static or dynamic SQL and XQuery statements, and whether the same dynamic query is repeatedly executed.

About this task

For static SQL and XQuery statements, the query compilation time and resources are expended only once, and the resulting plan can be used many times. In general, static SQL and XQuery statements should always use the default query optimization class (5). Because dynamic statements are bound and executed at run time, consider whether the overhead of additional optimization for dynamic statements improves overall performance. However, if the same dynamic SQL or XQuery statement is executed repeatedly, the selected access plan is cached. Such statements can use the same optimization levels as static SQL and XQuery statements.

If you are not sure whether a query might benefit from additional optimization, or you are concerned about compilation time and resource consumption, consider benchmark testing.

Procedure

To specify a query optimization class:

1. Analyze performance factors.

- For a dynamic query statement, tests should compare the average run time for the statement. Use the following formula to estimate the average run time:

$$\frac{\text{compilation time} + \text{sum of execution times for all iterations}}{\text{number of iterations}}$$

The number of iterations represents the number of times that you expect the statement might be executed each time that it is compiled.

Note: After initial compilation, dynamic SQL and XQuery statements are recompiled whenever a change to the environment requires it. If the environment does not change after a statement is cached, subsequent PREPARE statements reuse the cached statement.

- For static SQL and XQuery statements, compare the statement run times. Although you might also be interested in the compilation time of static SQL and XQuery statements, the total compilation and execution time for a static statement is difficult to assess in any meaningful context. Comparing the total times does not recognize the fact that a static statement can be executed many times whenever it is bound, and that such a statement is generally not bound during run time.
2. Specify the optimization class.
- Dynamic SQL and XQuery statements use the optimization class that is specified by the CURRENT QUERY OPTIMIZATION special register. For example, the following statement sets the optimization class to 1:

```
SET CURRENT QUERY OPTIMIZATION = 1
```

To ensure that a dynamic SQL or XQuery statement always uses the same optimization class, include a SET statement in the application program.

If the CURRENT QUERY OPTIMIZATION special register has not been set, dynamic statements are bound using the default query optimization class. The default value for both dynamic and static queries is determined by the value of the **dft_queryopt** database configuration parameter, whose default value is 5. The default values for the bind option and the special register are also read from the **dft_queryopt** database configuration parameter.

- Static SQL and XQuery statements use the optimization class that is specified on the **PREP** and **BIND** commands. The QUERYOPT column in the SYSCAT.PACKAGES catalog view records the optimization class that is used to bind a package. If the package is rebound, either implicitly or by using the **REBIND PACKAGE** command, this same optimization class is used for static statements. To change the optimization class for such static SQL and XQuery statements, use the **BIND** command. If you do not specify the optimization class, the data server uses the default optimization class, as specified by the **dft_queryopt** database configuration parameter.

Optimization profiles and guidelines

An optimization profile is an XML document that can contain optimization guidelines for one or more SQL statements. The correspondence between each SQL statement and its associated optimization guidelines is established using the SQL text and other information that is needed to unambiguously identify an SQL statement.

The DB2 optimizer is one of the most sophisticated cost-based optimizers in the industry. However, in rare cases the optimizer might select a less than optimal execution plan. As a DBA familiar with the database, you can use utilities such as

db2advis, **runstats**, and **db2expln**, as well as the optimization class setting to help you tune the optimizer for better database performance. If you do not receive expected results after all tuning options have been exhausted, you can provide explicit optimization guidelines to the DB2 optimizer.

For example, suppose that even after you had updated the database statistics and performed all other tuning steps, the optimizer still did not choose the I_SUPPKEY index to access the SUPPLIERS table in the following subquery:

```
SELECT S.S_NAME, S.S_ADDRESS, S.S_PHONE, S.S_COMMENT
FROM PARTS P, SUPPLIERS S, PARTSUPP PS
WHERE P.PARTKEY = PS.PS_PARTKEY
AND S.S_SUPPKEY = PS.PS_SUPPKEY
AND P.P_SIZE = 39
AND P.P_TYPE = 'BRASS'
AND S.S_NATION = 'MOROCCO'
AND S.S_NATION IN ('MOROCCO', 'SPAIN')
AND PS.PS_SUPPLYCOST = (SELECT MIN(PS1.PS_SUPPLYCOST)
FROM PARTSUPP PS1, SUPPLIERS S1
WHERE P.PARTKEY = PS1.PS_PARTKEY
AND S1.S_SUPPKEY = PS1.PS_SUPPKEY
AND S1.S_NATION = S.S_NATION)
```

In this case, an explicit optimization guideline can be used to influence the optimizer. For example:

```
<OPTGUIDELINES><IXSCAN TABLE="S" INDEX="I_SUPPKEY"/></OPTGUIDELINES>
```

Optimization guidelines are specified using a simple XML specification. Each element within the OPTGUIDELINES element is interpreted as an optimization guideline by the DB2 optimizer. There is one optimization guideline element in this example. The IXSCAN element requests that the optimizer use index access. The TABLE attribute of the IXSCAN element indicates the target table reference (using the exposed name of the table reference) and the INDEX attribute specifies the index.

The following example is based on the previous query, and shows how an optimization guideline can be passed to the DB2 optimizer using an optimization profile.

```
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.1.0.0">
<STMTPROFILE ID="Guidelines for SAMP Q9">
<STMTKEY SCHEMA="SAMP">
  SELECT S.S_NAME, S.S_ADDRESS, S.S_PHONE, S.S_COMMENT
  FROM PARTS P, SUPPLIERS S, PARTSUPP PS
  WHERE P.PARTKEY = PS.PS_PARTKEY
  AND S.S_SUPPKEY = PS.PS_SUPPKEY
  AND P.P_SIZE = 39
  AND P.P_TYPE = 'BRASS'
  AND S.S_NATION = 'MOROCCO'
  AND S.S_NATION IN ('MOROCCO', 'SPAIN')
  AND PS.PS_SUPPLYCOST = (SELECT MIN(PS1.PS_SUPPLYCOST)
  FROM PARTSUPP PS1, SUPPLIERS S1
  WHERE P.PARTKEY = PS1.PS_PARTKEY
  AND S1.S_SUPPKEY = PS1.PS_SUPPKEY
  AND S1.S_NATION = S.S_NATION)
</STMTKEY>
<OPTGUIDELINES><IXSCAN TABLE="S" INDEX="I_SUPPKEY"/></OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

Each STMTPROFILE element provides a set of optimization guidelines for one application statement. The targeted statement is identified by the STMTKEY subelement. The optimization profile is then given a schema-qualified name and inserted into the database. The optimization profile is put into effect for the statement by specifying this name on the **BIND** or **PRECOMPILE** command.

Optimization profiles allow optimization guidelines to be provided to the optimizer without application or database configuration changes. You simply compose the simple XML document, insert it into the database, and specify the

name of the optimization profile on the **BIND** or **PRECOMPILE** command. The optimizer automatically matches optimization guidelines to the appropriate statement.

Optimization guidelines do not need to be comprehensive, but should be targeted to a desired execution plan. The DB2 optimizer still considers other possible access plans using the existing cost-based methods. Optimization guidelines targeting specific table references cannot override general optimization settings. For example, an optimization guideline specifying the merge join between tables A and B is not valid at optimization class 0.

The optimizer ignores invalid or inapplicable optimization guidelines. If any optimization guidelines are ignored, an execution plan is created and SQL0437W with reason code 13 is returned. You can then use the EXPLAIN statement to get detailed diagnostic information regarding optimization guidelines processing.

Collecting accurate catalog statistics, including advanced statistics features

Accurate database statistics are critical for query optimization. Perform **RUNSTATS** command operations regularly on any tables that are critical to query performance.

You might also want to collect statistics on system catalog tables, if an application queries these tables directly and if there is significant catalog update activity, such as that resulting from the execution of data definition language (DDL) statements. Automatic statistics collection can be enabled to allow the DB2 data server to automatically perform a **RUNSTATS** command operation. Real-time statistics collection can be enabled to allow the DB2 data server to provide even more timely statistics by collecting them immediately before queries are optimized.

If you are collecting statistics manually by using the **RUNSTATS** command, use the following options at a minimum:

```
RUNSTATS ON TABLE DB2USER.DAILY_SALES  
WITH DISTRIBUTION AND SAMPLED DETAILED INDEXES ALL
```

Distribution statistics make the optimizer aware of data skew. Detailed index statistics provide more details about the I/O required to fetch data pages when the table is accessed by using a particular index. Collecting detailed index statistics uses considerable processing time and memory for large tables. The **SAMPLED** option provides detailed index statistics with nearly the same accuracy but requires a fraction of the CPU and memory. These options are used by automatic statistics collection when a statistical profile is not provided for a table.

To improve query performance, consider collecting more advanced statistics, such as column group statistics or LIKE statistics, or creating statistical views.

Statistical views are helpful when gathering statistics for complex relationships. Gathering statistics for statistical views can be automated through the automatic statistics collection feature in DB2 for Linux, UNIX, and Windows. Enabling or disabling the automatic statistic collection of statistical views is done by using the **auto_stats_views** database configuration parameter. To enable this function, issue the following command:

```
update db cfg for dbname using auto_stats_views on
```

To disable this feature, issue the following command:

```
update db cfg for dbname using auto_stats_views off
```

This database configuration parameter is off by default. The command that is issued to automatically collect statistics on statistical views is equivalent to the following command:

```
runstats on view view_name with distribution
```

Collecting statistics for a large table or statistical view can be time consuming. Statistics of the same quality can often be collected by considering just a small sample of the overall data. Consider enabling automatic sampling for all background statistic collections; this may reduce the statistic collection time. To enable this function, issue the following command:

```
update db cfg for dbname using auto_sampling on
```

Collected statistics are not always exact. In addition to providing more efficient data access, an index can help provide more accurate statistics for columns which are often used in query predicates. When statistics are collected for a table and its indexes, index objects can provide accurate statistics for the leading index columns.

Configuration parameters that affect query optimization

Several configuration parameters affect the access plan chosen by the SQL or XQuery compiler. Many of these are appropriate to a single-partition database environment and some are only appropriate to a partitioned database environment.

Assuming a homogeneous partitioned database environment, where the hardware is the same, the values used for each parameter should be the same on all database partitions.

Note: When you change a configuration parameter dynamically, the optimizer might not read the changed parameter values immediately because of older access plans in the package cache. To reset the package cache, execute the **FLUSH PACKAGE CACHE** command.

In a federated system, if the majority of your queries access nicknames, evaluate the types of queries that you send before you change your environment. For example, in a federated database, the buffer pool does not cache pages from data sources, which are the DBMSs and data within the federated system. For this reason, increasing the size of the buffer does not guarantee that the optimizer considers additional access-plan alternatives when it chooses an access plan for queries that contain nicknames. However, the optimizer might decide that local materialization of data source tables is the least-cost route or a necessary step for a sort operation. In that case, increasing the resources available might improve performance.

The following configuration parameters or factors affect the access plan chosen by the SQL or XQuery compiler:

- The size of the buffer pools that you specified when you created or altered them.
When the optimizer chooses the access plan, it considers the I/O cost of fetching pages from disk to the buffer pool and estimates the number of I/Os required to satisfy a query. The estimate includes a prediction of bufferpool usage, because additional physical I/Os are not required to read rows in a page that is already in the buffer pool.

The optimizer considers the value of the npages column in the SYSCAT.BUFFERPOOLS system catalog tables and, in partitioned database environments, the SYSCAT.BUFFERPOOLDBPARTITIONS system catalog tables.

The I/O costs of reading the tables can have an impact on how two tables are joined and if an unclustered index is used to read the data

- Default Degree (**dft_degree**)

The **dft_degree** configuration parameter specifies parallelism by providing a default value for the **CURRENT DEGREE** special register and the DEGREE bind option. A value of one (1) means no intrapartition parallelism. A value of minus one (-1) means the optimizer determines the degree of intrapartition parallelism based on the number of processors and the type of query.

Note: Intra-parallel processing does not occur unless you enable it by setting the **intra_parallel** database manager configuration parameter.

- Default Query Optimization Class (**dft_queryopt**)

Although you can specify a query optimization class when you compile SQL or XQuery queries, you can also set a default query optimization class.

- Average Number of Active Applications (**avg_apps**)

The optimizer uses the **avg_apps** parameter to help estimate how much of the buffer pool might be available at run-time for the access plan chosen. Higher values for this parameter can influence the optimizer to choose access plans that are more conservative in buffer pool usage. If you specify a value of one (1), the optimizer considers that the entire buffer pool is available to the application.

- Sort Heap Size (**sortheap**)

If the rows to be sorted occupy more than the space available in the sort heap, several sort passes are performed, where each pass sorts a subset of the entire set of rows. Each sort pass is stored in a system temporary table in the buffer pool, which might be written to disk. When all the sort passes are complete, these sorted subsets are merged into a single sorted set of rows. A sort that does not require a system temporary table to store the list of data always results in better performance and is used if possible.

When choosing an access plan, the optimizer estimates the cost of the sort operations, including evaluating whether a sort can be read in a single, sequential access, by estimating the amount of data to be sorted and looking at the **sortheap** parameter to determine if there is enough space to read a sort in a single, sequential access.

- Maximum Storage for Lock List (**locklist**) and Maximum Percent of Lock List Before Escalation (**maxlocks**)

When the isolation level is repeatable read (RR), the optimizer considers the values of the **locklist** and **maxlocks** parameters to determine whether row level locks might be escalated to a table level lock. If the optimizer estimates that lock escalation might occur for a table access, then it chooses a table level lock for the access plan, instead of incurring the overhead of lock escalation during the query execution.

- CPU Speed (**cpuspeed**)

The optimizer uses the CPU speed to estimate the cost of performing certain operations. CPU cost estimates and various I/O cost estimates help select the best access plan for a query.

The CPU speed of a machine can have a significant influence on the access plan chosen. This configuration parameter is automatically set to an appropriate value when the database is installed or upgraded. Do not adjust this parameter unless you are modelling a production environment on a test system or assessing the

impact of a hardware change. Using this parameter to model a different hardware environment allows you to find out the access plans that might be chosen for that environment. To have the database manager recompute the value of this automatic configuration parameter, set it to minus one (-1).

- Statement Heap Size (**stmtheap**)

Although the size of the statement heap does not influence the optimizer in choosing different access paths, it can affect the amount of optimization performed for complex SQL or XQuery statements.

If the **stmtheap** parameter is not set to a sufficient value, you might receive a warning indicating that there is not enough memory available to process the statement. For example, SQLCODE +437 (SQLSTATE 01602) might indicate that the amount of optimization that has been used to compile a statement is less than the amount that you requested.

- Communications Bandwidth (**comm_bandwidth**)

Communications bandwidth is used by the optimizer to determine access paths. The optimizer uses the value in this parameter to estimate the cost of performing certain operations between the database partition servers in a partitioned database environment.

- Application Heap Size (**applheapsz**)

Large schemas require sufficient space in the application heap.

Chapter 20. Memory allocation

Memory allocation and deallocation occurs at various times. Memory might be allocated to a particular memory area when a specific event occurs (for example, when an application connects), or it might be reallocated in response to a configuration change.

Figure 71 shows the different memory areas that the database manager allocates for various uses and the configuration parameters that enable you to control the size of these memory areas. Note that in a partitioned database environment, each database partition has its own database manager shared memory set.

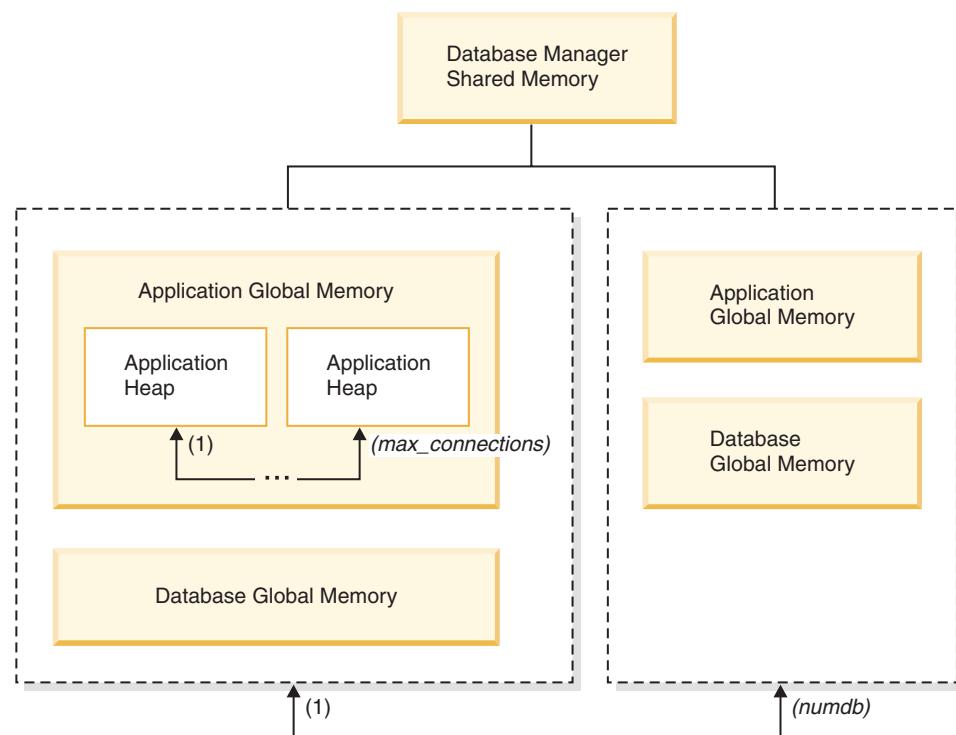


Figure 71. Types of memory allocated by the database manager

Memory is allocated by the database manager whenever one of the following events occurs:

When the database manager starts (`db2start`)

Database manager shared memory (also known as *instance shared memory*) remains allocated until the database manager stops (`db2stop`). This area contains information that the database manager uses to manage activity across all database connections. DB2 automatically controls the size of the database manager shared memory.

When a database is activated or connected to for the first time

Database global memory is used across all applications that connect to the database. The size of the database global memory is specified by the **database_memory** database configuration parameter. By default, this parameter is set to automatic, allowing DB2 to calculate the initial amount

of memory allocated for the database and to automatically configure the database memory size during run time based on the needs of the database.

The following memory areas can be dynamically adjusted:

- Buffer pools (using the ALTER BUFFERPOOL statement)
- Database heap (including log buffers)
- Utility heap
- Package cache
- Catalog cache
- Lock list

The **sortheap**, **sheapthres_shr**, and **sheapthres** configuration parameters are also dynamically updatable. The only restriction is that **sheapthres** cannot be dynamically changed from 0 to a value that is greater than zero, or vice versa.

Shared sort operations are performed by default, and the amount of database shared memory that can be used by sort memory consumers at any one time is determined by the value of the **sheapthres_shr** database configuration parameter. Private sort operations are performed only if intra-partition parallelism, database partitioning, and the connection concentrator are all disabled, and the **sheapthres** database manager configuration parameter is set to a non-zero value.

When an application connects to a database

Each application has its own *application heap*, part of the *application global memory*. You can limit the amount of memory that any one application can allocate by using the **applheapsz** database configuration parameter, or limit overall application memory consumption by using the **appl_memory** database configuration parameter.

When an agent is created

Agent private memory is allocated for an agent when that agent is assigned as the result of a connect request or a new SQL request in a partitioned database environment. Agent private memory contains memory that is used only by this specific agent. If private sort operations have been enabled, the private sort heap is allocated from agent private memory.

The following configuration parameters limit the amount of memory that is allocated for each type of memory area. Note that in a partitioned database environment, this memory is allocated on each database partition.

numdb This database manager configuration parameter specifies the maximum number of concurrent active databases that different applications can use. Because each database has its own global memory area, the amount of memory that can be allocated increases if you increase the value of this parameter.

maxappls

This database configuration parameter specifies the maximum number of applications that can simultaneously connect to a specific database. The value of this parameter affects the amount of memory that can be allocated for both agent private memory and application global memory for that database.

max_connections

This database manager configuration parameter limits the number of database connections or instance attachments that can access the data server at any one time.

max_coordagents

This database manager configuration parameter limits the number of database manager coordinating agents that can exist simultaneously across all active databases in an instance (and per database partition in a partitioned database environment). Together with **maxapps** and **max_connections**, this parameter limits the amount of memory that is allocated for agent private memory and application global memory.

You can use the memory tracker, invoked by the **db2mtrk** command, to view the current allocation of memory within the instance. You can also use the ADMIN_GET_MEM_USAGE table function to determine the total memory consumption for the entire instance or for just a single database partition. Use the MON_GET_MEMORY_SET and MON_GET_MEMORY_POOL table functions to examine the current memory usage at the instance, database, or application level.

On UNIX and Linux operating systems, although the **ipcs** command can be used to list all the shared memory segments, it does not accurately reflect the amount of resources consumed. You can use the **db2mtrk** command as an alternative to **ipcs**.

Database manager shared memory

Database manager shared memory is organized into several different memory areas. Configuration parameters enable you to control the sizes of these areas.

Figure 72 on page 578 shows how database manager shared memory is allocated.

Database manager shared memory (including FCM)

Monitor heap (*mon_heap_sz*) Audit buffer size (*audit_buf_sz*)

Database global memory (database_memory)

Utility heap
(*util_heap_sz*)

Backup buffer
Package cache
(*pckcachesz*)

Sort heap threshold
for shared sort
memory consumers
(*sheapthres_shr*)

Sort heap (*sortheap*)
Lock list (*locklist*)

Catalog cache
(*catalogcache_sz*)

Database heap
(*dbheap*)

Log buffer (*logbufsz*)
Buffer pools

Application global memory (appl_memory)

Shared application memory

Application-specific memory

Application heap (*applheapsz*)
Statistics heap (*stat_heap_sz*)
Statement heap (*stmtheadp*)

Agent private memory

Agent stack
(*agent_stack_sz*)

Sort heap threshold
for private sort
memory consumers
(*sheapthres*)

Java heap
(*java_heap_sz*)

Sort heap (*sortheap*)

Client I/O block
(*rqrioblk*) (remote)

Agent/Application shared memory

Application support
layer heap (*aslheapsz*)

Client I/O block
(*rqrioblk*) (local)

Note: Box size does not indicate relative size of memory.

Figure 72. How memory is used by the database manager

Database Manager Shared Memory

Database Manager Shared Memory is affected by the following configuration parameters:

- The **audit_buf_sz** configuration parameter determines the size of the buffer used in database auditing activities.

- The **mon_heap_sz** configuration parameter determines the size of the memory area used for database system monitoring data.
- For partitioned database systems, the Fast Communications Manager (FCM) requires substantial memory space, especially if the value of **fcm_num_buffers** is large. The FCM memory requirements are allocated from the FCM Buffer Pool.

Database global memory

Database global memory is affected by the size of the buffer pools and by the following database configuration parameters:

- **catalogcache_sz**
- **database_memory**
- **dbheap**
- **locklist**
- **pckcachesz**
- **sheapthres_shr**
- **util_heap_sz**

Application global memory

Application global memory can be controlled by the **appl_memory** configuration parameter. The following database configuration parameters can be used to limit the amount of memory that any one application can consume:

- **applheapsz**
- **stat_heap_sz**
- **stmtheap**

Agent private memory

Each agent requires its own private memory region. The data server creates as many agents as it needs and in accordance with configured memory resources. You can control the maximum number of coordinator agents using the **max_coordagents** database manager configuration parameter. The maximum size of each agent's private memory region is determined by the values of the following configuration parameters:

- **agent_stack_sz**
- **sheapthres** and **sortheap**

Agent/Application shared memory

The total number of agent/application shared memory segments for local clients is limited by the lesser of the following two values:

- The total value of the **maxapp1s** database configuration parameter for all active databases
- The value of the **max_coordagents** database configuration parameter

Note: In configurations where engine concentration is enabled (**max_connections > max_coordagents**), application memory consumption is limited by **max_connections**.

Agent/Application shared memory is also affected by the following database configuration parameters:

- **as1heapsz**
- **rqrioblk**

The FCM buffer pool and memory requirements

In a partitioned database system, the fast communication manager (FCM) buffer shared memory is allocated from the database manager shared memory.

This is shown in Figure 73.

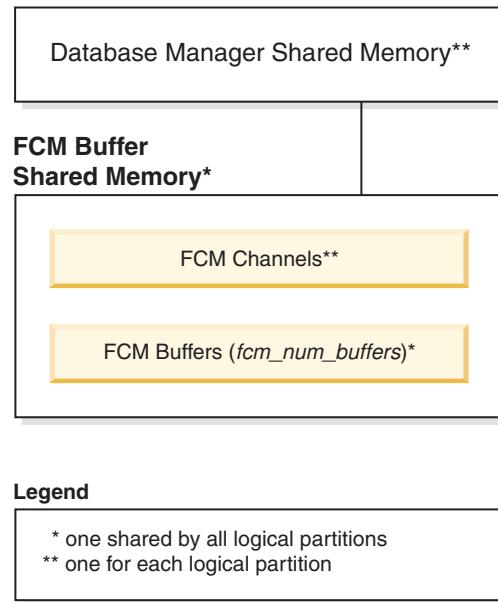


Figure 73. The FCM buffer pool when multiple logical partitions are used

The number of FCM buffers for each database partition is controlled by the **fcm_num_buffers** database manager configuration parameter. By default, this parameter is set to automatic. To tune this parameter manually, use data from the **buff_free** and **buff_free_bottom** system monitor elements.

The number of FCM channels for each database partition is controlled by the **fcm_num_channels** database manager configuration parameter. By default, this parameter is set to automatic. To tune this parameter manually, use data from the **ch_free** and **ch_free_bottom** system monitor elements.

The DB2 database manager can automatically manage FCM memory resources by allocating more FCM buffers and channels as needed. This leads to improved performance and prevents “out of FCM resource” runtime errors. On the Linux operating system, the database manager can preallocate a larger amount of system memory for FCM buffers and channels, up to a maximum default amount of 4 GB. Memory space is impacted only when additional FCM buffers or channels are required. To enable this behavior, set the **FCM_MAXIMIZE_SET_SIZE** option of the **DB2_FCM_SETTINGS** registry variable to YES (or TRUE). YES is the default value.

Guidelines for tuning parameters that affect memory usage

When tuning memory manually (that is, when not using the self-tuning memory manager), benchmark tests provide the best information about setting appropriate values for memory parameters.

In benchmark testing, representative and worst-case SQL statements are run against the server, and the values of memory parameters are changed until a point

of diminishing returns for performance is found. This is the point at which additional memory allocation provides no further performance value to the application.

The upper memory allocation limits for several parameters might be beyond the scope of existing hardware and operating systems. These limits allow for future growth. It is good practice to not set memory parameters at their highest values unless those values can be justified. This applies even to systems that have plenty of available memory. The idea is to prevent the database manager from quickly taking up all of the available memory on a system. Moreover, managing large amounts of memory incurs additional overhead.

For most configuration parameters, memory is committed as it is required, and the parameter settings determine the maximum size of a particular memory heap. For buffer pools and the following configuration parameters, however, all of the specified memory is allocated:

- **as1heapsz**
- **fcm_num_buffers**
- **fcm_num_channels**
- **locklist**

For valid parameter ranges, refer to the detailed information about each parameter.

Chapter 21. Configuring memory and memory heaps

With the simplified memory configuration feature, you can configure memory and memory heaps required by the DB2 data server by using the default AUTOMATIC setting for most memory-related configuration parameters, thereby, requiring much less tuning.

The simplified memory configuration feature provides the following benefits:

- You can use a single parameter, **instance_memory**, to specify all of the memory that the database manager is allowed to allocate from its private and shared memory heaps. Also, you can use the **appl_memory** configuration parameter to control the maximum amount of application memory that is allocated by DB2 database agents to service application requests.
- You are not required to manually tune parameters used solely for functional memory.
- You can use the **db2mtrk** command to monitor heap usage and the **ADMIN_GET_MEM_USAGE** table function to query overall memory consumption.
- The default DB2 configuration requires much less tuning, a benefit for new instances that you create.

The following table lists the memory configuration parameters whose values default to the AUTOMATIC setting. These parameters can also be configured dynamically, if necessary. Note that the meaning of the AUTOMATIC setting differs with each parameter, as described in the rightmost column.

Table 40. Memory configuration parameters whose values default to AUTOMATIC

Configuration parameter name	Description	Meaning of the AUTOMATIC setting
appl_memory	Controls the maximum amount of application memory that is allocated by DB2 database agents to service application requests.	If an instance_memory limit is enforced, the AUTOMATIC setting allows all application memory requests as long as the total amount of memory allocated by the database partition is within the instance_memory limit. Otherwise, it allows request as long as there are system resources available.
applheapsz	Starting with Version 9.5, this parameter refers to the total amount of application memory that can be consumed by the entire application. For partitioned database environments, Concentrator, or SMP configurations, this means that you might need to increase the applheapsz value used in previous releases unless you use the AUTOMATIC setting.	The AUTOMATIC setting allows the application heap size to increase, as needed. A limit might be enforced if there is an appl_memory limit or an instance_memory limit.

Table 40. Memory configuration parameters whose values default to AUTOMATIC (continued)

Configuration parameter name	Description	Meaning of the AUTOMATIC setting
database_memory	Specifies the amount of shared memory that is reserved for the database shared memory region.	When enabled, the memory tuner determines the overall memory requirements for the database and increases or decreases the amount of memory allocated for database shared memory depending on the current database requirements. Starting with Version 9.5, AUTOMATIC is the default setting for all DB2 server products.
dbheap	Determines the maximum memory used by the database heap.	The AUTOMATIC setting allows the database heap to increase as needed. A limit might be enforced if there is a database_memory limit or an instance_memory limit.
instance_memory	If you are using a DB2 database products with memory usage restrictions or if you set this parameter to a specific value, this parameter specifies the maximum amount of memory that can be allocated for a database partition.	The AUTOMATIC setting allows the overall memory consumed by the entire database manager instance to grow as needed, and STMM ensures that sufficient system memory is available to prevent memory overcommitment. For DB2 database products with memory usage restrictions, the AUTOMATIC setting enforces a limit based on the lower of a computed value (75-95% of RAM) and the allowable memory usage under the license. See instance_memory for details on when it is enforced as a limit.
mon_heap_sz	Determines the amount of the memory, in pages, to allocate for database system monitor data.	The AUTOMATIC setting allows the monitor heap to increase as needed. A limit might be enforced if there is an instance_memory limit.
stat_heap_sz	Indicates the maximum size of the heap used in collecting statistics using the RUNSTATS command.	The AUTOMATIC setting allows the statistics heap size to increase as needed. A limit might be enforced if there is an appl_memory limit or an instance_memory limit.
stmtheap	Specifies the size of the statement heap which is used as a work space for the SQL or XQuery compiler to compile an SQL or XQuery statement.	The AUTOMATIC setting allows the statement heap to increase as needed. A limit might be enforced if there is an appl_memory limit or an instance_memory limit.

Note: The DBMCFG and DBCFG administrative views retrieve database manager configuration parameter information for the currently connected database for all database partitions. For the **mon_heap_sz**, **stmtheap**, and **stat_heap_sz** configuration parameters, the DEFERRED_VALUE column on this view does not persist across database activations. That is, when you issue the **get dbm cfg show detail** or **get db cfg show detail** command, the output from the query shows updated (in memory) values.

The following table shows whether configuration parameters are set to the default AUTOMATIC value during instance upgrade or creation and during database upgrade or creation.

Table 41. Configuration parameters set to AUTOMATIC during instance and database upgrade and creation

Configuration parameters	Set to AUTOMATIC upon instance upgrade or creation	Set to AUTOMATIC upon database upgrade	Set to AUTOMATIC upon database creation
applheapsz¹		X	X
dbheap		X	X
instance_memory	X		
mon_heap_sz¹	X		
stat_heap_sz¹		X	X
stmtheap¹			X

As part of the move to simplified memory configuration, the following elements have been deprecated:

- Configuration parameters **appgroup_mem_sz**, **groupheap_ratio**, and **app_ctl_heap_sz**. These configuration parameters are replaced with the new **appl_memory** configuration parameter.
- The **-p** parameter of the **db2mtrk** memory tracker command. This option, which lists private agent memory heaps, is replaced with the **-a** parameter, which lists all application memory consumption.

Agent and process model configuration

Starting with Version 9.5, DB2 databases feature a less complex and more flexible mechanism for configuring process model-related parameters. This simplified configuration eliminates the need for regular adjustments to these parameters and reduces the time and effort required to configure them. It also eliminates the need to shut down and restart DB2 instances to have the new values take effect.

To allow for dynamic and automatic agent and memory configuration, slightly more memory resources are required when an instance is activated.

Chapter 22. Self-tuning memory

A memory-tuning feature simplifies the task of memory configuration by automatically setting values for several memory configuration parameters. When enabled, the memory tuner dynamically distributes available memory resources among the following memory consumers: buffer pools, locking memory, package cache, and sort memory.

The tuner works within the memory limits that are defined by the **database_memory** configuration parameter. The value of this parameter can be automatically tuned as well. When self-tuning is enabled (when the value of **database_memory** has been set to AUTOMATIC), the tuner determines the overall memory requirements for the database and increases or decreases the amount of memory allocated for database shared memory, depending on current database requirements. For example, if current database requirements are high and there is sufficient free memory on the system, more memory is allocated for database shared memory. If the database memory requirements decrease, or if the amount of free memory on the system becomes too low, some database shared memory is released.

If the **database_memory** configuration parameter is not set to AUTOMATIC, the database uses the amount of memory that has been specified for this parameter, distributing it across the memory consumers as required. You can specify the amount of memory in one of two ways: by setting **database_memory** to some numeric value or by setting it to COMPUTED. In the latter case, the total amount of memory is based on the sum of the initial values of the database memory heaps at database startup time.

You can also enable the memory consumers for self tuning as follows:

- For buffer pools, use the ALTER BUFFERPOOL or the CREATE BUFFERPOOL statement (specifying the AUTOMATIC keyword)
- For locking memory, use the **locklist** or the **maxlocks** database configuration parameter (specifying a value of AUTOMATIC)
- For the package cache, use the **pckcachesz** database configuration parameter (specifying a value of AUTOMATIC)
- For sort memory, use the **sheapthres_shr** or the **sortheap** database configuration parameter (specifying a value of AUTOMATIC)

Changes resulting from self-tuning operations are recorded in memory tuning log files that are located in the **stmmlog** subdirectory. These log files contain summaries of the resource demands from each memory consumer during specific tuning intervals, which are determined by timestamps in the log entries.

If little memory is available, the performance benefits of self tuning will be limited. Because tuning decisions are based on database workload, workloads with rapidly changing memory requirements limit the effectiveness of the self-tuning memory manager (STMM). If the memory characteristics of your workload are constantly changing, the STMM will tune less frequently and under shifting target conditions. In this scenario, the STMM will not achieve absolute convergence, but will try instead to maintain a memory configuration that is tuned to the current workload.

Self-tuning memory configuration

Enablement of self-tuning memory and memory consumers is controlled by database configuration parameters.

Self-tuning memory is enabled through the **self_tuning_mem** database configuration parameter.

The following memory-related database configuration parameters can be automatically tuned:

- **database_memory** - Database shared memory size
- **locklist** - Maximum storage for lock list
- **maxlocks** - Maximum percent of lock list before escalation
- **pckcachesz** - Package cache size
- **sheapthres_shr** - Sort heap threshold for shared sorts
- **sortheap** - Sort heap size

Enabling self-tuning memory

Self-tuning memory simplifies the task of memory configuration by automatically setting values for memory configuration parameters and sizing buffer pools.

About this task

When enabled, the memory tuner dynamically distributes available memory resources between several memory consumers, including buffer pools, locking memory, package cache, and sort memory.

Procedure

1. Enable self-tuning memory for the database by setting the **self_tuning_mem** database configuration parameter to **ON** using the **UPDATE DATABASE CONFIGURATION** command or the db2CfgSet API.
2. To enable the self tuning of memory areas that are controlled by memory configuration parameters, set the relevant configuration parameters to **AUTOMATIC** using the **UPDATE DATABASE CONFIGURATION** command or the db2CfgSet API.
3. To enable the self tuning of a buffer pool, set the buffer pool size to **AUTOMATIC** using the **CREATE BUFFERPOOL** statement or the **ALTER BUFFERPOOL** statement. In a partitioned database environment, that buffer pool should not have any entries in SYSCAT.BUFFERPOOLDBPARTITIONS.

Results

Note:

1. Because self-tuned memory is distributed between different memory consumers, at least two memory areas must be concurrently enabled for self tuning at any given time; for example, locking memory and database shared memory. The memory tuner actively tunes memory on the system (the value of the **self_tuning_mem** database configuration parameter is **ON**) when one of the following conditions is true:
 - One configuration parameter or buffer pool size is set to **AUTOMATIC**, and the **database_memory** database configuration parameter is set to either a numeric value or to **AUTOMATIC**

- Any two of **locklist**, **sheapthres_shr**, **pckcachesz**, or buffer pool size is set to AUTOMATIC
 - The **sortheap** database configuration parameter is set to AUTOMATIC
2. The value of the **locklist** database configuration parameter is tuned together with the **maxlocks** database configuration parameter. Disabling self tuning of the **locklist** parameter automatically disables self tuning of the **maxlocks** parameter, and enabling self tuning of the **locklist** parameter automatically enables self tuning of the **maxlocks** parameter.
 3. Automatic tuning of **sortheap** or the **sheapthres_shr** database configuration parameter is allowed only when the database manager configuration parameter **sheapthres** is set to 0.
 4. The value of **sortheap** is tuned together with **sheapthres_shr**. Disabling self tuning of the **sortheap** parameter automatically disables self tuning of the **sheapthres_shr** parameter, and enabling self tuning of the **sheapthres_shr** parameter automatically enables self tuning of the **sortheap** parameter.
 5. Self-tuning memory runs only on the high availability disaster recovery (HADR) primary server. When self-tuning memory is activated on an HADR system, it will never run on the secondary server, and it runs on the primary server only if the configuration is set properly. If the HADR database roles are switched, self-tuning memory operations will also switch so that they run on the new primary server. After the primary database starts, or the standby database converts to a primary database through takeover, the self-tuning memory manager (STMM) engine dispatchable unit (EDU) might not start until the first client connects.

Disabling self-tuning memory

Self-tuning memory can be disabled for the entire database or for one or more configuration parameters or buffer pools.

About this task

If self-tuning memory is disabled for the entire database, the memory configuration parameters and buffer pools that are set to AUTOMATIC remain enabled for automatic tuning; however, the memory areas remain at their current size.

Procedure

1. Disable self-tuning memory for the database by setting the **self_tuning_mem** database configuration parameter to OFF using the **UPDATE DATABASE CONFIGURATION** command or the db2CfgSet API.
2. To disable the self tuning of memory areas that are controlled by memory configuration parameters, set the relevant configuration parameters to MANUAL or specify numeric parameter values using the **UPDATE DATABASE CONFIGURATION** command or the db2CfgSet API.
3. To disable the self tuning of a buffer pool, set the buffer pool size to a specific value using the **ALTER BUFFERPOOL** statement.

Results

Note:

- In some cases, a memory configuration parameter can be enabled for self tuning only if another related memory configuration parameter is also enabled. This means that, for example, disabling self-tuning memory for the **locklist** or the

sortheap database configuration parameter disables self-tuning memory for the **maxlocks** or the **sheapthres_shr** database configuration parameter, respectively.

Determining which memory consumers are enabled for self tuning

You can view the self-tuning memory settings that are controlled by configuration parameters or that apply to buffer pools.

About this task

It is important to note that responsiveness of the memory tuner is limited by the time required to resize a memory consumer. For example, reducing the size of a buffer pool can be a lengthy process, and the performance benefits of trading buffer pool memory for sort memory might not be immediately realized.

Procedure

- To view the settings for configuration parameters, use one of the following methods:
 - Use the **GET DATABASE CONFIGURATION** command, specifying the **SHOW DETAIL** parameter.

The memory consumers that can be enabled for self tuning are grouped together in the output as follows:

Description	Parameter	Current Value	Delayed Value
Self tuning memory	(SELF_TUNING_MEM) = ON (Active)	ON	
Size of database shared memory (4KB)	(DATABASE_MEMORY) = AUTOMATIC(37200)	AUTOMATIC(37200)	
Max storage for lock list (4KB)	(LOCKLIST) = AUTOMATIC(7456)	AUTOMATIC(7456)	
Percent. of lock lists per application	(MAXLOCKS) = AUTOMATIC(98)	AUTOMATIC(98)	
Package cache size (4KB)	(PCKCACHESZ) = AUTOMATIC(5600)	AUTOMATIC(5600)	
Sort heap thres for shared sorts (4KB)	(SHEAPTHRES_SHR) = AUTOMATIC(5000)	AUTOMATIC(5000)	
Sort list heap (4KB)	(SORTHEAP) = AUTOMATIC(256)	AUTOMATIC(256)	

- Use the db2CfgGet API.

The following values are returned:

SQLF_OFF	0
SQLF_ON_ACTIVE	2
SQLF_ON_INACTIVE	3

SQLF_ON_ACTIVE indicates that self tuning is both enabled and active, whereas **SQLF_ON_INACTIVE** indicates that self tuning is enabled but currently inactive.

- To view the self-tuning settings for buffer pools, use one of the following methods:
 - To retrieve a list of the buffer pools that are enabled for self tuning from the command line, use the following query:

```
SELECT BPNAME, NPAGES FROM SYSCAT.BUFFERPOOLS
```

When self tuning is enabled for a buffer pool, the NPAGES field in the SYSCAT.BUFFERPOOLS view for that particular buffer pool is set to -2. When self tuning is disabled, the NPAGES field is set to the current size of the buffer pool.

- To determine the current size of buffer pools that are enabled for self tuning, use the **GET SNAPSHOT** command and examine the current size of the buffer pools (the value of the **bp_cur_buffsz** monitor element):

```
GET SNAPSHOT FOR BUFFERPOOLS ON database-alias
```

An ALTER BUFFERPOOL statement that specifies the size of a buffer pool on a particular database partition creates an exception entry (or updates an

existing entry) for that buffer pool in the SYSCAT.BUFFERPOOLDBPARTITIONS catalog view. If an exception entry for a buffer pool exists, that buffer pool does not participate in self-tuning operations when the default buffer pool size is set to AUTOMATIC.

Self-tuning memory in partitioned database environments

When using the self-tuning memory feature in partitioned database environments, there are a few factors that determine whether the feature will tune the system appropriately.

When self-tuning memory is enabled for partitioned databases, a single database partition is designated as the tuning partition, and all memory tuning decisions are based on the memory and workload characteristics of that database partition. After tuning decisions on that partition are made, the memory adjustments are distributed to the other database partitions to ensure that all database partitions maintain similar configurations.

The single tuning partition model assumes that the feature will be used only when all of the database partitions have similar memory requirements. Use the following guidelines when determining whether to enable self-tuning memory on your partitioned database.

Cases where self-tuning memory for partitioned databases is recommended

When all database partitions have similar memory requirements and are running on similar hardware, self-tuning memory can be enabled without any modifications. These types of environments share the following characteristics:

- All database partitions are on identical hardware, and there is an even distribution of multiple logical database partitions to multiple physical database partitions
- There is a perfect or near-perfect distribution of data
- Workloads are distributed evenly across database partitions, meaning that no database partition has higher memory requirements for one or more heaps than any of the others

In such an environment, if all database partitions are configured equally, self-tuning memory will properly configure the system.

Cases where self-tuning memory for partitioned databases is recommended with qualification

In cases where most of the database partitions in an environment have similar memory requirements and are running on similar hardware, it is possible to use self-tuning memory as long as some care is taken with the initial configuration. These systems might have one set of database partitions for data, and a much smaller set of coordinator partitions and catalog partitions. In such environments, it can be beneficial to configure the coordinator partitions and catalog partitions differently than the database partitions that contain data.

Self-tuning memory should be enabled on all of the database partitions that contain data, and one of these database partitions should be designated as the tuning partition. And because the coordinator and catalog partitions might be configured differently, self-tuning memory should be disabled on those partitions. To disable self-tuning memory on the coordinator and catalog partitions, set the

`self_tuning_mem` database configuration parameter on these partitions to OFF.

Cases where self-tuning memory for partitioned databases is not recommended

If the memory requirements of each database partition are different, or if different database partitions are running on significantly different hardware, it is good practice to disable the self-tuning memory feature. You can disable the feature by setting the `self_tuning_mem` database configuration parameter to OFF on all partitions.

Comparing the memory requirements of different database partitions

The best way to determine whether the memory requirements of different database partitions are sufficiently similar is to consult the snapshot monitor. If the following snapshot elements are similar on all database partitions (differing by no more than 20%), the memory requirements of the database partitions can be considered sufficiently similar.

Collect the following data by issuing the command: get snapshot for database on <dbname>

Locks held currently	= 0
Lock waits	= 0
Time database waited on locks (ms)	= 0
Lock list memory in use (Bytes)	= 4968
Lock escalations	= 0
Exclusive lock escalations	= 0
Total Shared Sort heap allocated	= 0
Shared Sort heap high water mark	= 0
Post threshold sorts (shared memory)	= 0
Sort overflows	= 0
Package cache lookups	= 13
Package cache inserts	= 1
Package cache overflows	= 0
Package cache high water mark (Bytes)	= 655360
Number of hash joins	= 0
Number of hash loops	= 0
Number of hash join overflows	= 0
Number of small hash join overflows	= 0
Post threshold hash joins (shared memory)	= 0
Number of OLAP functions	= 0
Number of OLAP function overflows	= 0
Active OLAP functions	= 0

Collect the following data by issuing the command: get snapshot for bufferpools on <dbname>

Buffer pool data logical reads	= 0
Buffer pool data physical reads	= 0
Buffer pool index logical reads	= 0
Buffer pool index physical reads	= 0
Total buffer pool read time (milliseconds)	= 0
Total buffer pool write time (milliseconds)	= 0

Using self-tuning memory in partitioned database environments

When self-tuning memory is enabled in partitioned database environments, there is a single database partition (known as the *tuning partition*) that monitors the memory configuration and propagates any configuration changes to all other database partitions to maintain a consistent configuration across all the participating database partitions.

The tuning partition is selected on the basis of several characteristics, such as the number of database partitions in the partition group and the number of buffer pools.

- To determine which database partition is currently specified as the tuning partition, call the **ADMIN_CMD** procedure as follows:

```
CALL SYSPROC.ADMIN_CMD('get stmm tuning dbpartitionnum')
```

- To change the tuning partition, call the **ADMIN_CMD** procedure as follows:

```
CALL SYSPROC.ADMIN_CMD('update stmm tuning dbpartitionnum <partitionnum>')
```

The tuning partition is updated asynchronously or at the next database startup. To have the memory tuner automatically select the tuning partition, enter -1 for the *partitionnum* value.

Starting the memory tuner in partitioned database environments

In a partitioned database environment, the memory tuner will start only if the database is activated by an explicit **ACTIVATE DATABASE** command, because self-tuning memory requires that all partitions be active.

Disabling self-tuning memory for a specific database partition

- To disable self-tuning memory for a subset of database partitions, set the **self_tuning_mem** database configuration parameter to OFF for those database partitions.
- To disable self-tuning memory for a subset of the memory consumers that are controlled by configuration parameters on a specific database partition, set the value of the relevant configuration parameter or the buffer pool size to MANUAL or to some specific value on that database partition. It is recommended that self-tuning memory configuration parameter values be consistent across all running partitions.
- To disable self-tuning memory for a particular buffer pool on a specific database partition, issue the **ALTER BUFFERPOOL** statement, specifying a size value and the partition on which self-tuning memory is to be disabled.

An **ALTER BUFFERPOOL** statement that specifies the size of a buffer pool on a particular database partition will create an exception entry (or update an existing entry) for that buffer pool in the **SYSCAT.BUFFERPOOLDBPARTITIONS** catalog view. If an exception entry for a buffer pool exists, that buffer pool will not participate in self-tuning operations when the default buffer pool size is set to AUTOMATIC. To remove an exception entry so that a buffer pool can be enabled for self tuning:

1. Disable self tuning for this buffer pool by issuing an **ALTER BUFFERPOOL** statement, setting the buffer pool size to a specific value.
2. Issue another **ALTER BUFFERPOOL** statement to set the size of the buffer pool on this database partition to the default.
3. Enable self tuning for this buffer pool by issuing another **ALTER BUFFERPOOL** statement, setting the buffer pool size to AUTOMATIC.

Enabling self-tuning memory in nonuniform environments

Ideally, data should be distributed evenly across all database partitions, and the workload that is run on each partition should have similar memory requirements. If the data distribution is skewed, so that one or more of your database partitions contain significantly more or less data than other database partitions, these anomalous database partitions should not be enabled for self tuning. The same is true if the memory requirements are skewed across the database partitions, which can happen, for example, if resource-intensive sorts are only performed on one partition, or if some database partitions are associated with different hardware and more available memory than others. Self tuning memory can still be enabled on some database partitions in this type of environment. To take advantage of self-tuning memory in environments with skew, identify a set of database partitions that have similar data and memory requirements and enable them for self tuning. Memory in the remaining partitions should be configured manually.

Chapter 23. Automatic maintenance

The database manager provides automatic maintenance capabilities for performing database backups, keeping statistics current, and reorganizing tables and indexes as necessary. Performing maintenance activities on your databases is essential in ensuring that they are optimized for performance and recoverability.

Maintenance of your database includes some or all of the following activities:

- **Backups.** When you back up a database, the database manager takes a copy of the data in the database and stores it on a different medium in case of failure or damage to the original. Automatic database backups help to ensure that your database is backed up properly and regularly so that you don't have to worry about when to back up or know the syntax of the **BACKUP** command.
- **Data defragmentation (table or index reorganization).** This maintenance activity can increase the efficiency with which the database manager accesses your tables. Automatic reorganization manages an offline table and index reorganization so that you don't need to worry about when and how to reorganize your data.
- **Data access optimization (statistics collection).** The database manager updates the system catalog statistics on the data in a table, the data in indexes, or the data in both a table and its indexes. The optimizer uses these statistics to determine which path to use to access the data. Automatic statistics collection attempts to improve the performance of the database by maintaining up-to-date table statistics. The goal is to allow the optimizer to choose an access plan based on accurate statistics.
- **Statistics profiling.** Automatic statistics profiling advises when and how to collect table statistics by detecting outdated, missing, or incorrect statistics, and by generating statistical profiles based on query feedback.

It can be time-consuming to determine whether and when to run maintenance activities, but automatic maintenance removes the burden from you. You can manage the enablement of the automatic maintenance features simply and flexibly by using the automatic maintenance database configuration parameters. By setting the automatic maintenance database configuration parameters, you can specify your maintenance objectives. The database manager uses these objectives to determine whether the maintenance activities need to be done and runs only the required ones during the next available maintenance window (a time period that you define).

In IBM Data Studio Version 3.1 or later, you can use the task assistant for configuring automatic maintenance. Task assistants can guide you through the process of setting options, reviewing the automatically generated commands to perform the task, and running these commands. For more details, see Administering databases with task assistants.

Maintenance windows

A maintenance window is a time period that you define for the running of automatic maintenance activities, which are backups, statistics collection, statistics profiling, and reorganizations. An offline window might be the time period when access to a database is unavailable. An online window might be the time period when users are permitted to connect to a database.

A maintenance window is different from a task schedule. During a maintenance window, each automatic maintenance activity is not necessarily run. Instead, the database manager evaluates the system to determine the need for each maintenance activity to be run. If the maintenance requirements are not met, the maintenance activity is run. If the database is already well maintained, the maintenance activity is not run.

Think about when you want the automatic maintenance activities to be run. Automatic maintenance activities consume resources on your system and might affect the performance of your database when the activities are run. Some of these activities also restrict access to tables, indexes, and databases. Therefore, you must provide appropriate windows when the database manager can run maintenance activities.

Offline maintenance activities

Offline maintenance activities (offline database backups and table and index reorganizations) are maintenance activities that can occur only in the offline maintenance window. The extent to which user access is affected depends on which maintenance activity is running:

- During an offline backup, no applications can connect to the database. Any currently connected applications are forced off.
- During an offline table or index reorganization (data defragmentation), applications can access but not update the data in tables.

Offline maintenance activities run to completion even if they go beyond the window specified. Over time, the internal scheduling mechanism learns how to best estimate job completion times. If the offline maintenance window is too small for a particular database backup or reorganization activity, the scheduler will not start the job the next time and relies on the health monitor to provide notification of the need to increase the offline maintenance window.

Online maintenance activities

Online maintenance activities (automatic statistics collection and profiling, online index reorganizations, and online database backups) are maintenance activities that can occur only in the online maintenance window. When online maintenance activities run, any currently connected applications are allowed to remain connected, and new connections can be established. To minimize the impact on the system, online database backups and automatic statistics collection and profiling are throttled by the adaptive utility throttling mechanism.

Online maintenance activities run to completion even if they go beyond the window specified.

Chapter 24. Automatic table and index maintenance

After many changes to table data, a table and its indexes can become fragmented. Logically sequential data might be found on nonsequential pages, forcing additional read operations by the database manager to access data.

The statistical information that is collected by the **RUNSTATS** utility shows the distribution of data within a table. Analysis of these statistics can indicate when and what type of reorganization is necessary.

The automatic reorganization process determines the need for table or index reorganization by using formulas that are part of the **REORGCHK** utility. It periodically evaluates tables and indexes that had their statistics updated to see whether reorganization is required, and schedules such operations whenever they are necessary.

The automatic reorganization feature can be enabled or disabled through the **auto_reorg**, **auto_tbl_maint**, and **auto_maint** database configuration parameters.

In a partitioned database environment, the initiation of automatic reorganization is done on the catalog database partition. These configuration parameters are enabled only on the catalog database partition. The **REORG** operation, however, runs on all of the database partitions on which the target tables are found.

If you are unsure about when and how to reorganize your tables and indexes, you can incorporate automatic reorganization as part of your overall database maintenance plan.

You can also reorganize multidimensional clustering (MDC) and insert time clustering (ITC) tables to reclaim space. The freeing of extents from MDC and ITC tables is only supported for tables in DMS table spaces and automatic storage. Freeing extents from your MDC and ITC tables can be done in an online fashion with the RECLAIM EXTENTS option of the REORG TABLE command.

You can also schedule an alternate means to reclaim space from your indexes. The REORG INDEX command has an index clause in which you can specify space-reclaim-options. When you specify RECLAIM EXTENTS in space-reclaim-options, space is released back to the table space in an online fashion. This operation provides space reclamation without the need for a full rebuild of the indexes. The REBUILD option of the REORG INDEX command also reclaims space, but not necessarily in an online fashion.

Automatic reorganization on data partitioned tables

For DB2 Version 9.7 Fix Pack 1 and earlier releases, automatic reorganization supports reorganization of a data partitioned table for the entire table. For DB2 V9.7 Fix Pack 1 and later releases, automatic reorganization supports reorganizing data partitions of a partitioned table and reorganizing the partitioned indexes on a data partition of a partitioned table.

To avoid placing an entire data partitioned table into ALLOW NO ACCESS mode, automatic reorganization performs **REORG INDEXES ALL** operations at the data

partition level on partitioned indexes that need to be reorganized. Automatic reorganization performs **REORG INDEX** operations on any nonpartitioned index that needs to be reorganized.

Automatic reorganization performs the following **REORG TABLE** operations on data partitioned tables:

- If any nonpartitioned indexes (except system-generated XML path indexes) are defined on the table and there is only one partition that needs to be reorganized, automatic reorganization performs a **REORG TABLE** operation with the **ON DATA PARTITION** clause to specify the partition that needs to be reorganized. Otherwise, automatic reorganization performs a **REORG TABLE** on the entire table without the **ON DATA PARTITION** clause.
- If no nonpartitioned indexes (except system-generated XML path indexes) are defined on the table, automatic reorganization performs a **REORG TABLE** operation with the **ON DATA PARTITION** clause on each partition that needs to be reorganized.

Automatic reorganization on volatile tables

You can enable automatic index reorganization for volatile tables. The automatic reorganization process determines whether index reorganization is required for volatile tables and schedules a REORG INDEX CLEANUP. Index reorganization is performed periodically on volatile tables and releases space that can be reused by the indexes defined on these tables.

Statistics cannot be collected in volatile tables because they are updated frequently. To determine what indexes need to be reorganized, automatic reorganization uses the numInxPseudoEmptyPagesForVolatile attribute instead of REORGCHK. The number of pseudo empty pages is maintained internally, visible through mon_get_index, and does not require a RUNSTATS operation like REORGCHK. This attribute in the AUTO_REORG policy indicates how many empty index pages with pseudo deleted keys an index must have so index reorganization is triggered.

To enable automatic index reorganization in volatile tables:

- The **DB2_WORKLOAD** registry variable must be set to SAP.
- Automatic reorganization must be enabled.
- The numInxPseudoEmptyPagesForVolatile attribute must be set.

Chapter 25. Automatic statistics collection

The DB2 optimizer uses catalog statistics to determine the most efficient access plan for a query. Out-of-date or incomplete table or index statistics might lead the optimizer to select a suboptimal plan, which slows down query execution. However, deciding which statistics to collect for a given workload is complex, and keeping these statistics up-to-date is time-consuming.

With automatic statistics collection, part of the DB2 automated table maintenance feature, you can let the database manager determine whether statistics need to be updated. Automatic statistics collection can occur *synchronously* at statement compilation time by using the real-time statistics (RTS) feature, or the **RUNSTATS** command can be enabled to simply run in the background for *asynchronous* collection. Although background statistics collection can be enabled while real-time statistics collection is disabled, background statistics collection must be enabled for real-time statistics collection to occur. Automatic background statistics collection **auto_runstats** and automatic real-time statistics collection **auto_stmt_stats** are enabled by default when you create a database.

Starting with DB2 Version 9, you can use the Configuration Advisor to determine the initial configuration for new databases including the appropriate setting for the **auto_stmt_stats** database configuration parameter.

In IBM Data Studio Version 3.1 or later, you can use the task assistant for configuring automatic statistics collection. Task assistants can guide you through the process of setting options, reviewing the automatically generated commands to perform the task, and running these commands. For more details, see Administering databases with task assistants.

Understanding asynchronous and real-time statistics collection

When real-time statistics collection is enabled, statistics can be fabricated by using certain metadata. *Fabrication* means deriving or creating statistics, rather than collecting them as part of normal **RUNSTATS** command activity. For example, the number of rows in a table can be derived from knowing the number of pages in the table, the page size, and the average row width. In some cases, statistics are not derived, but are maintained by the index and data manager and can be stored directly in the catalog. For example, the index manager maintains a count of the number of leaf pages and levels in each index.

The query optimizer determines how statistics are collected, based on the needs of the query and the amount of table update activity (the number of update, insert, or delete operations).

Real-time statistics collection provides more timely and more accurate statistics. Accurate statistics can result in better query execution plans and improved performance. Regardless of whether real-time statistics is enabled, asynchronous statistics collection occurs at two-hour intervals. This interval might not be frequent enough to provide accurate statistics for some applications.

Real-time statistics collection also initiates asynchronous collection requests when:

- Table activity is not high enough to require synchronous collection, but is high enough to require asynchronous collection

- Synchronous statistics collection used sampling because the table was large
- Synchronous statistics were fabricated
- Synchronous statistics collection failed because the collection time was exceeded

At most, two asynchronous requests can be processed at the same time, but only for different tables. One request must have been initiated by real-time statistics collection, and the other must have been initiated by asynchronous statistics collection checking.

The performance impact of automatic statistics collection is minimized in several ways:

- Asynchronous statistics collection is performed by using a throttled **RUNSTATS** utility. Throttling controls the amount of resource that is consumed by the **RUNSTATS** utility, based on current database activity: as database activity increases, the utility runs more slowly, reducing its resource demands.
- Synchronous statistics collection is limited to 5 seconds per query. This value can be controlled by the RTS optimization guideline. If synchronous collection exceeds the time limit, an asynchronous collection request is submitted.
- Synchronous statistics collection does not store the statistics in the system catalog. Instead, the statistics are stored in a statistics cache and are later stored in the system catalog by an asynchronous operation. This storage sequence avoids the overhead and possible lock contention involved when updating the system catalog. Statistics in the statistics cache are available for subsequent SQL compilation requests.
- Only one synchronous statistics collection operation occurs per table. Other agents requiring synchronous statistics collection fabricate statistics, if possible, and continue with statement compilation. This behavior is also enforced in a partitioned database environment, where agents on different database partitions might require synchronous statistics.
- You can customize the type of statistics that are collected by enabling statistics profiling, which uses information about previous database activity to determine which statistics are required by the database workload, or by creating your own statistics profile for a particular table.
- Only tables with missing statistics or high levels of activity (as measured by the number of update, insert, or delete operations) are considered for statistics collection. Even if a table meets the statistics collection criteria, synchronous statistics are not collected unless query optimization requires them. In some cases, the query optimizer can choose an access plan without statistics.
- For asynchronous statistics collection checking, large tables (tables with more than 4000 pages) are sampled to determine whether high table activity changed the statistics. Statistics for such large tables are collected only if warranted.
- For asynchronous statistics collection, the **RUNSTATS** utility is automatically scheduled to run during the online maintenance window that is specified in your maintenance policy. This policy also specifies the set of tables that are within the scope of automatic statistics collection, further minimizing unnecessary resource consumption.
- Synchronous statistics collection and fabrication do not follow the online maintenance window that is specified in your maintenance policy, because synchronous requests must occur immediately and have limited collection time. Synchronous statistics collection and fabrication follow the policy that specifies the set of tables that are within the scope of automatic statistics collection.
- While automatic statistics collection is being performed, the affected tables are still available for regular database activity (update, insert, or delete operations).

- Real-time statistics (synchronous or fabricated) are not collected for nicknames. To refresh nickname statistics in the system catalog for synchronous statistics collection, call the SYSPROC.NNSTAT procedure. For asynchronous statistics collection, DB2 for Linux, UNIX, and Windows automatically calls the SYSPROC.NNSAT procedure to refresh the nickname statistics in the system catalog.
- Real-time statistics (synchronous or fabricated) are not collected for statistical views.
- Declared temporary tables (DGTTs) can have only Real-time statistics collected.

Although real-time statistics collection is designed to minimize statistics collection overhead, try it in a test environment first to ensure that there is no negative performance impact. There might be a negative performance impact in some online transaction processing (OLTP) scenarios, especially if there is an upper boundary for how long a query can run.

Real-time synchronous statistics collection is performed for regular tables, materialized query tables (MQTs), and global temporary tables. Asynchronous statistics are not collected for global temporary tables. Global temporary tables cannot be excluded from real-time statistics via the automatic maintenance policy facility.

Automatic statistics collection (synchronous or asynchronous) does not occur for:

- Tables that are marked VOLATILE (tables that have the VOLATILE field set in the SYSCAT.TABLES catalog view)
- Created temporary tables (CGTTs)
- Tables that had their statistics manually updated, by issuing UPDATE statements directly against SYSSTAT catalog views

When you modify table statistics manually, the database manager assumes that you are now responsible for maintaining their statistics. To induce the database manager to maintain statistics for a table that had its statistics manually updated, collect statistics by using the **RUNSTATS** command or specify statistics collection when using the **LOAD** command. Tables created before Version 9.5 that had their statistics updated manually before upgrading are not affected, and their statistics are automatically maintained by the database manager until they are manually updated.

Statistics fabrication does not occur for:

- Statistical views
- Tables that had their statistics manually updated, by issuing UPDATE statements directly against SYSSTAT catalog views. If real-time statistics collection is not enabled, some statistics fabrication still occurs for tables that had their statistics manually updated.

In a partitioned database environment, statistics are collected on a single database partition and then extrapolated. The database manager always collects statistics (both synchronous and asynchronous) on the first database partition of the database partition group.

No real-time statistics collection activity will occur until at least five minutes after database activation.

Real-time statistics processing occurs for both static and dynamic SQL.

A table that was truncated, either by using the TRUNCATE statement or by using the **IMPORT** command, is automatically recognized as having out of date statistics.

Automatic statistics collection, both synchronous and asynchronous, invalidates cached dynamic statements that reference tables for which statistics were collected. This is done so that cached dynamic statements can be re-optimized with the latest statistics.

Asynchronous automatic statistics collection operations might be interrupted when the database is deactivated. If the database was not explicitly activated using the **ACTIVATE DATABASE** command or API, then the database is deactivated when the last user disconnects from the database. If operations are interrupted, then error messages might be recorded in the DB2 diagnostic log file. To avoid interrupting asynchronous automatic statistics collection operations, explicitly activate the database.

Real-time statistics and explain processing

There is no real-time processing for a query that is only explained (not executed) by using the EXPLAIN facility. The following table summarizes the behavior under different values of the CURRENT EXPLAIN MODE special register.

Table 42. Real-time statistics collection as a function of the value of the CURRENT EXPLAIN MODE special register

CURRENT EXPLAIN MODE value	Real-time statistics collection considered
YES	Yes
EXPLAIN	No
NO	Yes
REOPT	Yes
RECOMMEND INDEXES	No
EVALUATE INDEXES	No

Automatic statistics collection and the statistics cache

A statistics cache was introduced in DB2 Version 9.5 to make synchronously collected statistics available to all queries. This cache is part of the catalog cache. In a partitioned database environment, the statistics cache resides only on the catalog database partition even though each database partition has a catalog cache. When real-time statistics collection is enabled, catalog cache requirements are higher. Consider tuning the value of the **catalogcache_sz** database configuration parameter when real-time statistics collection is enabled.

Automatic statistics collection and statistical profiles

Synchronous and asynchronous statistics are collected according to a statistical profile that is in effect for a table, with the following exceptions:

- To minimize the overhead of synchronous statistics collection, the database manager might collect statistics by using sampling. In this case, the sampling rate and method might be different from those rates and methods that are specified in the statistical profile.
- Synchronous statistics collection might choose to fabricate statistics, but it might not be possible to fabricate all statistics that are specified in the statistical profile.

For example, column statistics such as COLCARD, HIGH2KEY, and LOW2KEY cannot be fabricated unless the column is leading in some index.

If synchronous statistics collection cannot collect all statistics that are specified in the statistical profile, an asynchronous collection request is submitted.

The following sections explain different operating characteristics of automatic statistics collection.

Chapter 26. Configuration Advisor

You can use the Configuration Advisor to obtain recommendations for the initial values of the buffer pool size, database configuration parameters, and database manager configuration parameters.

To use the Configuration Advisor, specify the **AUTOCONFIGURE** command for an existing database, or specify **AUTOCONFIGURE** as an option of the **CREATE DATABASE** command. To configure your database, you must have SYSADM, SYSCTRL, or SYSMAINT authority.

You can display the recommended values or apply them by specifying the **APPLY** parameter in the **CREATE DATABASE** and **AUTOCONFIGURE** commands. The recommendations are based on input that you provide and system information that the advisor gathers.

The values suggested by the Configuration Advisor are relevant for only one database per instance. If you want to use this advisor on more than one database, each database must belong to a separate instance.

Tuning configuration parameters using the Configuration Advisor

The Configuration Advisor helps you to tune performance and to balance memory requirements for a single database per instance by suggesting which configuration parameters to modify and suggesting values for them. The Configuration Advisor is automatically run when you create a database.

About this task

To disable this feature or to explicitly enable it, use the **db2set** command before creating a database, as follows:

```
db2set DB2_ENABLE_AUTOCONFIG_DEFAULT=NO  
db2set DB2_ENABLE_AUTOCONFIG_DEFAULT=YES
```

To define values for several of the configuration parameters and to determine the scope of the application of those parameters, use the **AUTOCONFIGURE** command, specifying one of the following options:

- NONE, meaning that none of the values are applied
- DB ONLY, meaning that only database configuration and buffer pool values are applied
- DB AND DBM, meaning that all parameters and their values are applied

Note: Even if you automatically enabled the Configuration Advisor when you ran the **CREATE DATABASE** command, you can still specify **AUTOCONFIGURE** command options. If you did not enable the Configuration Advisor when you ran the **CREATE DATABASE** command, you can run the Configuration Advisor manually afterwards.

Example: Requesting configuration recommendations using the Configuration Advisor

This scenario demonstrates to run the Configuration Advisor from the command line to generate recommendations and shows the output that the Configuration Advisor produces.

To run the Configuration Advisor:

1. Connect to the PERSONL database by specifying the following command from the command line:

```
DB2 CONNECT TO PERSONL
```

2. Issue the **AUTOCONFIGURE** command from the CLP, specifying how the database is used. As shown in the following example, set a value of **NONE** for the **APPLY** option to indicate that you want to view the configuration recommendations but not apply them:

```
DB2 AUTOCONFIGURE USING
  MEM_PERCENT 60
  WORKLOAD_TYPE MIXED
  NUM_STMTS 500
  ADMIN_PRIORITY BOTH
  IS_POPULATED YES
  NUM_LOCAL_APPS 0
  NUM_REMOTE_APPS 20
  ISOLATION RR
  BP_RESIZEABLE YES
  APPLY NONE
```

If you are unsure about the value of a parameter for the command, you can omit it, and the default will be used. You can pass up to 10 parameters without values: **MEM_PERCENT**, **WORKLOAD_TYPE**, and so on, as shown in the previous example.

The recommendations generated by the **AUTOCONFIGURE** command are displayed on the screen in table format, as shown in Figure 74 on page 607

Former and Applied Values for Database Manager Configuration			
Description	Parameter	Current Value	Recommended Value
Application support layer heap size (4KB)	(ASLHEAPSZ) = 15	15	
No. of int. communication buffers(4KB)	(FCM_NUM_BUFFERS) = AUTOMATIC	AUTOMATIC	
Enable intra-partition parallelism	(INTRA_PARALLEL) = NO	NO	
Maximum query degree of parallelism	(MAX_QUERYDEGREE) = ANY	1	
Agent pool size	(NUM_POOLAGENTS) = 100(calculated)	200	
Initial number of agents in pool	(NUM_INITAGENTS) = 0	0	
Max requester I/O block size (bytes)	(QRIOIBLK) = 32767	32767	
Sort heap threshold (4KB)	(SHEAPTHRES) = 0	0	

Former and Applied Values for Database Configuration			
Description	Parameter	Current Value	Recommended Value
Default application heap (4KB)	(APPLHEAPSZ) = 256	256	
Catalog cache size (4KB)	(CATALOGCACHE_SZ) = (MAXAPPLS*4)	260	
Changed pages threshold	(CHNGPGS_THRESH) = 60	80	
Database heap (4KB)	(DBHEAP) = 1200	2791	
Degree of parallelism	(DFT_DEGREE) = 1	1	
Default tablespace extentsize (pages)	(DFT_EXTENT_SZ) = 32	32	
Default prefetch size (pages)	(DFT_PREFETCH_SZ) = AUTOMATIC	AUTOMATIC	
Default query optimization class	(DFT_QUERYOPT) = 5	5	
Max storage for lock list (4KB)	(LOCKLIST) = 100	AUTOMATIC	
Log buffer size (4KB)	(LOGBUFSZ) = 8	99	
Log file size (4KB)	(LOGFILSZ) = 1000	1024	
Number of primary log files	(LOGPRIMARY) = 3	8	
Number of secondary log files	(LOGSECOND) = 2	3	
Max number of active applications	(MAXAPPLS) = AUTOMATIC	AUTOMATIC	
Percent. of lock lists per application	(MAXLOCKS) = 10	AUTOMATIC	
Group commit count	(MINCOMMIT) = 1	1	
Number of asynchronous page cleaners	(NUM_IOCLEANERS) = 1	1	
Number of I/O servers	(NUM_IOSERVERS) = 3	4	
Package cache size (4KB)	(PCKCACHESZ) = (MAXAPPLS*8)	1533	
Percent log file reclaimed before soft chkpt	(SOFTMAX) = 100	320	
Sort list heap (4KB)	(SORTHEAP) = 256	AUTOMATIC	
statement heap (4KB)	(STMTH HEAP) = 4096	4096	
Statistics heap size (4KB)	(STAT_HEAP_SZ) = 4384	4384	
Utilities heap size (4KB)	(UTIL_HEAP_SZ) = 5000	113661	
Self tuning memory	(SELF_TUNING_MEM) = ON	ON	
Automatic runstats	(AUTO_RUNSTATS) = ON	ON	
Sort heap thres for shared sorts (4KB)	(SHEAPTHRES_SHR) = 5000	AUTOMATIC	

Former and Applied Values for Bufferpool(s)			
Description	Parameter	Current Value	Recommended Value
IBMDFAULTBP	Bufferpool size = -2	340985	

DB210203I AUTOCONFIGURE completed successfully. Database manager or database configuration values may have been changed. The instance must be restarted before any changes come into effect. You may also want to rebind your packages after the new configuration parameters take effect so that the new values will be used.

Figure 74. Configuration Advisor sample output

If you agree with all of the recommendations, either reissue the **AUTOCONFIGURE** command but specify that you want the recommended values to be applied by using the **APPLY** option, or update individual configuration parameters using the **UPDATE DATABASE MANAGER CONFIGURATION** command and the **UPDATE DATABASE CONFIGURATION** command.

Chapter 27. Utility throttling

Utility throttling regulates the performance impact of maintenance utilities so that they can run concurrently during production periods. Although the impact policy, a setting that allows utilities to run in throttled mode, is defined by default, you must set the impact priority, a setting that each cleaner has indicating its throttling priority, when you run a utility if you want to throttle it.

The throttling system ensures that the throttled utilities are run as frequently as possible without violating the impact policy. You can throttle statistics collection, backup operations, rebalancing operations, and asynchronous index cleanups.

You define the impact policy by setting the **util_impact_lim** configuration parameter.

Cleaners are integrated with the utility throttling facility. By default, each (index) cleaner has a utility impact priority of 50 (acceptable values are between 1 and 100, with 0 indicating no throttling). You can change the priority by using the **SET UTIL_IMPACT_PRIORITY** command or the db2UtilityControl API.

Asynchronous index cleanup

Asynchronous index cleanup (AIC) is the deferred cleanup of indexes following operations that invalidate index entries. Depending on the type of index, the entries can be record identifiers (RIDs) or block identifiers (BIDs). Invalid index entries are removed by index cleaners, which operate asynchronously in the background.

AIC accelerates the process of detaching a data partition from a partitioned table, and is initiated if the partitioned table contains one or more nonpartitioned indexes. In this case, AIC removes all nonpartitioned index entries that refer to the detached data partition, and any pseudo-deleted entries. After all of the indexes have been cleaned, the identifier that is associated with the detached data partition is removed from the system catalog. In DB2 Version 9.7 Fix Pack 1 and later releases, AIC is initiated by an asynchronous partition detach task.

Prior to DB2 Version 9.7 Fix Pack 1, if the partitioned table has dependent materialized query tables (MQTs), AIC is not initiated until after a SET INTEGRITY statement is executed.

Normal table access is maintained while AIC is in progress. Queries accessing the indexes ignore any invalid entries that have not yet been cleaned.

In most cases, one cleaner is started for each nonpartitioned index that is associated with the partitioned table. An internal task distribution daemon is responsible for distributing the AIC tasks to the appropriate table partitions and assigning database agents. The distribution daemon and cleaner agents are internal system applications that appear in **LIST APPLICATIONS** command output with the application names **db2taskd** and **db2aic**, respectively. To prevent accidental disruption, system applications cannot be forced. The distribution daemon remains online as long as the database is active. The cleaners remain active until cleaning has been completed. If the database is deactivated while cleaning is in progress, AIC resumes when you reactivate the database.

AIC impact on performance

AIC incurs minimal performance impact.

An instantaneous row lock test is required to determine whether a pseudo-deleted entry has been committed. However, because the lock is never acquired, concurrency is unaffected.

Each cleaner acquires a minimal table space lock (IX) and a table lock (IS). These locks are released if a cleaner determines that other applications are waiting for locks. If this occurs, the cleaner suspends processing for 5 minutes.

Cleaners are integrated with the utility throttling facility. By default, each cleaner has a utility impact priority of 50. You can change the priority by using the **SET UTIL_IMPACT_PRIORITY** command or the db2UtilityControl API.

Monitoring AIC

You can monitor AIC with the **LIST UTILITIES** command. Each index cleaner appears as a separate utility in the output. The following is an example of output from the **LIST UTILITIES SHOW DETAIL** command:

ID	= 2
Type	= ASYNCHRONOUS INDEX CLEANUP
Database Name	= WSDB
Partition Number	= 0
Description	= Table: USER1.SALES, Index: USER1.I2
Start Time	= 12/15/2005 11:15:01.967939
State	= Executing
Invocation Type	= Automatic
Throttling:	
Priority	= 50
Progress Monitoring:	
Total Work	= 5 pages
Completed Work	= 0 pages
Start Time	= 12/15/2005 11:15:01.979033
ID	= 1
Type	= ASYNCHRONOUS INDEX CLEANUP
Database Name	= WSDB
Partition Number	= 0
Description	= Table: USER1.SALES, Index: USER1.I1
Start Time	= 12/15/2005 11:15:01.978554
State	= Executing
Invocation Type	= Automatic
Throttling:	
Priority	= 50
Progress Monitoring:	
Total Work	= 5 pages
Completed Work	= 0 pages
Start Time	= 12/15/2005 11:15:01.980524

In this case, there are two cleaners operating on the USERS1.SALES table. One cleaner is processing index I1, and the other is processing index I2. The progress monitoring section shows the estimated total number of index pages that need cleaning and the current number of clean index pages.

The State field indicates the current state of a cleaner. The normal state is Executing, but the cleaner might be in Waiting state if it is waiting to be assigned to an available database agent or if the cleaner is temporarily suspended because of lock contention.

Note that different tasks on different database partitions can have the same utility ID, because each database partition assigns IDs to tasks that are running on that database partition only.

Asynchronous index cleanup for MDC tables

You can enhance the performance of a rollout deletion—an efficient method for deleting qualifying blocks of data from multidimensional clustering (MDC) tables by using asynchronous index cleanup (AIC). AIC is the deferred cleanup of indexes following operations that invalidate index entries.

Indexes are cleaned up synchronously during a standard rollout deletion. When a table contains many record ID (RID) indexes, a significant amount of time is spent removing the index keys that reference the table rows that are being deleted. You can speed up the rollout by specifying that these indexes are to be cleaned up after the deletion operation commits.

To take advantage of AIC for MDC tables, you must explicitly enable the *deferred index cleanup rollout* mechanism. There are two methods of specifying a deferred rollout: setting the **DB2_MDC_ROLLOUT** registry variable to DEFER or issuing the SET CURRENT MDC ROLLOUT MODE statement. During a deferred index cleanup rollout operation, blocks are marked as rolled out without an update to the RID indexes until after the transaction commits. Block identifier (BID) indexes are cleaned up during the delete operation because they do not require row-level processing.

AIC rollout is invoked when a rollout deletion commits or, if the database was shut down, when the table is first accessed following database restart. While AIC is in progress, queries against the indexes are successful, including those that access the index that is being cleaned up.

There is one coordinating cleaner per MDC table. Index cleanup for multiple rollouts is consolidated within the cleaner, which spawns a cleanup agent for each RID index. Cleanup agents update the RID indexes in parallel. Cleaners are also integrated with the utility throttling facility. By default, each cleaner has a utility impact priority of 50 (acceptable values are between 1 and 100, with 0 indicating no throttling). You can change this priority by using the **SET UTIL_IMPACT_PRIORITY** command or the db2UtilityControl API.

Note: In DB2 Version 9.7 and later releases, deferred cleanup rollout is not supported on range-partitioned tables with partitioned RID indexes. Only the NONE and IMMEDIATE modes are supported. The cleanup rollout type is IMMEDIATE if the **DB2_MDC_ROLLOUT** registry variable is set to DEFER, or if the CURRENT MDC ROLLOUT MODE special register is set to DEFERRED to override the **DB2_MDC_ROLLOUT** setting.

If only nonpartitioned RID indexes exist on the table, deferred index cleanup rollout is supported. The MDC block indexes can be partitioned or nonpartitioned.

Monitoring the progress of deferred index cleanup rollout operation

Because the rolled-out blocks on an MDC table are not reusable until after the cleanup is complete, it is useful to monitor the progress of a deferred index cleanup rollout operation. Use the **LIST UTILITIES** command to display a utility monitor entry for each index being cleaned up. You can also retrieve the total

number of MDC table blocks in the database that are pending asynchronous cleanup following a rollout deletion (BLOCKS_PENDING_CLEANUP) by using the ADMIN_GET_TAB_INFO table function or the **GET SNAPSHOT** command.

In the following sample output for the **LIST UTILITIES SHOW DETAIL** command, progress is indicated by the number of pages in each index that have been cleaned up. Each phase represents one RID index.

```
ID = 2
Type = MDC ROLLOUT INDEX CLEANUP
Database Name = WSDB
Partition Number = 0
Description = TABLE.<schema_name>.<table_name>
Start Time = 06/12/2006 08:56:33.390158
State = Executing
Invocation Type = Automatic
Throttling:
    Priority = 50
Progress Monitoring:
    Estimated Percentage Complete = 83
    Phase Number = 1
        Description = <schema_name>.<index_name>
        Total Work = 13 pages
        Completed Work = 13 pages
        Start Time = 06/12/2006 08:56:33.391566
    Phase Number = 2
        Description = <schema_name>.<index_name>
        Total Work = 13 pages
        Completed Work = 13 pages
        Start Time = 06/12/2006 08:56:33.391577
    Phase Number = 3
        Description = <schema_name>.<index_name>
        Total Work = 9 pages
        Completed Work = 3 pages
        Start Time = 06/12/2006 08:56:33.391587
```

Chapter 28. Data compression

You can reduce storage needed for your data by using the compression capabilities built into DB2 for Linux, UNIX, and Windows to reduce the size of tables, indexes and even your backup images.

Tables and indexes often contain repeated information. This repetition can range from individual or combined column values, to common prefixes for column values, or to repeating patterns in XML data. There are a number of compression capabilities that you can use to reduce the amount of space required to store your tables and indexes, along with features you can employ to determine the savings compression can offer.

You can also use backup compression to reduce the size of your backups.²

Compression capabilities included with most editions of DB2 for Linux, UNIX, and Windows include:

- Value compression
- Backup compression.

The following additional compression capabilities are available with the a license for the DB2 Storage Optimization Feature:

- Row compression, including compression for XML storage objects.
- Temporary table compression
- Index compression.

For more details about index compression, see “Index compression” on page 627.

For more details about backup compression, see “Backup compression” on page 630.

Table compression

You can use less disk space for your tables by taking advantage of the DB2 table compression capabilities. Compression saves disk storage space by using fewer database pages to store data.

Also, because you can store more rows per page, fewer pages must be read to access the same amount of data. Therefore, queries on a compressed table need fewer I/O operations to access the same amount of data. Since there are more rows of data on a buffer pool page, the likelihood that needed rows are in the buffer pool increases. For this reason, compression can improve performance through improved buffer pool hit ratios. In a similar way, compression can also speed up backup and restore operations, as fewer pages of need to be transferred to the backup or restore the same amount of data.

You can use compression with both new and existing tables. Temporary tables are also compressed automatically, if the database manager deems it to be advantageous to do so.

2. See “Backup compression” in *Data Recovery and High Availability Guide and Reference* for more information.

There are two main types of data compression available for tables:

- Row compression (available with a license for the DB2 Storage Optimization Feature).
- Value compression

For a particular table, you can use row compression and value compression together or individually. However, you can use only one type of row compression for a particular table.

Value compression

Value compression optimizes space usage for the representation of data, and the storage structures used internally by the database management system to store data. Value compression involves removing duplicate entries for a value, and only storing one copy. The stored copy keeps track of the location of any references to the stored value.

When creating a table, you can use the optional VALUE COMPRESSION clause of the CREATE TABLE statement to specify that the table is to use value compression. You can enable value compression in an existing table with the ACTIVATE VALUE COMPRESSION clause of the ALTER TABLE statement. To disable value compression in a table, you use the DEACTIVATE VALUE COMPRESSION clause of the ALTER TABLE statement.

When VALUE COMPRESSION is used, NULLs and zero-length data that has been assigned to defined variable-length data types (VARCHAR, VARGRAPHICS, LONG VARCHAR, LONG VARGRAPHIC, BLOB, CLOB, and DBCLOB) will not be stored on disk.

If VALUE COMPRESSION is used then the optional COMPRESS SYSTEM DEFAULT option can also be used to further reduce disk space usage. Minimal disk space is used if the inserted or updated value is equal to the system default value for the data type of the column, as the default value will not be stored on disk. Data types that support COMPRESS SYSTEM DEFAULT include all numeric type columns, fixed-length character, and fixed-length graphic string data types. This means that zeros and blanks can be compressed.

When using value compression, the byte count of a compressed column in a row might be larger than that of the uncompressed version of the same column. If your row size approaches the maximum allowed for your page size, you must ensure that sum of the byte counts for compressed and uncompressed columns does not exceed allowable row length of the table in the table space. For example, in a table space with 4 KB page size, the allowable row length is 4005 bytes. If the allowable row length is exceeded, the error message SQL0670N is returned. The formula used to determine the byte counts for compressed and uncompressed columns is documented as part of the CREATE TABLE statement.

If you deactivate value compression:

- COMPRESS SYSTEM DEFAULTS will also be deactivated implicitly, if it had previously been enabled
- The uncompressed columns might cause the row size to exceed the maximum allowed by the current page size of the current table space. If this occurs, the error message SQL0670N will be returned.
- Existing compressed data will remain compressed until the row is updated or you perform a table reorganization with the REORG command.

Row compression

Row compression uses a dictionary-based compression algorithm to replace recurring strings with shorter symbols within data rows.

There are two types of row compression that you can choose from:

- “Classic” row compression.
- Adaptive compression

Row compression is available with a license for the DB2 Storage Optimization Feature. Depending on the DB2 product edition that you have, this feature might be included, or it might be an option that you order separately.

Classic row compression

Classic row compression, sometimes referred to as *static compression*, compresses data rows by replacing patterns of values that repeat across rows with shorter symbol strings.

The benefits of using classic row compression are similar to those of adaptive compression, in that you can store data in less space, which can significantly save storage costs. Unlike adaptive compression, however, classic row compression uses only a table-level dictionary to store globally recurring patterns; it doesn't use the page-level dictionaries that are used to compress data dynamically.

How classic row compression works

Classic row compression uses a table-level compression dictionary to compress data by row. The dictionary is used to map repeated byte patterns from table rows to much smaller symbols; these symbols then replace the longer byte patterns in the table rows. The compression dictionary is stored with the table data rows in the data object portions of the table.

What data gets compressed?

Data that is stored in base table rows and log records is eligible for classic row compression. In addition, the data in XML storage objects is eligible for compression. You can compress LOB data that you place inline in a table row; however, storage objects for long data objects are not compressed.

Restriction: You cannot compress data in XML columns that you created with DB2 Version 9.5 or DB2 Version 9.1. However, you can compress inline XML columns that you add to a table using DB2 Version 9.7 or later, provided the table was created without XML columns in an earlier release of the product. If a table that you created in an earlier release already has one or more XML columns and you want to add a compressed XML column by using DB2 Version 9.7 or later, you must use the ADMIN_MOVE_TABLE stored procedure to migrate the table before you can use compression.

Turning classic row compression on or off

To use classic row compression, you must have a license for the DB2 Storage Optimization Feature. You compress table data by setting the COMPRESS attribute of the table to YES STATIC. You can set this attribute when you create the table by specifying the COMPRESS YES STATIC option for the CREATE TABLE statement. You can also alter an existing table to use compression by using the same option for the ALTER TABLE statement. After you enable compression, operations that

add data to the table, such as an **INSERT**, **LOAD INSERT**, or **IMPORT INSERT** command operation, can use classic row compression. In addition, index compression is enabled for new indexes on the table. Indexes are created as compressed indexes unless you specify otherwise and if they are the types of indexes that can be compressed.

Important: When you enable classic row compression for a table, you enable it for the entire table, even if a table comprises more than one table partition.

To disable compression for a table, use the **ALTER TABLE** statement with the **COMPRESS NO** option; rows that you subsequently add are not compressed. To extract the entire table, you must perform a table reorganization with the **REORG TABLE** command.

If you have a license for the DB2 Storage Optimization Feature, compression for temporary tables is enabled automatically. You cannot enable or disable compression for temporary tables.

Effects of update activity on logs and compressed tables

Depending upon update activity and which columns are updated within a data row, log usage might increase.

If a row increases in size, the new version of the row might not fit on the current data page. Rather, the new image of the row is stored on an overflow page. To minimize the creation of pointer-overflow records, increase the percentage of each page that is to be left as free space after a reorganization by using the **ALTER TABLE** statement with the **PCTFREE** option. For example, if you set the **PCTFREE** option to 5% before you enabled compression, you might change it to 10% when you enable compression. Increasing the percentage of each page to be left as free space is especially important for data that is heavily updated.

Classic row compression for temporary tables

Compression for temporary tables is enabled automatically with the DB2 Storage Optimization Feature. When executing queries, the DB2 optimizer considers the storage savings and the impact on query performance that compression of temporary tables offers to determine whether it is worthwhile to use compression. If it is worthwhile, compression is used automatically. The minimum size that a table must be before compression is used is larger for temporary tables than for regular tables.

You can use the explain facility or the **db2pd** tool to see whether the optimizer used compression for temporary tables.

Reclaiming space that was freed by compression

You can reclaim space that was freed by compressing data. For more information, see “Reclaimable storage” on page 16.

Adaptive compression

Adaptive compression improves upon the compression rates that can be achieved using classic row compression by itself. Adaptive compression incorporates classic row compression; however, it also works on a page-by-page basis to further

compress data. Of the various data compression techniques in the DB2 product, adaptive compression offers the most dramatic possibilities for storage savings.

How adaptive compression works

Adaptive compression actually uses two compression approaches. The first employs the same table-level compression dictionary used in classic row compression to compress data based on repetition within a sampling of data from the table as a whole. The second approach uses a page-level dictionary-based compression algorithm to compress data based on data repetition within each page of data. The dictionaries map repeated byte patterns to much smaller symbols; these symbols then replace the longer byte patterns in the table. The table-level compression dictionary is stored within the table object for which it is created, and is used to compress data throughout the table. The page-level compression dictionary is stored with the data in the data page, and is used to compress only the data within that page. For more information about the role each of these dictionaries in compressing data, see “Compression dictionaries” on page 623.

Note: You can specify that a table be compressed with classic row compression only by using a table-level compression dictionary. However, you cannot specify that tables be compressed by using only page-level compression dictionaries. Adaptive compression uses both table-level and page-level compression dictionaries.

Data that is eligible for compression

Data that is stored *within* data rows, including inlined LOB or XML values, can be compressed with both adaptive and classic row compression. XML storage objects can be compressed using static compression. However storage objects for long data objects that are stored outside table rows is not compressed. In addition, though log records themselves are not compressed, the amount of log data written as a result of insert, update or delete operations is reduced by virtue of the rows being compressed.

Restriction: You cannot compress data in XML columns that you created with DB2 Version 9.5 or DB2 Version 9.1. However, you can compress inline XML columns that you add to a table using DB2 Version 9.7 or later, provided the table was created without XML columns in an earlier release of the product. If a table that you created in an earlier release already has one or more XML columns and you want to add a compressed XML column by using DB2 Version 9.7 or later, you must use the ADMIN_MOVE_TABLE stored procedure to migrate the table before you can use compression.

Turning adaptive compression on or off

To use adaptive compression, you must have a license for the DB2 Storage Optimization Feature. You compress table data by setting the COMPRESS attribute of the table to YES. You can set this attribute when you create the table by specifying the COMPRESS YES option for the CREATE TABLE statement. You can also alter an existing table to use compression by using the same option for the ALTER TABLE statement. After you enable compression, operations that add data to the table, such as an **INSERT**, **LOAD INSERT**, or **IMPORT INSERT** command operation, can use adaptive compression. In addition, index compression is enabled for new indexes on the table. Indexes are created as compressed indexes unless you specify otherwise and if they are the types of indexes that can be compressed.

Important: When you enable adaptive compression for a table, you enable it for the entire table, even if the table comprises more than one table partition.

To disable compression for a table, use the ALTER TABLE statement with the COMPRESS NO option; rows that you later add are not compressed. Existing rows remain compressed. To extract the entire table after you turn off compression, you must perform a table reorganization with the **REORG TABLE** command.

If you apply the licence for the DB2 Storage Optimization Feature, compression for temporary tables is enabled automatically if the database manager deems it valuable. You cannot enable or disable compression for temporary tables.

Effects of update activity on logs and compressed tables

Depending upon update activity and the position of updates in a data row, log usage might increase.

If a row increases in size after adding new data to it, the new version of the row might not fit on the current data page. Rather, the new image of the row is stored on an overflow page. To minimize the creation of pointer-overflow records, increase the percentage of each page that is to be left as free space after a reorganization by using the ALTER TABLE statement with the PCTFREE option. For example, if you set the PCTFREE option to 5% before you enabled compression, you might change it to 10% when you enable compression. Increasing the percentage of each page to be left as free space is especially important for data that is heavily updated.

Compression for temporary tables

Compression for temporary tables is enabled automatically with the DB2 Storage Optimization Feature. Only classic row compression is used for temporary tables.

System temporary tables

When executing queries, the DB2 optimizer considers the storage savings and the impact on query performance that compression of system-created temporary tables offers to determine whether it is worthwhile to use compression. If it is worthwhile, classic row compression is used automatically. The minimum size that a table must be before compression is used is larger for temporary tables than for regular tables.

User-created temporary tables

Created global temporary tables (CGTTs) and declared global temporary tables (DGTTs) are always compressed using classic row compression.

You can use the explain facility or the **db2pd** tool to see whether the optimizer used compression for system temporary tables.

Reclaiming space that was freed by compression

You can reclaim space that has been freed by compressing data. For more information, see “Reclaimable storage” on page 16.

Estimating storage savings offered by adaptive or classic row compression

You can view an estimate of the storage savings adaptive or classic row compression can provide for a table by using the ADMIN_GET_TAB_COMPRESS_INFO table function.

Before you begin

The estimated savings that adaptive or classic row compression offers depend on the statistics generated by running the **RUNSTATS** command. To get the most accurate estimate of the savings that can be achieved, run the **RUNSTATS** command before you perform the following steps.

Procedure

To estimate the storage savings adaptive or classic row compression can offer using the ADMIN_GET_TAB_COMPRESS_INFO table function:

1. Formulate a SELECT statement that uses the ADMIN_GET_TAB_COMPRESS_INFO table function. For example, for a table named SAMPLE1.T1, enter:
`SELECT * FROM TABLE(SYSPROC.ADMIN_GET_TAB_COMPRESS_INFO('SAMPLE1', 'T1'))`
2. Execute the SELECT statement. Executing the statement shown in Step 1 might yield a report like the following:

TABSCHEMA	TABNAME	DBPARTITIONNUM	DATAPARTITIONID	OBJECT_TYPE	ROWCOMPMode	...
SAMPLE1	T1	0	0	DATA	A	...

1 record(s) selected.

PCTPAGESSAVED_CURRENT	AVGROWSIZE_CURRENT	PCTPAGESSAVED_STATIC	...
96	24	81	...

AVGROWSIZE_STATIC	PCTPAGESSAVED_ADAPTIVE	AVGROWSIZE_ADAPTIVE	...
148	93	44	

Creating a table that uses compression

When you create a new table by issuing the CREATE TABLE statement, you have the option to compress the data contained in table rows.

Before you begin

You must decide which type of compression you want to use: adaptive compression, classic row compression, value compression, or a combination of value compression with either of the two types of row compression. Adaptive compression and classic row compression almost always save storage because they attempt to replace data patterns that span multiple columns with shorter symbol strings. Value compression can offer savings if you have many rows with columns that contain the same value, such as a city or country name, or if you have columns that contain the default value for the data type of the column.

Procedure

To create a table that uses compression, issue a CREATE TABLE statement.

- If you want to use adaptive compression, include the COMPRESS YES ADAPTIVE clause.

- If you want to use classic row compression, include the COMPRESS YES STATIC clause.
- If you want to use value compression, include the VALUE COMPRESSION clause. If you want to compress data that represents system default column values, also include the COMPRESS SYSTEM DEFAULT clause.

Results

After you create the table, all data that you add to the table from that point in time on is compressed. Any indexes that are associated with the table are also compressed, unless you specify otherwise by using the COMPRESS NO clause of the CREATE INDEX or ALTER INDEX statements.

Examples

Example 1: The following statement creates a table for customer information with adaptive compression enabled. In this example, the table is compressed by using both table-level and page-level compression dictionaries.

```
CREATE TABLE CUSTOMER
  (CUSTOMERNUM      INTEGER,
   CUSTOMERNAME    VARCHAR(80),
   ADDRESS         VARCHAR(200),
   CITY            VARCHAR(50),
   COUNTRY         VARCHAR(50),
   CODE             VARCHAR(15),
   CUSTOMERNUMDIM  INTEGER)
   COMPRESS YES ADAPTIVE;
```

Example 2: The following statement creates a table for customer information with classic row compression enabled. In this example, the table is compressed by using only a table-level compression dictionary.

```
CREATE TABLE CUSTOMER
  (CUSTOMERNUM      INTEGER,
   CUSTOMERNAME    VARCHAR(80),
   ADDRESS         VARCHAR(200),
   CITY            VARCHAR(50),
   COUNTRY         VARCHAR(50),
   CODE             VARCHAR(15),
   CUSTOMERNUMDIM  INTEGER)
   COMPRESS YES STATIC;
```

Example 3: The following statement creates a table for employee salaries. The SALARY column has a default value of 0, and row compression and system default compression are specified for the column.

```
CREATE TABLE EMPLOYEE_SALARY
  (DEPTNO    CHAR(3)      NOT NULL,
   DEPTNAME  VARCHAR(36)   NOT NULL,
   EMPNO     CHAR(6)      NOT NULL,
   SALARY    DECIMAL(9,2)  NOT NULL WITH DEFAULT COMPRESS SYSTEM DEFAULT)
   COMPRESS YES ADAPTIVE;
```

Note that the VALUE COMPRESSION clause was omitted from this statement. This statement creates a table that is called EMPLOYEE_SALARY; however, a warning message is returned:

```
SQL20140W COMPRESS column attribute ignored because VALUE COMPRESSION is deactivated for the table. SQLSTATE=01648
```

In this case, the COMPRESS SYSTEM DEFAULT clause is not applied to the SALARY column.

Example 4: The following statement creates a table for employee salaries. The SALARY column has a default value of 0, and row compression and system default compression are enabled for the column.

```
CREATE TABLE EMPLOYEE_SALARY
  (DEPTNO  CHAR(3)      NOT NULL,
   DEPTNAME VARCHAR(36)  NOT NULL,
   EMPNO    CHAR(6)      NOT NULL,
   SALARY    DECIMAL(9,2) NOT NULL WITH DEFAULT COMPRESS SYSTEM DEFAULT)
   VALUE COMPRESSION COMPRESS YES ADAPTIVE;
```

In this example, the VALUE COMPRESSION clause is included in the statement, which compresses the default value for the SALARY column.

Enabling compression in an existing table

By using the ALTER TABLE statement, you can modify an existing table to take advantage of the storage-saving benefits of compression.

Before you begin

You must decide which type of compression you want to use: adaptive compression, classic row compression, value compression, or a combination of value compression with either of the two types of row compression. Adaptive compression and classic row compression almost always save storage because they attempt to replace data patterns that span multiple columns with shorter symbol strings. Value compression can offer savings if you have many rows with columns that contain the same value, such as a city or country name, or if you have columns that contain the default value for the data type of the column.

Procedure

To enable compression in an existing table:

1. Issue the ALTER TABLE statement.
 - If you want to use adaptive compression, include the COMPRESS YES ADAPTIVE clause.
 - If you want to use classic row compression, include the COMPRESS YES STATIC clause.
 - If you want to use value compression, include the ACTIVATE VALUE COMPRESSION clause for each column that contains a value you want compressed. If you want to compress data in columns that contain system default values, also include the COMPRESS SYSTEM DEFAULT clause.

All rows that you subsequently append, insert, load, or update use the new compressed format.

2. Optional: To immediately apply compression to all the existing rows of a table, perform a table reorganization by using the REORG TABLE command. If you do not apply compression to all rows at this point, uncompressed rows will not be stored in the new compressed format until the next time that you update them, or the next time the REORG TABLE command runs.

Examples

Example 1: The following statement applies adaptive compression to an existing table that is named CUSTOMER:

```
ALTER TABLE CUSTOMER COMPRESS YES ADAPTIVE
```

Example 2: The following statement applies classic row compression to an existing table that is named CUSTOMER:

```
ALTER TABLE CUSTOMER COMPRESS YES STATIC
```

Example 3: The following statements apply row, value, and system default compression to the SALARY column of an existing table that is named EMPLOYEE_SALARY. The table is then reorganized.

```
ALTER TABLE EMPLOYEE_SALARY  
ALTER SALARY COMPRESS SYSTEM DEFAULT  
COMPRESS YES ACTIVATE VALUE COMPRESSION;  
  
REORG TABLE EMPLOYEE_SALARY
```

Changing or disabling compression for a compressed table

You can change how a table is compressed or disable compression entirely for a table that has adaptive, classic row, or value compression enabled by using one or more of the various compression-related clauses of the ALTER TABLE statement.

About this task

If you deactivate adaptive or classic row compression, index compression is not affected. If you want to uncompress an index, you must use the ALTER INDEX statement.

Procedure

To deactivate compression for a table, or to change from one type of row compression to another:

1. Issue an ALTER TABLE statement.
 - If you want to deactivate adaptive or classic row compression, include the COMPRESS NO clause.
 - If you want to change to a different type of row compression, specify the type of compression you want using the COMPRESS YES ADAPTIVE or COMPRESS YES STATIC clauses. For example, if you have a table that currently uses classic row compression, and you want to change to adaptive compression, execute the ALTER TABLE statement with the COMPRESS YES ADAPTIVE clause
 - If you want to deactivate value compression, include the DEACTIVATE VALUE COMPRESSION clause.
 - If you want to deactivate the compression of system default values, include the COMPRESS OFF option for the ALTER *column name* clause.
2. Perform an offline table reorganization using the **REORG TABLE** command.

Results

- If you turned off row compression using the COMPRESS NO clause, all row data is uncompressed.
- If you changed from one type of row compression to another, the entire table is compressed using the type of row compression you specified in the ALTER TABLE statement. (See Example 2.)
- Deactivating value compression has the following effects:
 - If a table had columns with COMPRESS SYSTEM DEFAULT enabled, compression is no longer enabled for these columns.

- Uncompressed columns might cause the row size to exceed the maximum that the current page size of the current table space allows. If this occurs, error message SQL0670N is returned.

Examples

Example 1: Turning off row compression: The following statements turn off adaptive or classic row compression in a table named CUSTOMER and then reorganizes the table to uncompress that data that was previously compressed:

```
ALTER TABLE CUSTOMER COMPRESS NO
REORG TABLE CUSTOMER
```

Example 2: Changing from static to adaptive compression: Assumes that the SALES table currently uses classic row compression. The following statements change the type of compression used to adaptive compression:

```
ALTER TABLE SALES COMPRESS ADAPTIVE YES
REORG TABLE SALES
```

Compression dictionaries

The database manager creates a table-level compression dictionary for each table that you enable for either adaptive or classic row compression. For tables that you enable for adaptive compression, the database manager also creates page-level compression dictionaries.

Both types of dictionaries are used to map repeated byte patterns from table rows to much smaller symbols; these symbols then replace the longer byte patterns in the table rows.

Table-level compression dictionaries

To build table-level dictionaries, the table is scanned for repeating patterns. Entire rows, not just certain fields or parts of rows, are examined for repeating entries or patterns. After collecting the repetitive entries, the database manager builds a compression dictionary, assigning short, numeric keys to those entries. Generally speaking, text strings provide greater opportunities for compression than numeric data; compressing numeric data involves replacing one number with another. Depending on the size of the numbers being replaced, the storage savings might not be as significant as those achieved by compressing text.

When a table-level dictionary is first created, it is built using a sample of data in the table. The dictionary is not updated again unless you explicitly cause the dictionary to be rebuilt using a classic, offline table reorganization. Even if you rebuild the dictionary, the dictionary reflects only a sample of the data from the entire table.

Remember: The table-level dictionary is static; unless you manually rebuild it, it does not change after it is initially created. Even if you do rebuild it, because of the sampling techniques used to create it, the dictionary might not reflect strings that recur within a single page.

The table-level compression dictionary is stored in hidden rows in the same object that they apply to and is cached in memory for quick access. This dictionary does not occupy much space. Even for extremely large tables, the compression dictionary typically occupies only approximately 100 KB.

Page-level compression dictionaries

Adaptive compression uses page-level dictionaries in addition to table-level dictionaries. However, unlike table-level dictionaries, page-level dictionaries are automatically created or recreated as pages are filled by the database manager. Like table-level compression dictionaries, page-level dictionaries are also stored in hidden rows within the table.

Table-level compression dictionary creation

Table-level compression dictionaries for tables that you enable for adaptive or classic row compression can be built automatically or manually. Tables that you enable for adaptive compression include page-level data dictionaries, which are always automatically created.

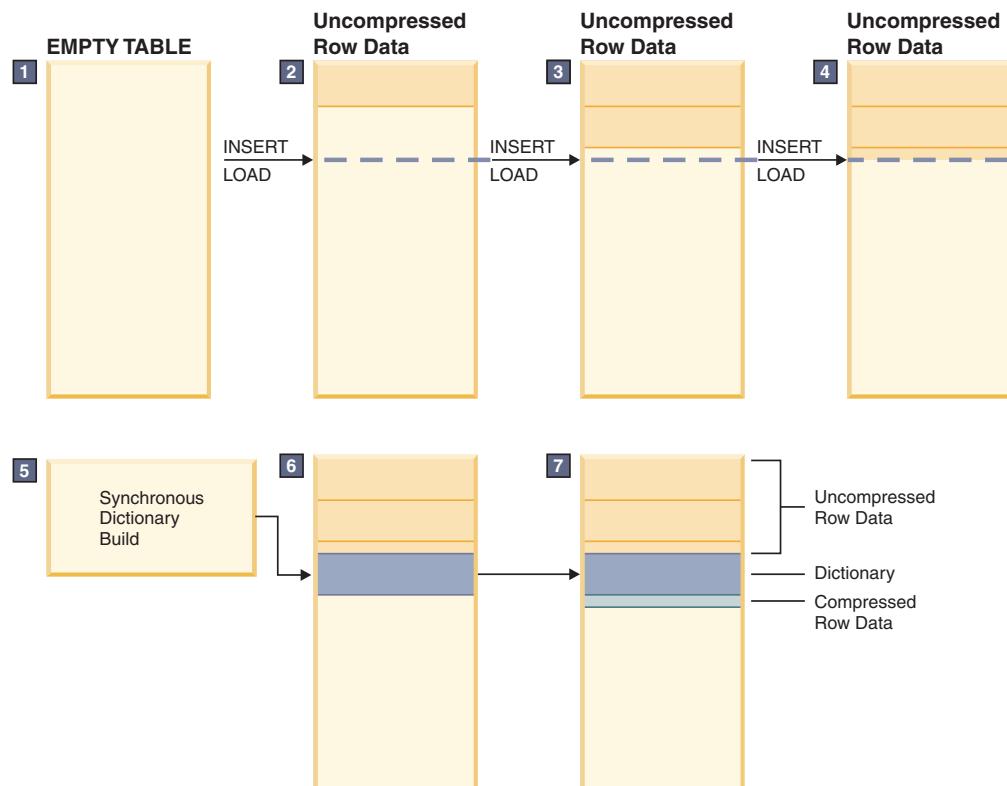
Automatic dictionary creation

Starting with DB2 Version 9.5, a table-level compression dictionary is created automatically if each of the following conditions is met:

- You set the COMPRESS attribute for the table by using the CREATE TABLE or ALTER TABLE statement with the COMPRESS YES ADAPTIVE or COMPRESS YES STATIC clause.
- A table-level compression dictionary does not already exist for the table.
- The table contains sufficient data for constructing a dictionary of repeated data.

Data that you move into the table after the dictionary is created is compressed using the dictionary if compression remains enabled.

The following diagram shows the process by which the compression dictionary is automatically created:



The sequence of events illustrated in the diagram is as follows:

1. A compression dictionary is not yet created, because the table is empty.
2. Data is inserted into the table by using insert or load operations and remains uncompressed.
3. As more data is inserted or loaded into the table, the data remains uncompressed.
4. After a threshold is reached, dictionary creation is triggered automatically if the COMPRESS attribute is set to YES ADAPTIVE or YES STATIC.
5. The dictionary is created.
6. The dictionary is appended to the table.
7. From this point forward, table-level compression is enabled, and the rows that are later inserted or added are compressed by the table-level compression dictionary.

Important: The rows that existed in a table before the dictionary was created remain uncompressed unless you change them or manually rebuild the dictionary.

If you create a table with DB2 Version 9.7 or later and the table contains at least one column of type XML, a second compression dictionary is created. This dictionary is used to compress the XML data that is stored in the default XML storage object that is associated with the table. Compression dictionary creation for XML data occurs automatically if each of the following conditions is met:

- You set the COMPRESS attribute on the table to YES ADAPTIVE or YES STATIC.
- A compression dictionary does not exist within that XML storage object.
- There is sufficient data in the XML storage object.

Restriction: You cannot compress data in XML columns that you created with DB2 Version 9.5 or DB2 Version 9.1. However, you can compress inline XML columns that you add to a table using DB2 Version 9.7 or later, provided the table was created without XML columns in an earlier release of the product. If a table that you created in an earlier release already has one or more XML columns and you want to add a compressed XML column by using DB2 Version 9.7 or later, you must use the ADMIN_MOVE_TABLE stored procedure to migrate the table before you can use compression.

The mechanism for creating table-level compression dictionaries for temporary tables is similar to the mechanism that is used for permanent tables. However, the database manager automatically makes the determination whether to use classic row compression for temporary tables, based on factors such as query complexity and the size of the result set.

Manual dictionary creation

Although dictionaries are created automatically when compression-enabled tables grow to a sufficient size, you can also force a table-level compression dictionary to be created if none exists by using the **REORG TABLE** command with the **RESETDICTIONARY** parameter. This command forces the creation of a compression dictionary if there is at least one row of data in the table. Table reorganization is an offline operation; one benefit of using automatic dictionary creation is that the table remains online as the dictionary is built.

Instead of using the **REORG TABLE** command to force the creation of a new dictionary, you can also use the **INSPECT** command with the **ROWCOMPESTIMATE** parameter. This command creates a compression dictionary if the table does not

already have one. The advantage of this approach over performing a table reorganization is that the table remains online. Rows that you add later are subject to compression; however, rows that existed before you ran the **INSPECT** command remain uncompressed until you perform a table reorganization. However, if compression is enabled, automatic dictionary creation will usually take place shortly after you activate compression, likely before you even have a chance to use the **INSPECT** command.

Resetting compression dictionaries

Whether a table-level compression dictionary is created automatically or manually, the dictionary is static; after it is built, it does not change. As you add or update rows, they are compressed based on the data that exists in the compression dictionary. For many situations, this behavior is appropriate. Consider, for example, a table in a database that is used for maintaining customer accounts for a city water utility. Such a table might have columns such as STREET_ADDRESS, CITY, PROVINCE, TELEPHONE_NUM, POSTAL_CODE, and ACCOUNT_TYPE. If a compression dictionary is built with data from a such table, even if it is only a modestly sized table, there is likely sufficient repetitive information for classic row compression to yield significant space savings. Much of the data might be common from customer to customer, for example, the values of the CITY, POSTAL_CODE, or PROVINCE column or portions of the value in the STREET_ADDRESS or TELEPHONE_NUM column.

However, other tables might change significantly over time. Consider a table that is used for retail sales data as follows:

- A master table is used to accumulate data on a month-by-month basis.
- Each month, a new set of records is loaded into the table.

In this case, a compression dictionary created in, for example, April might not reflect repeating data from sales in later parts of the year. In situations where data in a table changes significantly over time, you might want to reset your compression dictionaries by using the **REORG TABLE** command with the **RESETDICTIONARY** parameter. The advantage of resetting the compression dictionary is that data from the entire table is considered when the dictionary is built.

Impact of classic table reorganization on table-level compression dictionaries

When you reorganize a table that you enabled for adaptive compression or classic row compression using classic, offline table reorganization, you can retain the table-level compression dictionary or force the database manager to create a new one.

In DB2 Version 9.5 and later, a table-level compression dictionary is automatically created for a table that you enable for adaptive or classic row compression by using the CREATE TABLE or ALTER TABLE statement with the **COMPRESS YES** subclause. For a new table, the database manager waits until the table grows to a minimal size before creating the dictionary. For an existing table, the compression dictionary is created when the table grows to a sufficient size to allow pattern repetition to become apparent. Compression is applied only to rows that you insert or update after enabling compression.

If you reorganize a table with a classic table reorganization, and a table-level compression dictionary exists, the **KEEPDICIONARY** parameter of the **REORG TABLE** command is applied implicitly, which retains the dictionary. When you perform the

reorganization, all the rows that are processed are subject to compression using that dictionary. If a compression dictionary does not exist and if the table is large enough, a compression dictionary is created, and the rows are subject to compression using that dictionary.

You can force a new table-level compression dictionary to be built by performing a classic table reorganization that uses the **RESETDICTIONARY** parameter of the **REORG TABLE** command. When you specify the **RESETDICTIONARY** parameter, a new compression dictionary is built if there is at least one row in the table, replacing any existing dictionary.

Note: Table-level dictionaries can be rebuilt using only classic table reorganization. If you attempt to perform an inplace table reorganization of a table that has any rows compressed using a page-level compression dictionary, the REORG command fails with a SQL2219 error.

Multiple compression dictionaries for replication source tables

You can combine the DATA CAPTURE CHANGES clause with the COMPRESS YES STATIC or COMPRESS YES ADAPTIVE option for the CREATE TABLE and ALTER TABLE statements to enable row compression on source tables for replication.

When you enable compression, if you also specify the DATA CAPTURE CHANGES clause as part of the commands **REORG TABLE** or **LOAD REPLACE**, a source table can have two table-level compression dictionaries: an active *table-level compression dictionary* and a *historical compression dictionary*. In other words, if DATA CAPTURE CHANGES is enabled, the table-level compression dictionary is not replaced when you run the **REORG TABLE** or **LOAD REPLACE** commands. Instead, a new dictionary is generated, and the previous dictionary is retained.

The historical compression dictionary makes it possible for the db2ReadLog API to extract the row contents in log records that were written before the active dictionary was rebuilt as a result of specifying the **RESETDICTIONARY** option with a **REORG TABLE** or **LOAD** command.

Note: To have log readers return the data within log records in an uncompressed format instead of a raw compressed format, you must set the **iFilterOption** parameter of the db2ReadLog API to **DB2READLOG_FILTER_ON**.

If you specified the DATA CAPTURE NONE option as part of the CREATE TABLE statement used to create the table, then issuing the **REORG TABLE** command or performing table truncate operations by issuing the **LOAD REPLACE**, **IMPORT REPLACE**, or **TRUNCATE TABLE** command removes the historical compression dictionary for the table.

To see whether there is a historical dictionary present for the table, check the HISTORICAL_DICTIONARY column in the result set of the ADMIN_GET_TAB_DICTIONARY_INFO table function.

Index compression

Indexes, including indexes on declared or created temporary tables, can be compressed in order to reduce storage costs. This is especially useful for large OLTP and data warehouse environments.

By default, index compression is enabled for compressed tables, and disabled for uncompressed tables. You can override this default behavior by using the **COMPRESS YES** option of the CREATE INDEX statement. When working with existing indexes, use the ALTER INDEX statement to enable or disable index compression; you must then perform an index reorganization to rebuild the index.

Restriction: Index compression is not supported for the following types of indexes:

- block indexes
- XML path indexes.

In addition:

- Index specifications cannot be compressed
- Compression attributes for indexes on temporary tables cannot be altered with the ALTER INDEX command.

When index compression is enabled, the on-disk and memory format of index pages are modified based on the compression algorithms chosen by the database manager so as to minimize storage space. The degree of compression achieved will vary based on the type of index you are creating, as well as the data the index contains. For example, the database manager can compress an index with a large number of duplicate keys by storing an abbreviated format of the record identifier (RID) for the duplicate keys. In an index where there is a high degree of commonality in the prefixes of the index keys, the database manager can apply compression based on the similarities in prefixes of index keys.

There can be limitations and trade-offs associated with compression. If the indexes do not share common index column values or partial common prefixes, the benefits of index compression in terms of reduced storage might be negligible. And although a unique index on a timestamp column might have very high compression capabilities due to common values for year, month, day, hour, minute, or even seconds on the same leaf page, examining if common prefixes exist could cause performance to degrade.

If you believe that compression is not offering a benefit in your particular situation, you can either re-create the indexes without compression or alter the indexes and then perform an index reorganization to disable index compression.

There are a few things you should keep in mind when you are considering using index compression:

- If you enable row compression using the **COMPRESS YES** option on the CREATE TABLE or ALTER TABLE command, then by default, compression is enabled for all indexes for which compression is supported that are created after that point for that table, unless explicitly disabled by the CREATE INDEX or ALTER INDEX commands. Similarly, if you disable row compression with the CREATE TABLE or ALTER TABLE command, index compression is disabled for all indexes created after that point for that table unless explicitly enabled by the CREATE INDEX or ALTER INDEX commands.
- If you enable index compression using the ALTER INDEX command, compression will not take place until an index reorganization is performed. Similarly, if you disable compression, the index will remain compressed until you perform an index reorganization.
- During database migration, compression is not enabled for any indexes that might have been migrated. If you want compression to be used, you must use the ALTER INDEX command and then perform an index reorganization.

- CPU usage might increase slightly as a result of the processing required for index compression or decompression. If this is not acceptable, you can disable index compression for new or existing indexes.

Examples

Example 1: Checking whether an index is compressed.

The two statements that follow create a new table T1 that is enabled for row compression, and create an index I1 on T1.

```
CREATE TABLE T1 (C1 INT, C2 INT, C3 INT) COMPRESS YES
CREATE INDEX I1 ON T1(C1)
```

By default, indexes for T1 are compressed. The *compression attribute* for index T1, which shows whether compression is enabled, can be checked by using the catalog table or the admin table function:

```
SELECT COMPRESSION FROM SYSCAT.INDEXES WHERE TABNAME='T1'
```

```
COMPRESSION
-----
Y
```

1 record(s) selected.

Example 2: Determining whether compressed indexes require reorganization.

To see if compressed indexes require reorganization, use the **REORGCHK** command. Figure 75 on page 630 shows the command being run on a table called T1:

```

REORGCHK ON TABLE SCHEMA1.T1
Doing RUNSTATS .....

Table statistics:

F1: 100 * OVERFLOW / CARD < 5
F2: 100 * (Effective Space Utilization of Data Pages) > 70
F3: 100 * (Required Pages / Total Pages) > 80

SCHEMA.NAME      CARD  OV  NP  FP ACTBLK  TSIZE  F1  F2  F3 REORG
-----
Table: SCHEMA1.T1          879   0  14  14      -  51861   0 100 100  ---
-----
Index statistics:

F4: CLUSTERRATIO or normalized CLUSTERFACTOR > 80
F5: 100 * (Space used on leaf pages / Space available on non-empty leaf pages) >
     MIN(50, (100 - PCTFREE))
F6: (100 - PCTFREE) * (Amount of space available in an index with one less level /
     Amount of space required for all keys) < 100
F7: 100 * (Number of pseudo-deleted RIDs / Total number of RIDs) < 20
F8: 100 * (Number of pseudo-empty leaf pages / Total number of leaf pages) < 20

SCHEMA.NAME      INDCARD LEAF ELEAF LVLS NDEL KEYS LEAF_RECSIZE NLEAF_RECSIZE...
-----
Table: SCHEMA1.T1          ...
Index: SCHEMA1.I1          879   15    0    2    0   682           20          20...
-----
...LEAF_PAGE_OVERHEAD NLEAF_PAGE_OVERHEAD PCT_PAGES_SAVED  F4  F5  F6  F7  F8 REORG
...          ...
...          596           596           28  56  31   -   0   0  ----
...          ...

```

Figure 75. Output of REORGCHK command

The output of the **REORGCHK** command has been formatted to fit the page.

Example 3: Determining the potential space savings of index compression.

For an example of how you can calculate potential index compression savings, refer to the documentation for the ADMIN_GET_INDEX_COMPRESS_INFO table function.

Backup compression

In addition to the storage savings you can achieve through row compression in your active database, you can also use backup compression to reduce the size of your database backups.

Whereas row compression works on a table-by-table basis, when you use compression for your backups, *all* of the data in the backup image is compressed, including catalog tables, index objects, LOB objects, auxiliary database files and database meta-data.

You can use backup compression with tables that use row compression. Keep in mind, however, that backup compression requires additional CPU resources and extra time. It may be sufficient to use table compression alone to achieve a

reduction in your backup storage requirements. If you are using row compression, consider using backup compression only if storage optimization is of higher priority than the extra time it takes to perform the backup.

Tip: Consider using backup compression only on table spaces that do not contain compressed data if the following conditions apply:

- Data and index objects are separate from LOB and long field data, and
- You use row and index compression on the majority of your data tables and indexes, respectively

To use compression for your backups, use the **COMPRESS** option on the **BACKUP DATABASE** command.

Chapter 29. Relational indexes

Indexes can be used to improve performance when accessing table data. Relational indexes are used when accessing relational data, and indexes over XML data are used when accessing XML data.

Although the query optimizer decides whether to use a relational index to access relational table data, it is up to you to decide which indexes might improve performance and to create those indexes. The only exceptions to this are the dimension block indexes and the composite block index that are created automatically for each dimension when you create a multidimensional clustering (MDC) table.

Execute the runstats utility to collect new index statistics after you create a relational index or after you change the prefetch size. You should execute the runstats utility at regular intervals to keep the statistics current; without up-to-date statistics about indexes, the optimizer cannot determine the best data-access plan for queries.

To determine whether a relational index is used in a specific package, use the explain facility. To get advice about relational indexes that could be exploited by one or more SQL statements, use the **db2advis** command to launch the Design Advisor.

IBM InfoSphere Optim Query Workload Tuner provides tools for improving the performance of single SQL statements and the performance of groups of SQL statements, which are called query workloads. For more information about this product, see the product overview page at <http://www.ibm.com/software/data/optim/query-workload-tuner-db2-luw/index.html>. In Version 3.1.1 or later, you can also use the Workload Design Advisor to perform many operations that were available in the DB2 Design Advisor wizard. For more information see the documentation for the Workload Design Advisor at <http://publib.boulder.ibm.com/infocenter/dstudio/v3r1/topic/com.ibm.datatools.qrytune.workloadtunedb2luw.doc/topics/genrecsdsgn.html>.

Advantages of a relational index over no index

If no index on a table exists, a table scan must be performed for each table that is referenced in an SQL query. The larger the table, the longer such a scan will take, because a table scan requires that each row be accessed sequentially. Although a table scan might be more efficient for a complex query that requires most of the rows in a table, an index scan can access table rows more efficiently for a query that returns only some table rows.

The optimizer chooses an index scan if the relational index columns are referenced in the SELECT statement and if the optimizer estimates that an index scan will be faster than a table scan. Index files are generally smaller and require less time to read than an entire table, especially when the table is large. Moreover, it might not be necessary to scan an entire index. Any predicates that are applied to the index will reduce the number of rows that must be read from data pages.

If an ordering requirement on the output can be matched with an index column, scanning the index in column order will enable the rows to be retrieved in the

correct order without the need for a sort operation. Note that the existence of a relational index on the table being queried does not guarantee an ordered result set. Only an ORDER BY clause ensures the order of a result set.

A relational index can also contain include columns, which are non-indexed columns in an indexed row. Such columns can make it possible for the optimizer to retrieve required information from the index alone, without having to access the table itself.

Disadvantages of a relational index over no index

Although indexes can reduce access time significantly, they can also have adverse effects on performance. Before you create indexes, consider the effects of multiple indexes on disk space and processing time. Choose indexes carefully to address the needs of your application programs.

- Each index requires storage space. The exact amount depends on the size of the table and the size and number of columns in the relational index.
- Each insert or delete operation against a table requires additional updating of each index on that table. This is also true for each update operation that changes the value of an index key.
- Each relational index represents another potential access plan for the optimizer to consider, which increases query compilation time.

Indexes on partitioned tables

Indexes on partitioned tables operate similarly to indexes on nonpartitioned tables. However, indexes on partitioned tables are stored using a different storage model, depending on whether the indexes are partitioned or nonpartitioned.

Although the indexes for a regular nonpartitioned table all reside in a shared index object, a *nonpartitioned index* on a partitioned table is created in its own index object in a single table space, even if the data partitions span multiple table spaces. Both database managed space (DMS) and system managed space (SMS) table spaces support the use of indexes in a different location than the table data. Each nonpartitioned index can be placed in its own table space, including large table spaces. Each index table space must use the same storage mechanism as the data partitions, either DMS or SMS. Indexes in large table spaces can contain up to 2^{29} pages. All of the table spaces must be in the same database partition group.

A *partitioned index* uses an index organization scheme in which index data is divided across multiple *index partitions*, according to the partitioning scheme of the table. Each index partition refers only to table rows in the corresponding data partition. All index partitions for a specific data partition reside in the same index object.

Starting in DB2 Version 9.7 Fix Pack 1, user-created indexes over XML data on XML columns in partitioned tables can be either partitioned or nonpartitioned. The default is partitioned. System-generated XML region indexes are always partitioned, and system-generated column path indexes are always nonpartitioned. In DB2 V9.7, indexes over XML data are nonpartitioned.

Benefits of a nonpartitioned index include:

- The fact that indexes can be reorganized independently of one another
- Improved performance of drop index operations

- The fact that when individual indexes are dropped, space becomes immediately available to the system without the need for index reorganization

Benefits of a partitioned index include:

- Improved data roll-in and roll-out performance
- Less contention on index pages, because the index is partitioned
- An index B-tree structure for each index partition, which can result in the following benefits:
 - Improved insert, update, delete, and scan performance because the B-tree for an index partition normally contains fewer levels than an index that references all data in the table
 - Improved scan performance and concurrency when partition elimination is in effect. Although partition elimination can be used for both partitioned and nonpartitioned index scans, it is more effective for partitioned index scans because each index partition contains keys for only the corresponding data partition. This configuration can result in having to scan fewer keys and fewer index pages than a similar query over a nonpartitioned index.

Although a nonpartitioned index always preserves order on the index columns, a partitioned index might lose some order across partitions in certain scenarios; for example, if the partitioning columns do not match the index columns, and more than one partition is to be accessed.

During online index creation, concurrent read and write access to the table is permitted. After an online index is built, changes that were made to the table during index creation are applied to the new index. Write access to the table is blocked until index creation completes and the transaction commits. For partitioned indexes, each data partition is quiesced to read-only access *only* while changes that were made to that data partition (during the creation of the index partition) are applied.

Partitioned index support becomes particularly beneficial when you are rolling data in using the ALTER TABLE...ATTACH PARTITION statement. If nonpartitioned indexes exist (not including the XML columns path index, if the table has XML data), issue a SET INTEGRITY statement after partition attachment. This statement is necessary for nonpartitioned index maintenance, range validation, constraints checking, and materialized query table (MQT) maintenance. Nonpartitioned index maintenance can be time-consuming and require large amounts of log space. Use partitioned indexes to avoid this maintenance cost.

If there are nonpartitioned indexes (except XML columns path indexes) on the table to maintain after an attach operation, the SET INTEGRITY...ALL IMMEDIATE UNCHECKED statement behaves as though it were a SET INTEGRITY...IMMEDIATE CHECKED statement. All integrity processing, nonpartitioned index maintenance, and table state transitions are performed as though a SET INTEGRITY...IMMEDIATE CHECKED statement was issued.

The Figure 76 on page 636 diagram shows two nonpartitioned indexes on a partitioned table, with each index in a separate table space.

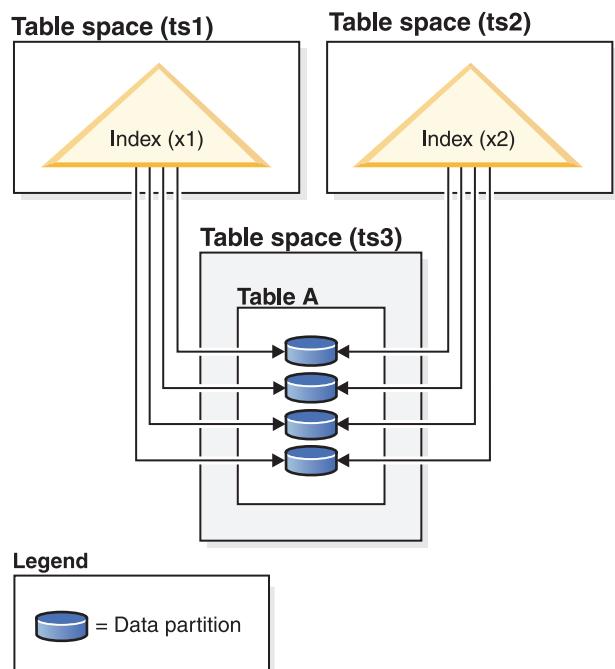
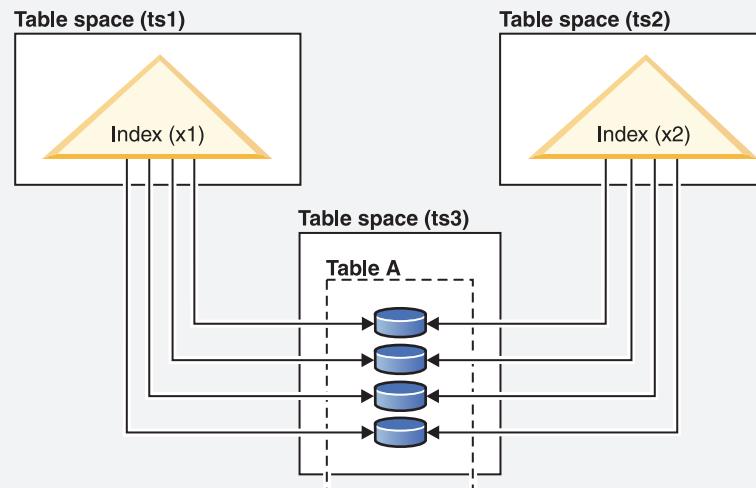


Figure 76. Nonpartitioned indexes on a partitioned table

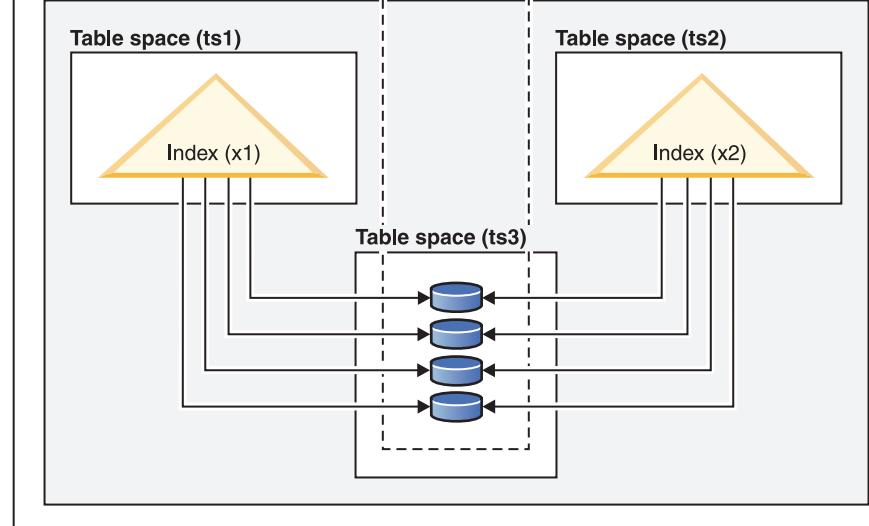
The Figure 77 on page 637 diagram shows a partitioned index on a partitioned table that spans two database partitions and resides in a single table space.

Database partition group (dbgroup1)

Database partition (dbpart1)



Database partition (dbpart2)



Legend



Figure 77. Nonpartitioned index on a table that is both distributed and partitioned

The Figure 78 on page 638 diagram shows a mix of partitioned and nonpartitioned indexes on a partitioned table.

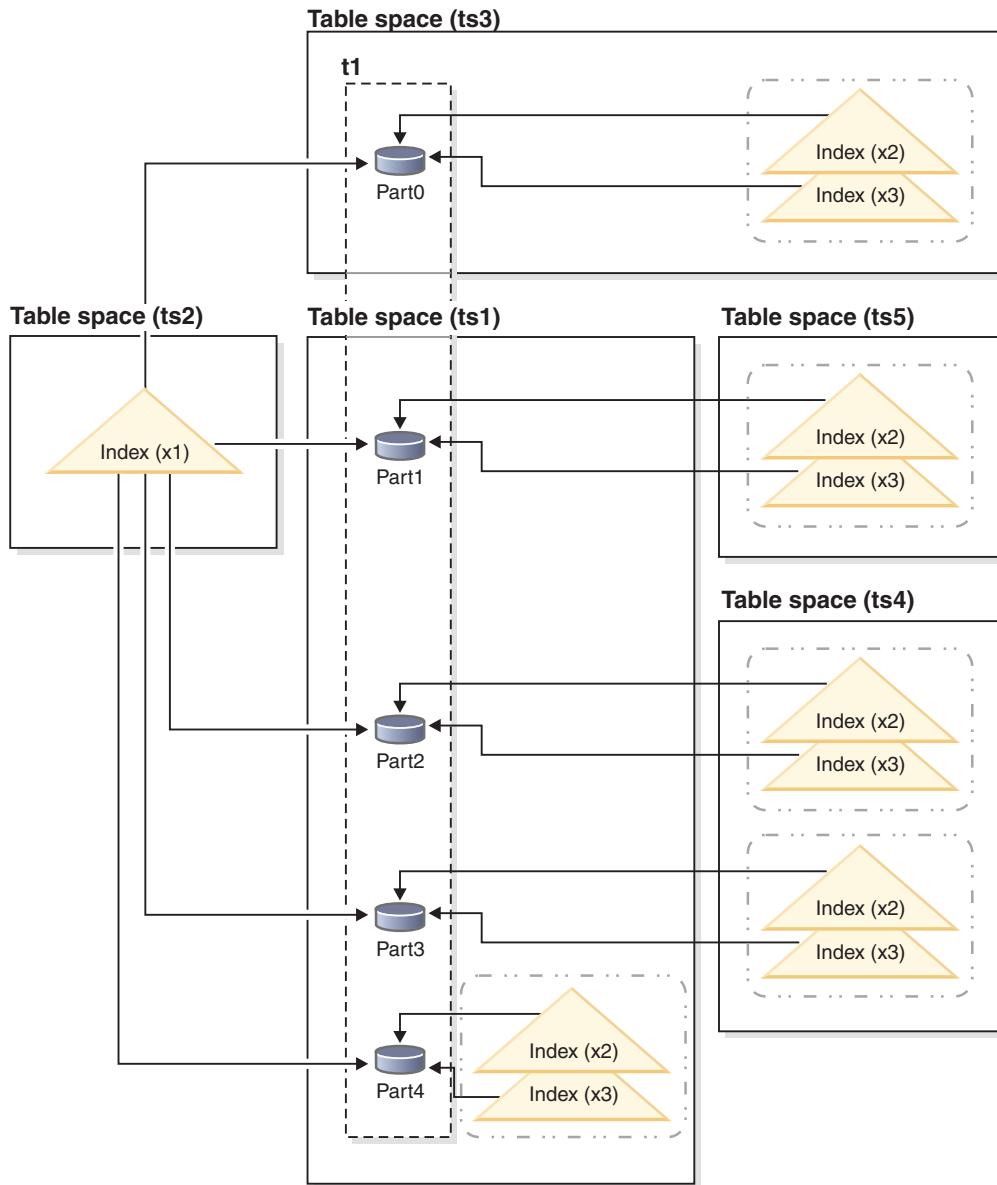


Figure 78. Partitioned and nonpartitioned indexes on a partitioned table

The nonpartitioned index *X1* refers to rows in all of the data partitions. By contrast, the partitioned indexes *X2* and *X3* refer only to rows in the data partition with which they are associated. Table space *TS3* also shows the index partitions sharing the table space of the data partitions with which they are associated. This configuration is the default for partitioned indexes.

You can override the default location for nonpartitioned and partitioned indexes, although the way that you do this is different for each. With nonpartitioned indexes, you can specify a table space when you create the index; for partitioned indexes, you need to determine the table spaces in which the index partitions are stored when you create the table.

Nonpartitioned indexes

To override the index location for nonpartitioned indexes, use the *IN* clause on the *CREATE INDEX* statement to specify an alternative table space location for the index. You can place different indexes in different

table spaces, as required. If you create a partitioned table without specifying where to place its nonpartitioned indexes, and you then create an index by using a CREATE INDEX statement that does not specify a table space, the index is created in the table space of the first attached or visible data partition. Each of the following three possible cases is evaluated in order, starting with case 1, to determine where the index is to be created. This evaluation to determine table space placement for the index stops when a matching case is found.

Case 1:

When an index table space is specified in the CREATE INDEX...IN *tbspace* statement, use the specified table space for this index.

Case 2:

When an index table space is specified in the CREATE TABLE...INDEX IN *tbspace* statement, use the specified table space for this index.

Case 3:

When no table space is specified, choose the table space that is used by the first attached or visible data partition.

Partitioned indexes

By default, index partitions are placed in the same table space as the data partitions that they reference. To override this default behavior, you must use the INDEX IN clause for each data partition that you define by using the CREATE TABLE statement. In other words, if you plan to use partitioned indexes for a partitioned table, you must anticipate where you want the index partitions to be stored when you create the table. If you try to use the INDEX IN clause when creating a partitioned index, you receive an error message.

Example 1: Given partitioned table SALES (a int, b int, c int), create a unique index A_IDX.

```
create unique index a_idx on sales (a)
```

Because the table SALES is partitioned, index a_idx is also created as a partitioned index.

Example 2: Create index B_IDX.

```
create index b_idx on sales (b)
```

Example 3: To override the default location for the index partitions in a partitioned index, use the INDEX IN clause for each partition that you define when creating the partitioned table. In the example that follows, indexes for the table Z are created in table space TS3.

```
create table z (a int, b int)
    partition by range (a) (starting from (1)
                           ending at (100)  index in ts3)

create index c_idx on z (a) partitioned
```

Relational index planning tips

A well-designed index can make it easier for queries to access relational data.

Use the Design Advisor (**db2advis** command) to find the best indexes for a specific query or for the set of queries that defines a workload. This tool can make performance-enhancing recommendations, such as include columns or indexes that are enabled for reverse scans.

The following guidelines can also help you to create useful relational indexes.

- Retrieving data efficiently
 - To improve data retrieval, add *include columns* to unique indexes. Good candidates are columns that:
 - Are accessed frequently and would benefit from index-only access
 - Are not required to limit the range of index scans
 - Do not affect the ordering or uniqueness of the index key

For example:

```
create unique index idx on employee (workdept) include (lastname)
```

Specifying LASTNAME as an include column rather than part of the index key means that LASTNAME is stored only on the leaf pages of the index.

- Create relational indexes on columns that are used in the WHERE clauses of frequently run queries.

In the following example, the WHERE clause will likely benefit from an index on WORKDEPT, unless the WORKDEPT column contains many duplicate values.

```
where workdept='A01' or workdept='E21'
```

- Create relational indexes with a compound key that names each column referenced in a query. When an index is specified in this way, relational data can be retrieved from the index only, which is more efficient than accessing the table.

For example, consider the following query:

```
select lastname  
      from employee  
     where workdept in ('A00','D11','D21')
```

If a relational index is defined on the WORKDEPT and LASTNAME columns of the EMPLOYEE table, the query might be processed more efficiently by scanning the index rather than the entire table. Because the predicate references WORKDEPT, this column should be the first key column of the relational index.

- Searching tables efficiently

Decide between ascending and descending key order, depending on the order that will be used most often. Although values can be searched in reverse direction if you specify the ALLOW REVERSE SCANS option on the CREATE INDEX statement, scans in the specified index order perform slightly better than reverse scans.

- Accessing larger tables efficiently

Use relational indexes to optimize frequent queries against tables with more than a few data pages, as recorded in the NPAGES column of the SYSCAT.TABLES catalog view. You should:

- Create an index on any column that you will use to join tables.
- Create an index on any column that you will be searching for specific values on a regular basis.
- Improving the performance of update or delete operations

- To improve the performance of such operations against a parent table, create relational indexes on foreign keys.
- To improve the performance of such operations against REFRESH IMMEDIATE and INCREMENTAL materialized query tables (MQTs), create unique relational indexes on the implied unique key of the MQT, which is composed of the columns in the GROUP BY clause of the MQT definition.
- Improving join performance

If you have more than one choice for the first key column in a multiple-column relational index, use the column that is most often specified with an equijoin predicate (*expression1 = expression2*) or the column with the greatest number of distinct values as the first key column.

- Sorting

- For fast sort operations, create relational indexes on columns that are frequently used to sort the relational data.
- To avoid some sorts, use the CREATE INDEX statement to define primary keys and unique keys whenever possible.
- Create a relational index to order the rows in whatever sequence is required by a frequently run query. Ordering is required by the DISTINCT, GROUP BY, and ORDER BY clauses.

The following example uses the DISTINCT clause:

```
select distinct workdept
      from employee
```

The database manager can use an index that is defined on the WORKDEPT column to eliminate duplicate values. The same index could also be used to group values, as in the following example that uses a GROUP BY clause:

```
select workdept, average(salary)
      from employee
     group by workdept
```

- Keeping newly inserted rows clustered and avoiding page splits

Define a clustering index, which should significantly reduce the need to reorganize the table. Use the PCTFREE option on the CREATE TABLE statement to specify how much free space should be left on each page so that rows can be inserted appropriately. You can also specify the pagefreespace file type modifier on the **LOAD** command.

- Saving index maintenance costs and storage space

- Avoid creating indexes that are partial keys of other existing indexes. For example, if there is an index on columns A, B, and C, another index on columns A and B is generally not useful.
- Do not create arbitrary indexes on many columns. Unnecessary indexes not only waste space, but also cause lengthy prepare times.
 - For online transaction processing (OLTP) environments, create one or two indexes per table.
 - For read-only query environments, you might create more than five indexes per table.

Note: For workloads involving many ad-hoc queries with many different predicates where index gap cardinality is small and the selectivity of non index gaps after the index gap is small, it is likely more appropriate to create a few large composite indexes for a table, as opposed to many smaller indexes.

- For mixed query and OLTP environments, between two and five indexes per table is likely appropriate.
- Enabling online index defragmentation

Use the MINPCTUSED option when you create relational indexes.

MINPCTUSED enables online index defragmentation; it specifies the minimum amount of space that must be in use on an index leaf page.

Relational index performance tips

There are a number of actions that you can take to ensure that your relational indexes perform well.

- Specify a large utility heap

If you expect a lot of update activity against the table on which a relational index is being created or reorganized, consider configuring a large utility heap (**util_heap_sz** database configuration parameter), which will help to speed up these operations.

- To avoid sort overflows in a symmetric multiprocessor (SMP) environment, increase the value of the **sheapthres** database manager configuration parameter
- Create separate table spaces for relational indexes

You can create index table spaces on faster physical devices, or assign index table spaces to a different buffer pool, which might keep the index pages in the buffer longer because they do not compete with data pages.

If you use a different table space for indexes, you can optimize the configuration of that table space for indexes. Because indexes are usually smaller than tables and are spread over fewer containers, indexes often have smaller extent sizes. The query optimizer considers the speed of the device that contains a table space when it chooses an access plan.

- Ensure a high degree of clustering

If your SQL statement requires ordering of the result (for example, if it contains an ORDER BY, GROUP BY, or DISTINCT clause), the optimizer might not choose an available index if:

- Index clustering is poor. For information about the degree of clustering in a specific index, query the CLUSTERRATIO and CLUSTERFACTOR columns of the SYSCAT.INDEXES catalog view.
- The table is so small that it is cheaper to scan the table and to sort the result set in memory.
- There are competing indexes for accessing the table.

A clustering index attempts to maintain a particular order of the data, improving the CLUSTERRATIO or CLUSTERFACTOR statistics that are collected by the runstats utility. After you create a clustering index, perform an offline table reorg operation. In general, a table can only be clustered on one index. Build additional indexes after you build the clustering index.

A table's PCTFREE value determines the amount of space on a page that is to remain empty for future data insertions, so that this inserted data can be clustered appropriately. If you do not specify a PCTFREE value for a table, reorganization eliminates all extra space.

Except in the case of range-clustered tables, data clustering is not maintained during update operations. That is, if you update a record so that its key value in the clustering index changes, the record is not necessarily moved to a new page to maintain the clustering order. To maintain clustering, delete the record and then insert an updated version of the record, instead of using an update operation.

- Keep table and index statistics up-to-date

After you create a new relational index, execute the runstats utility to collect index statistics. These statistics help the optimizer to determine whether using the index can improve data-access performance.

- Enable online index defragmentation

Online index defragmentation is enabled if MINPCTUSED for the relational index is set to a value that is greater than zero. Online index defragmentation enables indexes to be compacted through the merging of index leaf pages when the amount of free space on a page falls below the specified MINPCTUSED value.

- Reorganize relational indexes as necessary

To get the best performance from your indexes, consider reorganizing them periodically, because updates to tables can cause index page prefetching to become less effective.

To reorganize an index, either drop it and recreate it, or use the reorg utility.

To reduce the need for frequent reorganization, specify an appropriate PCTFREE value on the CREATE INDEX statement to leave sufficient free space on each index leaf page as it is being created. During future activity, records can be inserted into the index with less likelihood of index page splitting, which decreases page contiguity and, therefore, the efficiency of index page prefetching. The PCTFREE value that is specified when you create a relational index is preserved when the index is reorganized.

- Analyze explain information about relational index use

Periodically issue EXPLAIN statements against your most frequently used queries and verify that each of your relational indexes is being used at least once. If an index is not being used by any query, consider dropping that index.

Explain information also lets you determine whether a large table being scanned is processed as the inner table of a nested-loop join. If it is, an index on the join-predicate column is either missing or considered to be ineffective for applying the join predicate.

- Declare tables that vary widely in size as “volatile”

A *volatile table* is a table whose cardinality at run time can vary greatly. For this kind of table, the optimizer might generate an access plan that favors a table scan instead of an index scan.

Use the ALTER TABLE statement with the VOLATILE clause to declare such a table as volatile. The optimizer will use an index scan instead of a table scan against such tables, regardless of statistics, if:

- All referenced columns are part of the index
- The index can apply a predicate during the index scan

In the case of typed tables, the ALTER TABLE...VOLATILE statement is supported only for the root table of a typed table hierarchy.

Online index defragmentation

Online index defragmentation is enabled by the user-definable threshold, MINPCTUSED, for the minimum amount of used space on an index leaf page.

When an index key is deleted from a leaf page and this threshold is exceeded, the neighboring index leaf pages are checked to determine whether two leaf pages can be merged. If there is sufficient space on a page, and the merging of two neighboring pages is possible, the merge occurs immediately. The resulting empty index leaf page is then deleted.

The MINPCTUSED clause cannot be altered by the ALTER INDEX statement. If existing indexes require the ability to be merged via online index defragmentation, they must be dropped and then recreated with the CREATE INDEX statement specifying the MINPCTUSED clause. When enabling online index defragmentation, to increase the likelihood that pages can be merged when neighboring pages are checked, MINPCTUSED should be set to a value less than 50. A value of zero, which is the default, disables online defragmentation. Whether MINPCTFREE is set or not, the ability to perform a REORG CLEANUP on that index is not affected. Setting MINPCTFREE to a low value, 1-50, might reduce the work left for REORG CLEANUP to do as more page merging is performed automatically at run time.

Index nonleaf pages are not merged during online index defragmentation. However, empty nonleaf pages are deleted and made available for reuse by other indexes on the same table. To free deleted pages for other objects in a database managed space (DMS) storage model there are two reorganization options, REBUILD or RECLAIM EXTENTS. For system managed space (SMS) storage model only REORG REBUILD is allowed. RECLAIM EXTENTS moves pages to create full extents of deleted pages and then frees them. REBUILD rebuilds the index from scratch making the index as small as possible respecting PCTFREE.

Only REBUILD addresses the number of levels in an index. If reducing the number of levels in an index is a concern perform a reorganization with the REBUILD option.

When there is an X lock on a table, keys are physically removed from a page during key deletion. In this case, online index defragmentation is effective. However, if there is no X lock on the table during key deletion, keys are marked deleted but are not physically removed from the index page, and index defragmentation is not attempted.

To defragment indexes regardless of the value of MINPCTUSED, invoke the REORG INDEXES command with the CLEANUP ALL option. The whole index is checked, and whenever possible two neighboring leaf pages are merged. This merge is possible if at least PCTFREE free space is left on the merged page. PCTFREE can be specified at index creation time; the default value is 10 (percent).

Chapter 30. Parallel processing for applications

The DB2 product supports parallel environments, specifically on symmetric multiprocessor (SMP) machines.

In SMP machines, more than one processor can access the database, allowing the execution of complex SQL requests to be divided among the processors. This *intra-partition parallelism* is the subdivision of a single database operation (for example, index creation) into multiple parts, which are then executed in parallel within a single database partition.

To specify the degree of parallelism when you compile an application, use the CURRENT DEGREE special register, or the DEGREE bind option. *Degree* refers to the number of query parts that can execute concurrently. There is no strict relationship between the number of processors and the value that you select for the degree of parallelism. You can specify a value that is more or less than the number of processors on the machine. Even for uniprocessor machines, you can set the degree to be higher than one to improve performance. Note, however, that each degree of parallelism adds to the system memory and processor overhead.

You can also specify the degree of parallelism for workloads using the MAXIMUM DEGREE workload attribute. In the affected workload, values set using MAXIMUM DEGREE will override values assigned by the CURRENT DEGREE special register, or the DEGREE bind option.

Some configuration parameters must be modified to optimize performance when you use parallel execution of queries. In an environment with a high degree of parallelism, you should review and modify configuration parameters that control the amount of shared memory and prefetching.

The following configuration parameters control and manage parallel processing.

- The **intra_parallel** database manager configuration parameter enables or disables parallelism.
- The **max_querydegree** database manager configuration parameter sets an upper limit on the degree of parallelism for any query in the database. This value overrides the CURRENT DEGREE special register and the DEGREE bind option.
- The **dft_degree** database configuration parameter sets the default value for the CURRENT DEGREE special register and the DEGREE bind option.

To enable or disable intra-partition parallelism from within a database application, you can call the ADMIN_SET_INTRA_PARALLEL procedure. Setting ADMIN_SET_INTRA_PARALLEL will apply intra-partition parallelism only to your application. For your application, this value will override the **intra_parallel** database manager configuration parameter.

To enable or disable intra-partition parallelism from within a workload, you can set the MAXIMUM DEGREE workload attribute. This will apply intra-partition parallelism only to your workload. This value will override both the the **intra_parallel** database manager configuration parameter and any values assigned by the ADMIN_SET_INTRA_PARALLEL procedure.

If a query is compiled with DEGREE = ANY, the database manager chooses the degree of intra-partition parallelism on the basis of a number of factors, including the number of processors and the characteristics of the query. The actual degree used at run time might be lower than the number of processors, depending on these factors and the amount of activity on the system. The degree of parallelism might be reduced before query execution if the system is busy.

Use the DB2 explain facility to display information about the degree of parallelism chosen by the optimizer. Use the database system monitor to display information about the degree of parallelism actually being used at run time.

Parallelism in non-SMP environments

You can specify a degree of parallelism without having an SMP machine. For example, I/O-bound queries on a uniprocessor machine might benefit from declaring a degree of 2 or more. In this case, the processor might not have to wait for I/O tasks to complete before starting to process a new query. Utilities such as load can control I/O parallelism independently.

Intrapartition parallelism improvements

One goal of the DB2 query optimizer is to choose parallel execution strategies that maintain data balance among subagents and keep them equally busy. In this release, the parallelization capabilities of the optimizer have been further enhanced to enable more workloads to better use multi-core processors.

Rebalancing imbalanced subagent workloads

Data filtering and data skew can cause workloads between subagents to become imbalanced while a query executes. The inefficiency of imbalanced workloads is magnified by joins and other computationally expensive operations. The optimizer looks for sources of imbalance in the query's access plan and applies a balancing strategy, ensuring that work is evenly divided between the subagents. For an unordered outer data stream, the optimizer balances the join using the REBAL operator on the outer. For an ordered data stream (where ordered data is produced by an index access or a sort), the optimizer balances the data using a shared sort. A shared sort will not be used if the sort overflows into the temporary tables, due to the high cost of a sort overflow.

Parallel scans on range partitioned tables and indexes

Parallel table scans can be run against range partitioned tables, and similarly, parallel index scans can be run against partitioned indexes. For a parallel scan, partitioned indexes are divided into ranges of records, based on index key values and the number of key entries for a key value. When a parallel scan begins, subagents are assigned a range of records, and once the subagent completes a range, it is assigned a new range. The index partitions are scanned sequentially with subagents potentially scanning unreserved index partitions at any point in time without waiting for each other. Only the subset of index partitions that is relevant to the query based on data partition elimination analysis is scanned.

Ability to throttle the degree of parallelism to optimize for transactional workloads

Individual applications or workloads can now dynamically throttle the degree of Intrapartition parallelism to optimize performance for the types of queries being

executed. In previous versions of DB2, it was only possible control the degree of parallelism (and whether it was turned on or off) for the whole instance. Turning parallelism on or off also required the instance to be restarted. On database servers with mixed workloads, a more flexible approach to controlling Intrapartition parallelism is needed. Transactional workloads, which typically include short insert, update, and delete transactions, do not benefit from parallelization. There is some processing overhead when Intrapartition parallelism is enabled, which introduces a negative impact to transactional workloads. However; data warehouse workloads benefit greatly from parallelization as they typically include processor-intensive long-running queries.

For mixed workloads, with transactional and data warehousing components, you can now configure the database system to provide parallelism settings that are optimal for the kind of workload deployed by each application. You can either control the parallelism settings through application logic, or through the DB2 workload manager (which does not require application changes).

Controlling Intrapartition parallelism from database applications: To enable or disable Intrapartition parallelism from within a database application, you can call the new ADMIN_SET_INTRA_PARALLEL procedure. For example, the following statement enables Intrapartition parallelism:

```
CALL ADMIN_SET_INTRA_PARALLEL('YES')
```

Although the procedure is called in the current transaction, it takes effect starting with the following transaction, and is only applicable to the calling application. The setting for Intrapartition parallelism set by ADMIN_SET_INTRA_PARALLEL will override whatever value is in the **intra_parallel** configuration parameter.

Controlling Intrapartition parallelism from the DB2 workload manager: To enable or disable Intrapartition parallelism for a specified workload, you can set the MAXIMUM DEGREE workload attribute. For example, the following statement disables Intrapartition parallelism for a workload called trans:

```
ALTER WORKLOAD trans MAXIMUM DEGREE 1
```

All statements in the workload executed after the ALTER WORKLOAD statement will be run with Intrapartition parallelism turned off. The setting for Intrapartition parallelism set with the MAXIMUM DEGREE workload attribute overrides calls to ADMIN_SET_INTRA_PARALLEL, and will override whatever value is in the **intra_parallel** configuration parameter.

Optimization strategies for intra-partition parallelism

The optimizer can choose an access plan to execute a query in parallel within a single database partition if a degree of parallelism is specified when the SQL statement is compiled.

At run time, multiple database agents called subagents are created to execute the query. The number of subagents is less than or equal to the degree of parallelism that was specified when the SQL statement was compiled.

To parallelize an access plan, the optimizer divides it into a portion that is run by each subagent and a portion that is run by the coordinating agent. The subagents pass data through table queues to the coordinating agent or to other subagents. In a partitioned database environment, subagents can send or receive data through table queues from subagents in other database partitions.

Intra-partition parallel scan strategies

Relational scans and index scans can be performed in parallel on the same table or index. For parallel relational scans, the table is divided into ranges of pages or rows, which are assigned to subagents. A subagent scans its assigned range and is assigned another range when it has completed work on the current range.

For parallel index scans, the index is divided into ranges of records based on index key values and the number of index entries for a key value. The parallel index scan proceeds like a parallel table scan, with subagents being assigned a range of records. A subagent is assigned a new range when it has completed work on the current range.

Parallel table scans can be run against range partitioned tables, and similarly, parallel index scans can be run against partitioned indexes. For a parallel scan, partitioned indexes are divided into ranges of records, based on index key values and the number of key entries for a key value. When a parallel scan begins, subagents are assigned a range of records, and once the subagent completes a range, it is assigned a new range. The index partitions are scanned sequentially with subagents potentially scanning unreserved index partitions at any point in time without waiting for each other. Only the subset of index partitions that is relevant to the query based on data partition elimination analysis is scanned.

The optimizer determines the scan unit (either a page or a row) and the scan granularity.

Parallel scans provide an even distribution of work among the subagents. The goal of a parallel scan is to balance the load among the subagents and to keep them equally busy. If the number of busy subagents equals the number of available processors, and the disks are not overworked with I/O requests, the machine resources are being used effectively.

Other access plan strategies might cause data imbalance as the query executes. The optimizer chooses parallel strategies that maintain data balance among subagents.

Intra-partition parallel sort strategies

The optimizer can choose one of the following parallel sort strategies:

- Round-robin sort

This is also known as a *redistribution sort*. This method uses shared memory to efficiently redistribute the data as evenly as possible to all subagents. It uses a round-robin algorithm to provide the even distribution. It first creates an individual sort for each subagent. During the insert phase, subagents insert into each of the individual sorts in a round-robin fashion to achieve a more even distribution of data.

- Partitioned sort

This is similar to the round-robin sort in that a sort is created for each subagent. The subagents apply a hash function to the sort columns to determine into which sort a row should be inserted. For example, if the inner and outer tables of a merge join are a partitioned sort, a subagent can use merge join to join the corresponding table portions and execute in parallel.

- Replicated sort

This sort is used if each subagent requires all of the sort output. One sort is created and subagents are synchronized as rows are inserted into the sort. When

the sort is complete, each subagent reads the entire sort. If the number of rows is small, this sort can be used to rebalance the data stream.

- Shared sort

This sort is the same as a replicated sort, except that subagents open a parallel scan on the sorted result to distribute the data among the subagents in a way that is similar to a round-robin sort.

Intra-partition parallel temporary tables

Subagents can cooperate to produce a temporary table by inserting rows into the same table. This is called a *shared temporary table*. The subagents can open private scans or parallel scans on the shared temporary table, depending on whether the data stream is to be replicated or split.

Intra-partition parallel aggregation strategies

Aggregation operations can be performed by subagents in parallel. An aggregation operation requires the data to be ordered on the grouping columns. If a subagent can be guaranteed to receive all the rows for a set of grouping column values, it can perform a complete aggregation. This can happen if the stream is already split on the grouping columns because of a previous partitioned sort.

Otherwise, the subagent can perform a partial aggregation and use another strategy to complete the aggregation. Some of these strategies are:

- Send the partially aggregated data to the coordinator agent through a merging table queue. The coordinator agent completes the aggregation.
- Insert the partially aggregated data into a partitioned sort. The sort is split on the grouping columns and guarantees that all rows for a set of grouping columns are contained in one sort partition.
- If the stream needs to be replicated to balance processing, the partially aggregated data can be inserted into a replicated sort. Each subagent completes the aggregation using the replicated sort, and receives an identical copy of the aggregation result.

Intra-partition parallel join strategies

Join operations can be performed by subagents in parallel. Parallel join strategies are determined by the characteristics of the data stream.

A join can be parallelized by partitioning or by replicating the data stream on the inner and outer tables of the join, or both. For example, a nested-loop join can be parallelized if its outer stream is partitioned for a parallel scan and the inner stream is again evaluated independently by each subagent. A merged join can be parallelized if its inner and outer streams are value-partitioned for partitioned sorts.

Data filtering and data skew can cause workloads between subagents to become imbalanced while a query executes. The inefficiency of imbalanced workloads is magnified by joins and other computationally expensive operations. The optimizer looks for sources of imbalance in the query's access plan and applies a balancing strategy, ensuring that work is evenly divided between the subagents. For an unordered outer data stream, the optimizer balances the join using the REBAL operator on the outer data stream. For an ordered data stream (where ordered data is produced by an index access or a sort), the optimizer balances the data using a shared sort. A shared sort will not be used if the sort overflows into the

temporary tables, due to the high cost of a sort overflow.

Part 5. Advanced concepts

DB2 capabilities such as federated database environments, replication, DB2 pureScale environments, and the DB2 audit facility help to deliver scalability and high availability in DB2 environments.

Chapter 31. Federated systems

A *federated system* is a special type of distributed database management system (DBMS). A federated system consists of a DB2 instance that operates as a federated server, a database that acts as the federated database, one or more data sources, and clients (users and applications) that access the database and data sources.

With a federated system, you can send distributed requests to multiple data sources within a single SQL statement. For example, you can join data that is located in a DB2 table, an Oracle table, and an XML tagged file in a single SQL statement. The following figure shows the components of a federated system and a sample of the data sources you can access.

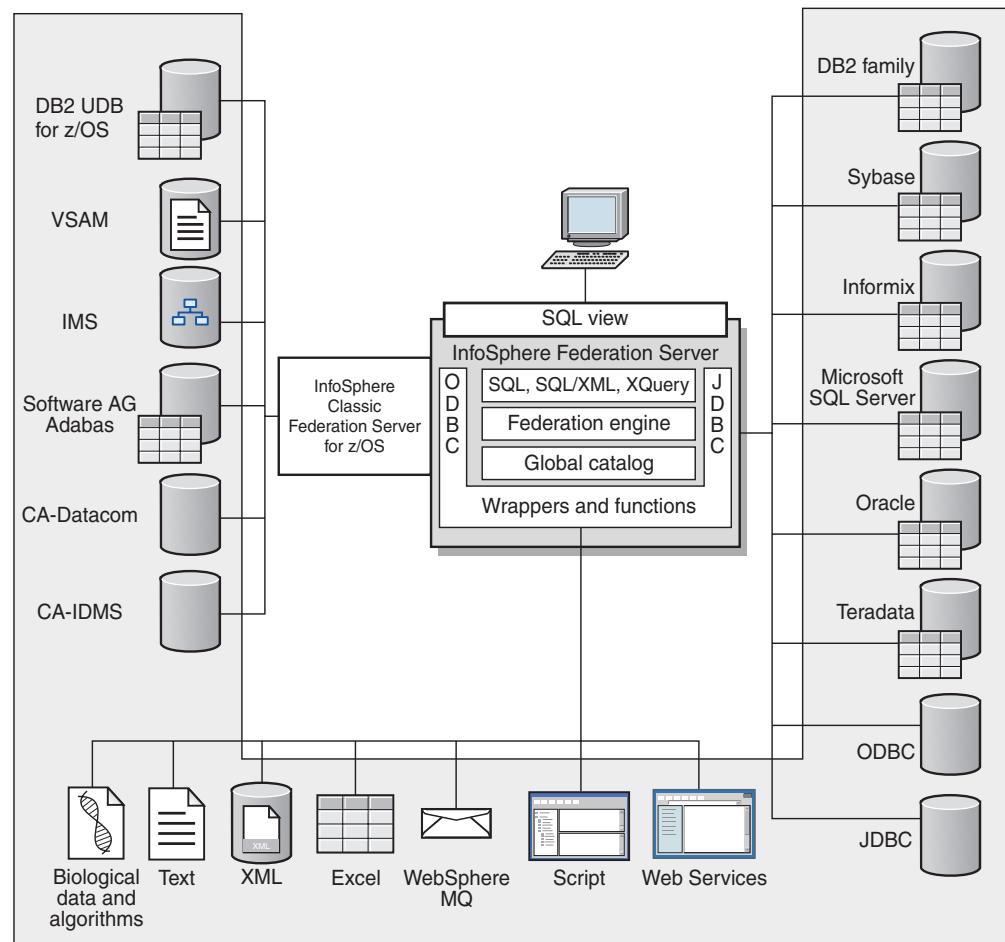


Figure 79. The components of a federated system

The power of a federated system is in its ability to:

- Correlate data from local tables and remote data sources, as if all the data is stored locally in the federated database
- Update data in relational data sources, as if the data is stored in the federated database
- Move data to and from relational data sources

- Take advantage of the data source processing strengths, by sending requests to the data sources for processing
- Compensate for SQL limitations at the data source by processing parts of a distributed request at the federated server

What is a data source?

In a federated system, a *data source* can be a relational database (such as Oracle or Sybase) or a nonrelational data source (such as an XML tagged file).

Through some data sources you can access other data sources. For example, with the ODBC wrapper you can access IBM(r) InfoSphere(tm) Classic Federation Server for z/OS(r) data sources such as DB2 for z/OS, IMS™, CA-IDMS, CA-Datacom, Software AG Adabas, and VSAM.

The method, or protocol, used to access a data source depends on the type of data source. For example, DRDA® is used to access DB2 for z/OS data sources.

Data sources are autonomous. For example, the federated server can send queries to Oracle data sources at the same time that Oracle applications can access these data sources. A federated system does not monopolize or restrict access to the other data sources, beyond integrity and locking constraints.

The federated database

To end users and client applications, data sources appear as a single collective database in the DB2 database system. Users and applications interface with the *federated database* that is managed by the federated server.

The federated database contains a system catalog that stores information about data. The federated database system catalog contains entries that identify data sources and their characteristics. The federated server consults the information stored in the federated database system catalog and the data source wrapper to determine the best plan for processing SQL statements.

The federated system processes SQL statements as if the data from the data sources were ordinary relational tables or views within the federated database. As a result:

- The federated system can correlate relational data with data in nonrelational formats. This is true even when the data sources use different SQL dialects, or do not support SQL at all.
- The characteristics of the federated database take precedence when there are differences between the characteristics of the federated database and the characteristics of the data sources. Query results conform to DB2 semantics, even if data from other non-DB2 data sources is used to compute the query result.

Examples:

- The code page that the federated server uses is different than the code page used that the data source uses. In this case, character data from the data source is converted based on the code page used by the federated database, when that data is returned to a federated user.
- The collating sequence that the federated server uses is different than the collating sequence that the data source uses. In this case, any sort operations on character data are performed at the federated server instead of at the data source.

Wrappers and wrapper modules

Wrappers are mechanisms by which the federated database interacts with data sources. The federated database uses routines stored in a library called a *wrapper module* to implement a wrapper.

These routines allow the federated database to perform operations such as connecting to a data source and retrieving data from it iteratively. Typically, the federated instance owner uses the CREATE WRAPPER statement to register a wrapper in the federated database. You can register a wrapper as fenced or trusted using the DB2_FENCED wrapper option.

You create one wrapper for each type of data source that you want to access. For example, you want to access three DB2 for z/OS database tables, one DB2 for System i® table, two Informix tables, and one Informix view. In this case, you need to create one wrapper for the DB2 data source objects and one wrapper for the Informix data source objects. After these wrappers are registered in the federated database, you can use these wrappers to access other objects from those data sources. For example, you can use the DRDA wrapper with all DB2 family data source objects—DB2 Database for Linux, UNIX, and Windows, DB2 for z/OS, DB2 for System i, and DB2 Server for VM and VSE.

You use the server definitions and nicknames to identify the specifics (name, location, and so forth) of each data source object.

A wrapper performs many tasks. Some of these tasks are:

- It connects to the data source. The wrapper uses the standard connection API of the data source.
- It submits queries to the data source.
 - For data sources that support SQL, the query is submitted in SQL.
 - For data sources that do not support SQL, the query is translated into the native query language of the source or into a series of source API calls.
- It receives results sets from the data source. The wrapper uses the data source standard APIs for receiving results set.
- It responds to federated database queries about the default data type mappings for a data source. The wrapper contains the default type mappings that are used when nicknames are created for a data source object. For relational wrappers, data type mappings that you create override the default data type mappings. User-defined data type mappings are stored in the global catalog.
- It responds to federated database queries about the default function mappings for a data source. The federated database needs data type mapping information for query planning purposes. The wrapper contains information that the federated database needs to determine if DB2 functions are mapped to functions of the data source, and how the functions are mapped. This information is used by the SQL Compiler to determine if the data source is able to perform the query operations. For relational wrappers, function mappings that you create override the default function type mappings. User-defined function mappings are stored in the global catalog.

Wrapper options are used to configure the wrapper or to define how IBM(r) InfoSphere(tm) Federation Server uses the wrapper.

How you interact with a federated system

Because the federated database is a DB2 family database, you can interact with it in a variety of ways.

You can interact with a federated system using by any one of these methods:

- The DB2 command line processor (CLP)
- Application programs
- DB2 family tools
- Web services providers

The steps in the federated documentation provide the commands and SQL statements that can be entered in the DB2 command line processor. The documentation indicates when tasks can be performed through the IBM Data Studio GUI.

The federated server

The DB2 server in a federated system is referred to as the federated server. Any number of DB2 instances can be configured to function as federated servers. You can use existing DB2 instances as your federated servers, or you can create new ones specifically for the federated system.

The DB2 instance that manages the federated system is called a server because it responds to requests from end users and client applications. The federated server often sends parts of the requests it receives to the data sources for processing. A pushdown operation is an operation that is processed remotely. The DB2 instance that manages the federated system is referred to as the federated server, even though it acts as a client when it pushes down requests to the data sources.

Like any other application server, the federated server is a database manager instance. Application processes connect and submit requests to the database within the federated server. However, two main features distinguish it from other application servers:

- A federated server is configured to receive requests that might be partially or entirely intended for data sources. The federated server distributes these requests to the data sources.
- Like other application servers, a federated server uses DRDA communication protocols (over TCP/IP) to communicate with DB2 family instances. However, unlike other application servers, a federated server uses the native client of the data source to access the data source. For example, a federated server uses the Sybase Open Client to access Sybase data sources and an Microsoft SQL Server ODBC Driver to access Microsoft SQL Server data sources.

Federated systems and DB2 pureScale

DB2 for Linux, UNIX, and Windows introduced the DB2 pureScale Feature in Version 9.8. In Version 10, Federation Server functions are integrated with the DB2 pureScale environment.

You can use federation with DB2 pureScale to take advantage of the availability and scalability that the DB2 pureScale feature offers. For detailed information about the pureScale Feature, see Introduction to the IBM DB2 pureScale Feature.

Federation functions can operate on any member of a DB2 pureScale instance. Those functions include the majority of functions that federation supports such as insert, update, and delete operations, and stored procedures.

With automatic client reroute and workload balancing, every member of a DB2 pureScale instance can run federated statements. It is important to install client libraries on all DB2 pureScale members.

Server options that affect federated databases

A federated database system is composed of a DB2 data server (the federated database) and one or more data sources. You identify the data sources to the federated database when you issue CREATE SERVER statements. You can also specify server options that refine and control various aspects of federated system operation.

You must install the distributed join installation option and set the **federated** database manager configuration parameter to YES before you can create servers and specify server options. To change server options later, use the ALTER SERVER statement.

The server option values that you specify on the CREATE SERVER statement affect query pushdown analysis, global optimization, and other aspects of federated database operations. For example, you can specify performance statistics as server option values. The *cpu_ratio* option specifies the relative speeds of the processors at the data source and the federated server, and the *io_ratio* option specifies the relative rates of the data I/O divides at the data source and the federated server.

Server option values are written to the system catalog (SYSCAT.SERVEROPTIONS), and the optimizer uses this information when it develops access plans for the data source. If a statistic changes (for example, when a data source processor is upgraded), use the ALTER SERVER statement to update the catalog with the new value.

Federated database query-compiler phases

In a federated database, there are additional steps that are taken by the query compiler. These additional steps are needed to determine which remote data sources to use to improve query performance.

Global analysis on the federated database might result in information that can be used to optimize the federated environment overall.

Federated database pushdown analysis

For queries that are to run against federated databases, the optimizer performs pushdown analysis to determine whether a particular operation can be performed at a remote data source.

An operation might be a function, such as a relational operator, or a system or user function; or it might be an SQL operator, such as, for example, ORDER BY or GROUP BY.

Be sure to update local catalog information regularly, so that the DB2 query compiler has access to accurate information about SQL support at remote data

sources. Use DB2 data definition language (DDL) statements (such as CREATE FUNCTION MAPPING or ALTER SERVER, for example) to update the catalog.

If functions cannot be pushed down to the remote data source, they can significantly impact query performance. Consider the effect of forcing a selective predicate to be evaluated locally instead of at the data source. Such evaluation could require the DB2 server to retrieve the entire table from the remote data source and then filter it locally against the predicate. Network constraints and a large table could cause performance to suffer.

Operators that are not pushed down can also significantly affect query performance. For example, having a GROUP BY operator aggregate remote data locally could also require the DB2 server to retrieve an entire table from the remote data source.

For example, consider nickname N1, which references the data source table EMPLOYEE in a DB2 for z/OS data source. The table has 10 000 rows, one of the columns contains the last names of employees, and one of the columns contains salaries. The optimizer has several options when processing the following statement, depending on whether the local and remote collating sequences are the same:

```
select lastname, count(*)  from n1
  where
    lastname > 'B' and
    salary > 50000
  group by lastname
```

- If the collating sequences are the same, the query predicates can probably be pushed down to DB2 for z/OS. Filtering and grouping results at the data source is usually more efficient than copying the entire table and performing the operations locally. For this query, the predicates and the GROUP BY operation can take place at the data source.
- If the collating sequences are not the same, both predicates cannot be evaluated at the data source. However, the optimizer might decide to push down the salary > 50000 predicate. The range comparison must still be done locally.
- If the collating sequences are the same, and the optimizer knows that the local DB2 server is very fast, the optimizer might decide that performing the GROUP BY operation locally is the least expensive approach. The predicate is evaluated at the data source. This is an example of pushdown analysis combined with global optimization.

In general, the goal is to ensure that the optimizer evaluates functions and operators at remote data sources. Many factors affect whether a function or an SQL operator can be evaluated at a remote data source, including the following:

- Server characteristics
- Nickname characteristics
- Query characteristics

Server characteristics that affect pushdown opportunities

Certain data source-specific factors can affect pushdown opportunities. In general, these factors exist because of the rich SQL dialect that is supported by the DB2 product. The DB2 data server can compensate for the lack of function that is available at another data server, but doing so might require that the operation take place at the DB2 server.

- SQL capabilities

Each data source supports a variation of the SQL dialect and different levels of functionality. For example, most data sources support the GROUP BY operator, but some limit the number of items on the GROUP BY list, or have restrictions on whether an expression is allowed on the GROUP BY list. If there is a restriction at the remote data source, the DB2 server might have to perform a GROUP BY operation locally.

- **SQL restrictions**

Each data source might have different SQL restrictions. For example, some data sources require parameter markers to bind values to remote SQL statements. Therefore, parameter marker restrictions must be checked to ensure that each data source can support such a bind mechanism. If the DB2 server cannot determine a good method to bind a value for a function, this function must be evaluated locally.

- **SQL limits**

Although the DB2 server might allow the use of larger integers than those that are permitted on remote data sources, values that exceed remote limits cannot be embedded in statements that are sent to data sources, and any impacted functions or operators must be evaluated locally.

- **Server specifics**

Several factors fall into this category. For example, if null values at a data source are sorted differently from how the DB2 server would sort them, ORDER BY operations on a nullable expression cannot be remotely evaluated.

- **Collating sequence**

Retrieving data for local sorts and comparisons usually decreases performance. If you configure a federated database to use the same collating sequence that a data source uses and then set the COLLATING_SEQUENCE server option to Y, the optimizer can consider pushing down many query operations. The following operations might be pushed down if collating sequences are the same:

- Comparisons of character or numeric data
- Character range comparison predicates
- Sorts

You might get unusual results, however, if the weighting of null characters is different between the federated database and the data source. Comparisons might return unexpected results if you submit statements to a case-insensitive data source. The weights that are assigned to the characters "I" and "i" in a case-insensitive data source are the same. The DB2 server, by default, is case sensitive and assigns different weights to these characters.

To improve performance, the federated server allows sorts and comparisons to take place at data sources. For example, in DB2 for z/OS, sorts that are defined by ORDER BY clauses are implemented by a collating sequence that is based on an EBCDIC code page. To use the federated server to retrieve DB2 for z/OS data that is sorted in accordance with ORDER BY clauses, configure the federated database so that it uses a predefined collating sequence based on the EBCDIC code page.

If the collating sequences of the federated database and the data source differ, the DB2 server retrieves the data to the federated database. Because users expect to see query results ordered by the collating sequence that is defined for the federated server, ordering the data locally ensures that this expectation is fulfilled. Submit your query in passthrough mode, or define the query in a data source view if you need to see the data ordered in the collating sequence of the data source.

- **Server options**

Several server options can affect pushdown opportunities, including COLLATING_SEQUENCE, VARCHAR_NO_TRAILING_BLANKS, and PUSHDOWN.

- DB2 type mapping and function mapping factors

The default local data type mappings on the DB2 server are designed to provide sufficient buffer space for each data source data type, which avoids loss of data. You can customize the type mapping for a specific data source to suit specific applications. For example, if you are accessing an Oracle data source column with a DATE data type, which by default is mapped to the DB2 TIMESTAMP data type, you might change the local data type to the DB2 DATE data type.

In the following three cases, the DB2 server can compensate for functions that a data source does not support:

- The function does not exist at the remote data source.
- The function exists, but the characteristics of the operand violate function restrictions. The IS NULL relational operator is an example of this situation. Most data sources support it, but some might have restrictions, such as allowing a column name to appear only on the left hand side of the IS NULL operator.
- The function might return a different result if it is evaluated remotely. An example of this situation is the greater than ('>') operator. For data sources with different collating sequences, the greater than operator might return different results if it is evaluated locally by the DB2 server.

Nickname characteristics that affect pushdown opportunities

The following nickname-specific factors can affect pushdown opportunities.

- Local data type of a nickname column

Ensure that the local data type of a column does not prevent a predicate from being evaluated at the data source. Use the default data type mappings to avoid possible overflow. However, a joining predicate between two columns of different lengths might not be considered at a data source whose joining column is shorter, depending on how DB2 binds the longer column. This situation can affect the number of possibilities that the DB2 optimizer can evaluate in a joining sequence. For example, Oracle data source columns that were created using the INTEGER or INT data type are given the type NUMBER(38). A nickname column for this Oracle data type is given the local data type FLOAT, because the range of a DB2 integer is from 2^{31} to $(-2^{31})-1$, which is roughly equivalent to NUMBER(9). In this case, joins between a DB2 integer column and an Oracle integer column cannot take place at the DB2 data source (because of the shorter joining column); however, if the domain of this Oracle integer column can be accommodated by the DB2 INTEGER data type, change its local data type with the ALTER NICKNAME statement so that the join can take place at the DB2 data source.

- Column options

Use the ALTER NICKNAME statement to add or change column options for nicknames.

Use the VARCHAR_NO_TRAILING_BLANKS option to identify a column that contains no trailing blanks. The compiler pushdown analysis step will then take this information into account when checking all operations that are performed on such columns. The DB2 server might generate a different but equivalent form of a predicate to be used in the SQL statement that is sent to a data source. You might see a different predicate being evaluated against the data source, but the net result should be equivalent.

Use the NUMERIC_STRING option to indicate whether the values in that column are always numbers without trailing blanks.

Table 43 describes these options.

Table 43. Column Options and Their Settings

Option	Valid Settings	Default Setting
NUMERIC_STRING	<p>Y: Specifies that this column contains only strings of numeric data. It does not contain blank characters that could interfere with sorting of the column data. This option is useful when the collating sequence of a data source is different from that of the DB2 server. Columns that are marked with this option are not excluded from local (data source) evaluation because of a different collating sequence. If the column contains only numeric strings that are followed by trailing blank characters, do not specify Y.</p> <p>N: Specifies that this column is not limited to strings of numeric data.</p>	N
VARCHAR_NO_TRAILING_BLANKS	<p>Y: Specifies that this data source uses non-blank-padded VARCHAR comparison semantics, similar to the DB2 data server. For variable-length character strings that contain no trailing blank characters, non-blank-padded comparison semantics of some data servers return the same results as DB2 comparison semantics. Specify this value if you are certain that all VARCHAR table or view columns at a data source contain no trailing blank characters</p> <p>N: Specifies that this data source does not use non-blank-padded VARCHAR comparison semantics, similar to the DB2 data server.</p>	N

Query characteristics that affect pushdown opportunities

A query can reference an SQL operator that might involve nicknames from multiple data sources. The operation must take place on the DB2 server to combine the results from two referenced data sources that use one operator, such as a set operator (for example, UNION). The operator cannot be evaluated at a remote data source directly.

Guidelines for determining where a federated query is evaluated

The DB2 explain utility, which you can start by invoking the `db2expln` command, shows where queries are evaluated. The execution location for each operator is included in the command output.

- If a query is pushed down, you should see a RETURN operator, which is a standard DB2 operator. If a SELECT statement retrieves data from a nickname, you also see a SHIP operator, which is unique to federated database operations: it changes the server property of the data flow and separates local operators from remote operators. The SELECT statement is generated using the SQL dialect that is supported by the data source.

- If an INSERT, UPDATE, or DELETE statement can be entirely pushed down to the remote data source, you might not see a SHIP operator in the access plan. All remotely executed INSERT, UPDATE, or DELETE statements are shown for the RETURN operator. However, if a query cannot be pushed down in its entirety, the SHIP operator shows which operations were performed remotely.

Understanding why a query is evaluated at a data source instead of by the DB2 server

Consider the following key questions when you investigate ways to increase pushdown opportunities:

- Why isn't this predicate being evaluated remotely?

This question arises when a very selective predicate could be used to filter rows and reduce network traffic. Remote predicate evaluation also affects whether a join between two tables of the same data source can be evaluated remotely.

Areas to examine include:

- Subquery predicates. Does this predicate contain a subquery that pertains to another data source? Does this predicate contain a subquery that involves an SQL operator that is not supported by this data source? Not all data sources support set operators in a subquery predicate.
- Predicate functions. Does this predicate contain a function that cannot be evaluated by this remote data source? Relational operators are classified as functions.
- Predicate bind requirements. If it is remotely evaluated, does this predicate require bind-in of some value? Would that violate SQL restrictions at this data source?
- Global optimization. The optimizer might have decided that local processing is more cost effective.
- Why isn't the GROUP BY operator evaluated remotely?

Areas to examine include:

- Is the input to the GROUP BY operator evaluated remotely? If the answer is no, examine the input.
- Does the data source have any restrictions on this operator? Examples include:
 - A limited number of GROUP BY items
 - Limited byte counts for combined GROUP BY items
 - Column specifications only on the GROUP BY list
 - Does the data source support this SQL operator?
- Global optimization. The optimizer might have decided that local processing is more cost effective.
- Does the GROUP BY clause contain a character expression? If it does, verify that the remote data source and the DB2 server have the same case sensitivity.
- Why isn't the set operator evaluated remotely?

Areas to examine include:

- Are both of its operands evaluated in their entirety at the same remote data source? If the answer is no, and it should be yes, examine each operand.
- Does the data source have any restrictions on this set operator? For example, are large objects (LOBs) or LONG field data valid input for this specific set operator?
- Why isn't the ORDER BY operation evaluated remotely?

Areas to examine include:

- Is the input to the ORDER BY operation evaluated remotely? If the answer is no, examine the input.
- Does the ORDER BY clause contain a character expression? If yes, do the remote data source and the DB2 server have different collating sequences or case sensitivities?
- Does the remote data source have any restrictions on this operator? For example, is there a limit to the number of ORDER BY items? Does the remote data source restrict column specification to the ORDER BY list?

Remote SQL generation and global optimization in federated databases

For a federated database query that uses relational nicknames, the access strategy might involve breaking down the original query into a set of remote query units and then combining the results. Such remote SQL generation helps to produce a globally optimized access strategy for a query.

The optimizer uses the output of pushdown analysis to decide whether each operation is to be evaluated locally at the DB2 server or remotely at a data source. It bases its decision on the output of its cost model, which includes not only the cost of evaluating the operation, but also the cost of shipping the data and messages between the DB2 server and the remote data source.

Although the goal is to produce an optimized query, the following factors significantly affect global optimization, and thereby query performance.

- Server characteristics
- Nickname characteristics

Server options that affect global optimization

The following data source server options can affect global optimization:

- Relative ratio of processing speed

Use the CPU_RATIO server option to specify how fast or slow the processing speed at the data source should be relative to the processing speed at the DB2 server. A low ratio indicates that the processing speed at the data source is faster than the processing speed at the DB2 server; in this case, the DB2 optimizer is more likely to consider pushing processor-intensive operations down to the data source.

- Relative ratio of I/O speed

Use the IO_RATIO server option to specify how fast or slow the system I/O speed at the data source should be relative to the system I/O speed at the DB2 server. A low ratio indicates that the I/O speed at the data source is faster than the I/O speed at the DB2 server; in this case, the DB2 optimizer is more likely to consider pushing I/O-intensive operations down to the data source.

- Communication rate between the DB2 server and the data source

Use the COMM_RATE server option to specify network capacity. Low rates, which indicate slow network communication between the DB2 server and a data source, encourage the DB2 optimizer to reduce the number of messages that are sent to or from this data source. If the rate is set to 0, the optimizer creates an access plan that requires minimal network traffic.

- Data source collating sequence

Use the COLLATING_SEQUENCE server option to specify whether a data source collating sequence matches the local DB2 database collating sequence. If this option is not set to Y, the DB2 optimizer considers any data that is retrieved from this data source as being unordered.

- Remote plan hints

Use the PLAN_HINTS server option to specify that plan hints should be generated or used at a data source. By default, the DB2 server does not send any plan hints to the data source.

Plan hints are statement fragments that provide extra information to the optimizer at a data source. For some queries, this information can improve performance. The plan hints can help the optimizer at a data source to decide whether to use an index, which index to use, or which table join sequence to use.

If plan hints are enabled, the query that is sent to the data source contains additional information. For example, a statement with plan hints that is sent to an Oracle optimizer might look like this:

```
select /*+ INDEX (table1, t1index)*/
       coll
      from table1
```

The plan hint is the string: `/*+ INDEX (table1, t1index)*/`

- Information in the DB2 optimizer knowledge base

The DB2 server has an optimizer knowledge base that contains data about native data sources. The DB2 optimizer does not generate remote access plans that cannot be generated by specific database management systems (DBMSs). In other words, the DB2 server avoids generating plans that optimizers at remote data sources cannot understand or accept.

Nickname characteristics that affect global optimization

The following nickname-specific factors can affect global optimization.

- Index considerations

To optimize queries, the DB2 server can use information about indexes at data sources. For this reason, it is important that the available index information be current. Index information for a nickname is initially acquired when the nickname is created. Index information is not collected for view nicknames.

- Creating index specifications on nicknames

You can create an index specification for a nickname. Index specifications build an index definition (not an actual index) in the catalog for the DB2 optimizer to use. Use the CREATE INDEX SPECIFICATION ONLY statement to create index specifications. The syntax for creating an index specification on a nickname is similar to the syntax for creating an index on a local table. Consider creating index specifications in the following circumstances:

- When the DB2 server cannot retrieve any index information from a data source during nickname creation
- When you want an index for a view nickname
- When you want to encourage the DB2 optimizer to use a specific nickname as the inner table of a nested-loop join. You can create an index on the joining column, if none exists.

Before you issue CREATE INDEX statements against a nickname for a view, consider whether you need one. If the view is a simple SELECT on a table with an index, creating local indexes on the nickname to match the indexes on the table at the data source can significantly improve query performance. However,

if indexes are created locally over a view that is not a simple SELECT statement, such as a view that is created by joining two tables, query performance might suffer. For example, if you create an index over a view that is a join between two tables, the optimizer might choose that view as the inner element in a nested-loop join. The query will perform poorly, because the join is evaluated several times. An alternate approach is to create nicknames for each of the tables that are referenced in the data source view, and then to create a local view at the DB2 server that references both nicknames.

- Catalog statistics considerations

System catalog statistics describe the overall size of nicknames and the range of values in associated columns. The optimizer uses these statistics when it calculates the least-cost path for processing queries that contain nicknames. Nickname statistics are stored in the same catalog views as table statistics.

Although the DB2 server can retrieve the statistical data that is stored at a data source, it cannot automatically detect updates to that data. Furthermore, the DB2 server cannot automatically detect changes to the definition of objects at a data source. If the statistical data for-or the definition of-an object has changed, you can:

- Run the equivalent of a **RUNSTATS** command at the data source, drop the current nickname, and then recreate it. Use this approach if an object's definition has changed.
- Manually update the statistics in the SYSSTAT.TABLES catalog view. This approach requires fewer steps, but it does not work if an object's definition has changed.

Global analysis of federated database queries

The DB2 explain utility, which you can start by invoking the **db2expln** command, shows the access plan that is generated by the remote optimizer for those data sources that are supported by the remote explain function. The execution location for each operator is included in the command output.

You can also find the remote SQL statement that was generated for each data source in the SHIP or RETURN operator, depending on the type of query. By examining the details for each operator, you can see the number of rows that were estimated by the DB2 optimizer as input to and output from each operator.

Understanding DB2 optimization decisions

Consider the following key questions when you investigate ways to increase performance:

- Why isn't a join between two nicknames of the same data source being evaluated remotely?

Areas to examine include:

- Join operations. Can the remote data source support them?
- Join predicates. Can the join predicate be evaluated at the remote data source? If the answer is no, examine the join predicate.

- Why isn't the GROUP BY operator being evaluated remotely?

Examine the operator syntax, and verify that the operator can be evaluated at the remote data source.

- Why is the statement not being completely evaluated by the remote data source? The DB2 optimizer performs cost-based optimization. Even if pushdown analysis indicates that every operator can be evaluated at the remote data source, the

optimizer relies on its cost estimate to generate a global optimization plan. There are a great many factors that can contribute to that plan. For example, even though the remote data source can process every operation in the original query, its processing speed might be much slower than the processing speed of the DB2 server, and it might turn out to be more beneficial to perform the operations at the DB2 server instead. If results are not satisfactory, verify your server statistics in the SYSCAT.SERVEROPTIONS catalog view.

- Why does a plan that is generated by the optimizer, and that is completely evaluated at a remote data source, perform much more poorly than the original query executed directly at the remote data source?

Areas to examine include:

- The remote SQL statement that is generated by the DB2 optimizer. Ensure that this statement is identical to the original query. Check for changes in predicate order. A good query optimizer should not be sensitive to the order of predicates in a query. The optimizer at the remote data source might generate a different plan, based on the order of input predicates. Consider either modifying the order of predicates in the input to the DB2 server, or contacting the service organization of the remote data source for assistance. You can also check for predicate replacements. A good query optimizer should not be sensitive to equivalent predicate replacements. The optimizer at the remote data source might generate a different plan, based on the input predicates. For example, some optimizers cannot generate transitive closure statements for predicates.
- Additional functions. Does the remote SQL statement contain functions that are not present in the original query? Some of these functions might be used to convert data types; be sure to verify that they are necessary.

Chapter 32. IBM Replication solutions

IBM offers two primary replication solutions: Q replication and SQL replication.

The primary components of Q replication are the Q Capture program and the Q Apply program. The primary components of SQL replication are the Capture program and Apply program. Both types of replication share the Replication Alert Monitor tool. You can set up and administer these replication components using the Replication Center and the ASNCLP command-line program.

The following list briefly summarizes these replication components:

Q Capture program

Reads the DB2 recovery log looking for changes to DB2 source tables and translates committed source data into WebSphere MQ messages that can be published in XML format to a subscribing application, or replicated in a compact format to the Q Apply program.

Q Apply program

Takes WebSphere MQ messages from a queue, transforms the messages into SQL statements, and updates a target table or stored procedure. Supported targets include DB2 databases or subsystems and Oracle, Sybase, Informix and Microsoft SQL Server databases that are accessed through federated server nicknames.

Capture program

Reads the DB2 recovery log for changes made to registered source tables or views and then stages committed transactional data in relational tables called change-data (CD) tables, where they are stored until the target system is ready to copy them. SQL replication also provides Capture triggers that populate a staging table called a consistent-change-data (CCD) table with records of changes to non-DB2 source tables.

Apply program

Reads data from staging tables and makes the appropriate changes to targets. For non-DB2 data sources, the Apply program reads the CCD table through that table's nickname on the federated database and makes the appropriate changes to the target table.

Replication Alert Monitor

A utility that checks the health of the Q Capture, Q Apply, Capture, and Apply programs. It checks for situations in which a program terminates, issues a warning or error message, reaches a threshold for a specified value, or performs a certain action, and then issues notifications to an email server, pager, or the z/OS console.

Use the Replication Center to:

- Define registrations, subscriptions, publications, queue maps, alert conditions, and other objects.

- Start, stop, suspend, resume, and reinitialize the replication programs.
- Specify the timing of automated copying.
- Specify SQL enhancements to the data.
- Define relationships between the source and the target tables.

Replication tools

The replication tools consist of the ASNCLP command-line program, the Replication Center, and the Replication Alert Monitor tool.

ASNCLP command-line program

You can use the ASNCLP program to administer SQL replication, Q replication, Classic replication, relational event publishing, and the Replication Alert Monitor .

You can use the ASNCLP commands to create, modify, and remove information in control tables of the replication programs. You can also use the ASNCLP commands to generate SQL scripts to create, modify, and remove information about replication sources, targets, queues, and other options in control tables.

Replication Center

You can use the Replication Center to set up and administerg the replication components for Q replication and SQL replication. The Replication Center is only supported in Linux and Windows operating systems.

Use the Replication Center to perform the following tasks:

- Define registrations, subscriptions, publications, queue maps, alert conditions, and other objects
- Start, stop, suspend, resume, and reinitialize the replication programs
- Specify the timing of automated copying
- Specify SQL enhancements to the data
- Define relationships between the source and the target tables

The following wizards and launchpads are available from the Replication Center:

- Replication Center launchpad
- Add Capture Control Server wizard
- Add Apply Control Server wizard
- Add Monitor Control Server wizard
- Add Q Capture Server wizard
- Add Q Apply Server wizard
- Add Server Information wizard
- Create Monitor wizard
- Create Q Capture Control Tables wizard
- Create Q Apply Control Tables wizard
- Create Q Subscriptions wizard
- Create Publications wizard

The Replication Center is installed by default as part of the replication tools component in typical or custom installations. The compact installation no longer installs the replication tools component, which includes the Replication Center.

Replication Alert Monitor

The Replication Alert Monitor is a utility that checks the health of the Q Capture, Q Apply, Capture, and Apply programs. It checks for situations in which a program terminates, issues a warning or error message, reaches a threshold for a specified value, or performs a certain action, and then issues notifications to an email server, pager, or the z/OS console.

Changes to the Replication Center in DB2 10.1

The Replication Center is now a stand-alone tool. Installation default options and command to start the Replication Center have changed.

Differences with earlier releases

In Version 10.1, the Replication Center is available as a stand-alone tool on Linux and Windows operating systems. In earlier releases, it was grouped with other Administration tools such as the Control Center. The Administration tools have been discontinued.

In Version 10.1, the Replication Center is installed as part of the replication tools component by default in typical or custom installations. However, the compact installation no longer installs the replication tools component, which includes the Replication Center. In earlier releases, the replication tools were a required component for compact installations for certain products. For information about the DB2 database product editions that include Replication tools, see Functionality in DB2 features and DB2 product editions.

In Version 10.1, the **db2rc** command is available to start the Replication Center. The **db2cc -rc** command that was available in previous releases is discontinued.

On Windows operating systems, you can also click **Start > Programs > IBM DB2 > DB2 copy name > Replication Center**, where *DB2 copy name* indicates the name of the DB2 copy that you specified during installation.

All the Replication Center functionality of previous releases is still available and supported.

New ways to access the replication tools

To install the Replication Center, make sure that you select a typical or custom installation for any of the DB2 database products.

To start the Replication Center, issue the **db2rc** command. On Windows operating systems, you can also use the **Start** menu.

Chapter 33. DB2 pureScale feature

In a competitive, ever-changing global business environment, you cannot afford to let your IT infrastructure slow you down. This reality demands IT systems that provide capacity as needed, exceptional levels of availability, and transparency toward your existing applications.

When workloads grow, does your distributed database system require you to change your applications or change how data is distributed? If so, your system does not scale transparently. Even simple application changes incur time and cost penalties and can pose risks to system availability. The stakes are always high: Every second lost in system availability can have a direct bearing on customer retention, compliance with service level agreements, and your bottom line.

The IBM DB2 pureScale Feature might help reduce the risk and cost associated with growing your distributed database solution by providing extreme capacity and application transparency. Designed for continuous availability-high availability capable of exceeding even the strictest industry standard-this feature tolerates both planned maintenance and component failure with ease.

With the DB2 pureScale Feature, scaling your database solution is simple. Multiple database servers, known as members, process incoming database requests; these members operate in a clustered system and share data. You can transparently add more members to scale out to meet even the most demanding business needs. There are no application changes to make, data to redistribute, or performance tuning to do.

To deliver on a design capable of exceptional levels of database availability, the DB2 pureScale Feature builds on familiar and proven design features from DB2 for z/OS database software. By also integrating several advanced hardware and software technologies, the DB2 pureScale Feature supports the strictest requirements for high fault tolerance and can sustain processing of database requests even under extreme circumstances.

In the sections that follow, you can learn more about these design features and benefits of the DB2 pureScale Feature:

Extreme capacity

The IBM DB2 pureScale Feature can scale with near-linear efficiency and high predictability. Adding capacity is as simple as adding new members to the instance.

High scalability

During testing with typical web commerce and OLTP workloads, the DB2 pureScale Feature demonstrated that it can scale to different levels with exceptional efficiency; the maximum supported configuration provides extreme capacity. To scale out, your existing applications do not have to be aware of the topology of

your DB2 pureScale environment.³
When two more members join the instance, they immediately begin processing

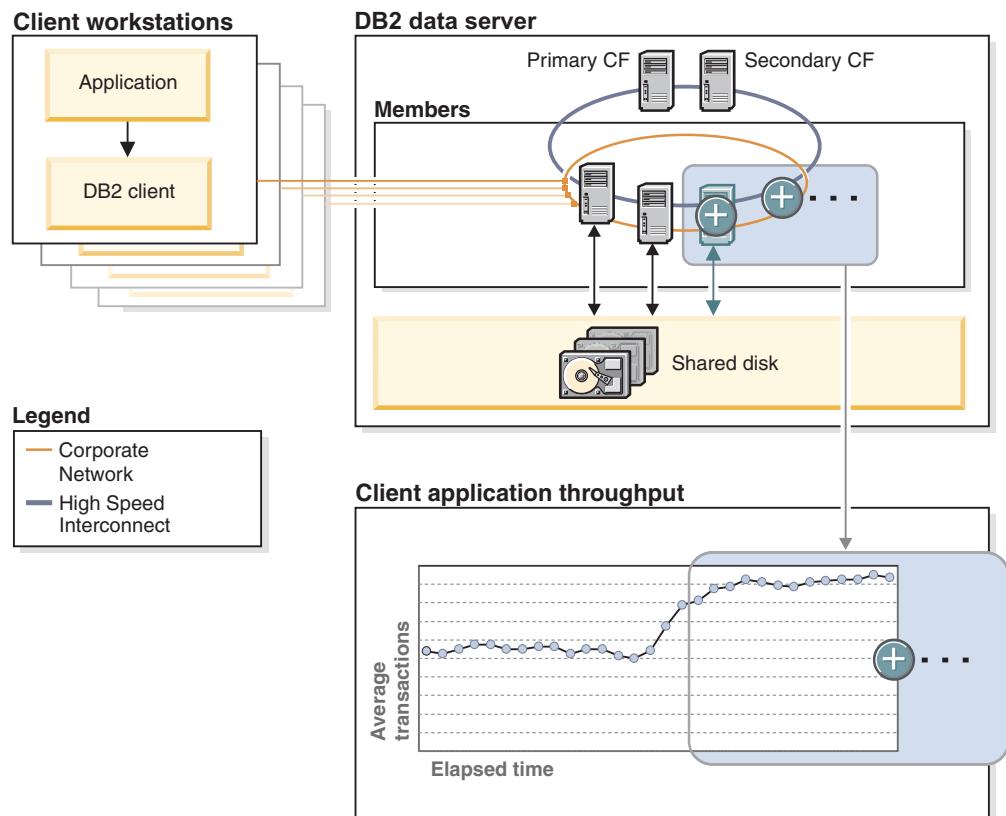


Figure 80. Scalability of a DB2 pureScale environment. Additional members begin processing incoming database requests as soon as they join the instance. Overall throughput almost doubles as the number of members doubles.

incoming database requests. Overall throughput almost doubles as the number of members doubles. For more information about scalability, see the DB2 pureScale Feature road map.

Scalability by design

Why does the DB2 pureScale Feature scale so well? The answer lies in the highly efficient design, which tightly integrates several advanced hardware and software technologies.

For example, the cluster caching facility (CF) handles instance-wide lock management and global caching with great efficiency. Without the equivalent of such a dedicated component to handle locking and caching, the database servers in a cluster must communicate with each other to maintain vital locking and data consistency information. Each time that a database server is added, the amount of communication "chatter" increases, reducing scale-out efficiency.

Even in the maximum supported configuration, your DB2 pureScale environment communicates efficiently. Data pages in the group buffer pool (global cache) are shared between members and the cluster caching facility through Remote Direct

3. During testing, database requests were workload balanced across members by the DB2 pureScale Feature, not routed. Update and select operations were randomized to ensure that the location of data on the shared disk storage had no effect on scalability.

Memory Access (RDMA), without requiring any processor time or I/O cycles on members. All operations are performed over the InfiniBand high-speed interconnect and do not require context switching or routing through a slower IP network stack. Round-trip communication times between cluster components are typically measured in the low tens of microseconds. The end result is an instance that is always aware of what data is in flight and where, but without the performance penalty.

Continuous availability

Whether it is planned system maintenance or an extreme circumstance, such as when multiple components fail simultaneously, the IBM DB2 pureScale Feature is designed to continue processing incoming database requests without interruption. Automatic load balancing across all active members means optimal resource utilization at all times, which helps to keep application response times low.

Unplanned events

A sudden software or hardware failure can be highly disruptive, even in a system that employs redundant components. The DB2 pureScale Feature incorporates several design features to deliver fault tolerance that not only can keep your instance available but also minimizes the effect of component failures on the rest of the database system.

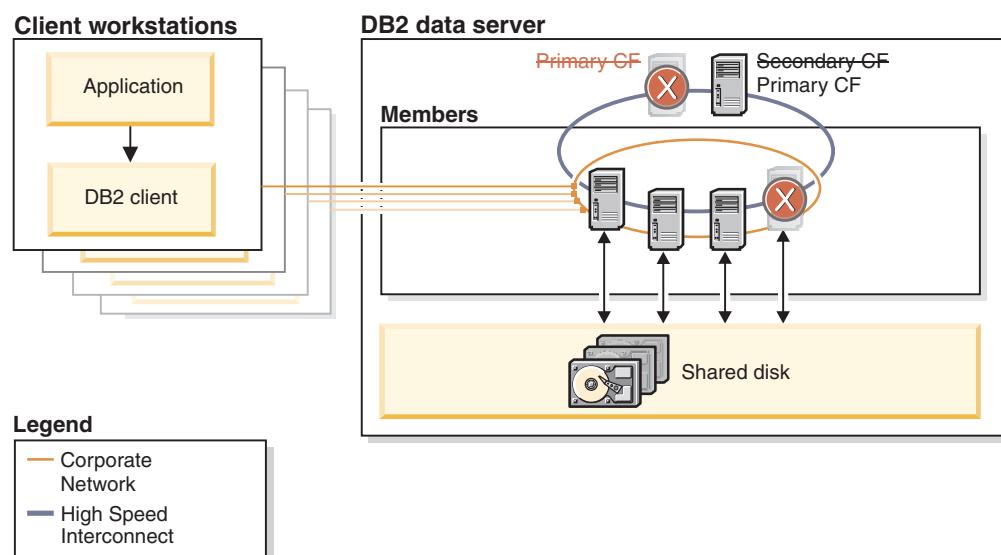


Figure 81. Component failures in a DB2 pureScale environment; database requests continue to be processed.

Robust heartbeat detection ensures that failed components are identified and isolated rapidly. Recovery from component failures is fully automatic and requires no intervention.

If a member fails while processing database requests, it is immediately fenced off from the rest of the system. During the failure, most of your data on the shared disk storage remains available to active members processing database requests. Only the data that was inflight on the failed member is temporarily held by a retained lock until the DB2 pureScale Feature completes the automated member crash recovery.

After a software failure, the member is restarted on its home host, and recovery is performed. The member resumes transaction processing as soon as recovery is complete. After a hardware failure, the member restarts on another host (a process known as *restart light*) so that the data can be recovered. As soon as its home host is available again, the member fails back to that host, restarts, and resumes processing.

After a software or hardware failure on the primary cluster caching facility, a secondary, duplexed cluster caching facility automatically takes over the primary role. This takeover is transparent to applications and causes only minimal delay because of the continuous duplexing of locking and caching information between cluster caching facilities. The instance remains available.

Planned events

System maintenance in a DB2 pureScale environment is designed to cause as little disruption as possible. You can roll out system upgrades without stopping the DB2 pureScale instance or affecting database availability.

To perform system maintenance on a member, you quiesce it. After existing transactions on the member are completed (drained), you take the member offline and perform the system maintenance. During the maintenance period, new transaction requests are automatically directed to other, active members, a process that is transparent to applications.

After the maintenance is complete and you restart the member, it begins processing database transactions again as soon as it rejoins the instance.

Application transparency

Getting started with the IBM DB2 pureScale Feature is quick and simple: Applications do not have to be aware of the topology of your database environment when you deploy the feature. This means that applications work just as they did before, yet they can benefit

from the extreme capacity and continuous availability from the moment that you start your DB2 pureScale instance for the first time.

Increasing capacity

Capacity planning with the DB2 pureScale Feature is simple. You can start small and add members to your database environment as your needs grow, scaling out from the most basic highly available configuration all the way to the maximum supported configuration, which provides extreme processing capacity. Scaling is near linear in efficiency and highly predictable.

When you scale out, no application changes or repartitioning of your data is required. No performance tuning is required to scale efficiently. If you need more capacity, you simply add more members.

Maintaining availability

Maintaining database availability means both compliance with service level agreements (SLAs) and high tolerance to component failures. To maximize hardware utilization rates and help keep response times consistent for your applications, incoming database requests are automatically load balanced across all

active members in your DB2 pureScale instance. To minimize the impact of component failures, the automated restart and recovery process of the DB2 pureScale Feature runs quickly and without affecting most database requests. Only those database requests that were being processed by a failed member must be resubmitted by the originating application; the resubmitted requests are then processed by the next available member.

In the example in the following diagram, several events take place in short succession. Multiple component failures require automated, internal recovery, and a scale-out operation increases the capacity of the DB2 pureScale instance. None of these events requires any application awareness. The box containing the components of the DB2 pureScale Feature is shaded to indicate application transparency.

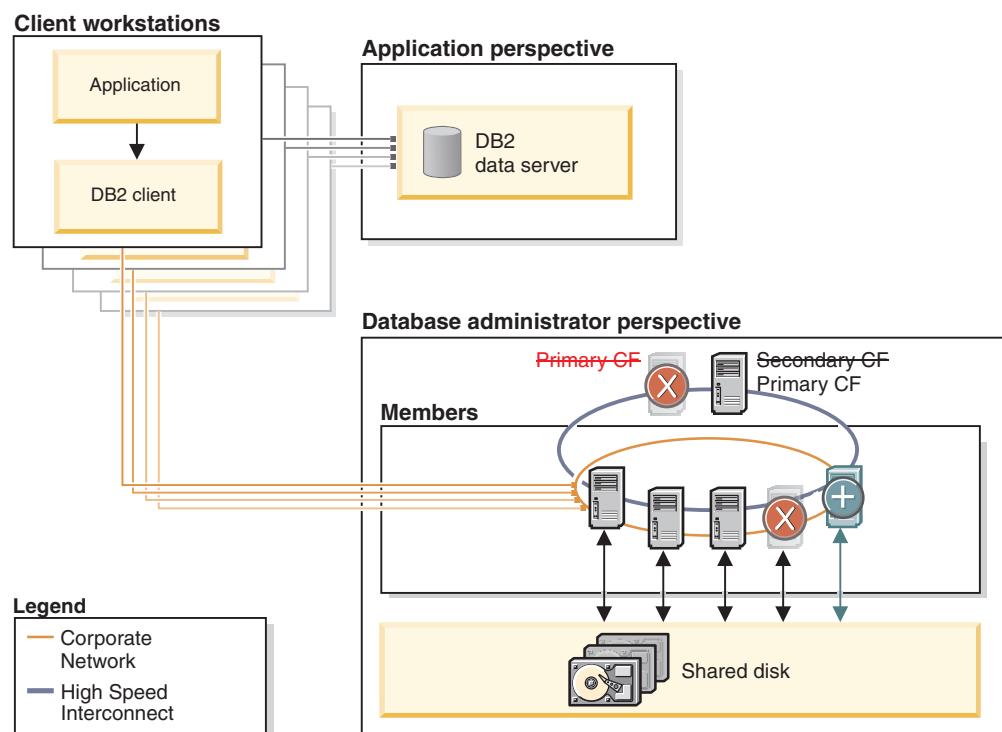


Figure 82. A DB2 pureScale environment encountering multiple component failures and being scaled out. Applications connecting to the database need not be aware of these events.

Planning made simple

The ability to easily add and remove resources helps you to manage challenges such as the following ones:

- Cyclical workloads. If some of your workloads are cyclical (for example, seasonal), you can add resources before they are required, and then move the extra capacity somewhere else later on.
- Sudden increases in workloads. An SLA might dictate minimum response times for completing database requests. If you discover sudden workload surges from some applications that are threatening response times, you can help meet your SLA by quickly moving additional members to the database that is experiencing the peak demand.
- Maintenance-related slowdowns. To help negate the effect of system maintenance on the overall throughput of your DB2 pureScale environment, you

can add a member to your environment before commencing maintenance on an existing member. After you complete the system maintenance and the original member rejoins the instance, you can remove the additional resource or perform maintenance on other members.

Getting started with the DB2 pureScale Feature

The IBM DB2 pureScale Feature greatly simplifies the deployment of an inherently complex distributed database environment. All software components are installed and configured for you from a single host.

A single invocation of the wizard from the installation-initiating host installs all of the components of the DB2 pureScale Feature across all hosts that you specify as part of the DB2 pureScale environment. All software components, including the following ones, are integrated tightly into the DB2 pureScale Feature:

- DB2 members
- The cluster caching facilities
- DB2 cluster services instance management software, which is based on IBM Tivoli System Automation for Multiplatforms
- The cluster file system, which is based on GPFS

The DB2 Information Center provides more information about the underlying DB2 pureScale components, but you do not need to be closely familiar with them to deploy the feature. What is important is that you meet the installation prerequisites before running the DB2 Setup wizard.

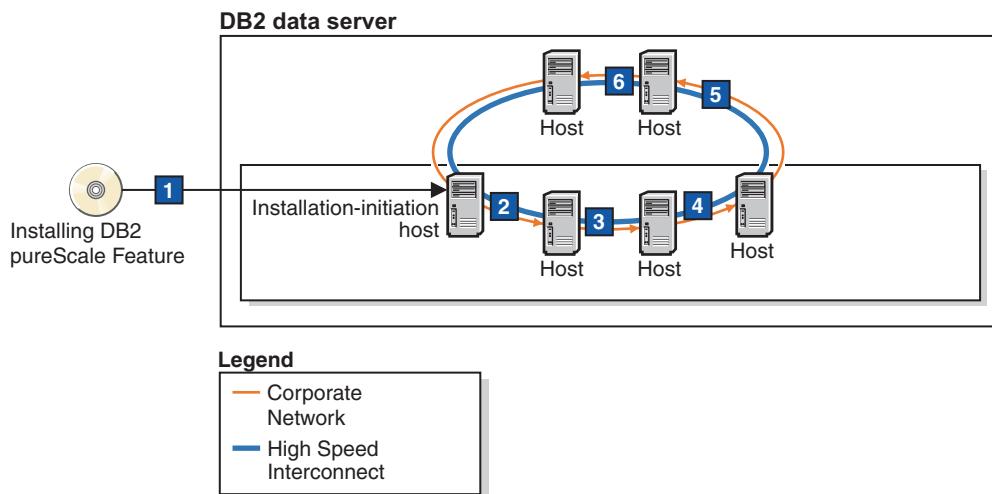


Figure 83. Installation of the DB2 pureScale Feature from the installation-initiating host (1) across all hosts (1 to 6) that will be part of the DB2 pureScale environment.

The installation process automatically creates the DB2 instance and configures all of the software components according to tested best practices; no additional user scripting or configuration is required. The first time that you start the instance, all benefits of the DB2 pureScale Feature are available.

The fix pack installation process for the DB2 pureScale Feature follows the same well-conceived paradigm of simplicity: Fix packs include all required updates for all of the components that are integrated into the feature, so multiple software installations and upgrades are not necessary. The upgrade process does not affect database availability.

Management of the DB2 pureScale Feature

The IBM DB2 pureScale Feature integrates full instance management and monitoring functionality for all components directly into the familiar DB2 database environment, protecting investment in skills and personnel.

As a database administrator, you do not have to be familiar with any of the underlying components or interact with them directly. You manage all components by using only DB2 commands, administrative views, and table functions. With a single-system view of your entire DB2 pureScale environment, you can efficiently perform the following administrative tasks from any member, with instance-wide effect:

- Backup, restore, and roll forward operations
- Instance monitoring
- Database alert management
- Database storage management
- Access management

To demonstrate how well integrated the DB2 pureScale Feature is, consider that you can also perform the following system administration tasks using simple DB2 commands:

- Remote installation and configuration of the DB2 pureScale Feature
- Creation and deletion of computing resources
- Addition and removal of hosts from the cluster manager and shared file system
- Host system maintenance

As part of the integrated cluster management provided by the DB2 cluster services, the status of all major DB2 pureScale Feature components is monitored and reacted to automatically. These components include the cluster caching facilities, the DB2 members, and the DB2 cluster file system. The ongoing, automatic management almost eliminates the need for manual intervention during component failures and reduces the amount of monitoring specific to DB2 pureScale Feature that you must perform on a regular basis. The feature includes new monitoring functionality that you can use to obtain the status of DB2 pureScale Feature components, and DB2 pureScale monitoring support is also integrated into the separately available Optim tools.

Chapter 34. DB2 audit facility

To manage access to your sensitive data, you can use a variety of authentication and access control mechanisms to establish rules and controls for acceptable data access. But to protect against and discover unknown or unacceptable behaviors you can monitor data access by using the DB2 audit facility.

Successful monitoring of unwanted data access and subsequent analysis can lead to improvements in the control of data access and the ultimate prevention of malicious or careless unauthorized access to data. The monitoring of application and individual user access, including system administration actions, can provide a historical record of activity on your database systems.

The DB2 audit facility generates, and allows you to maintain, an audit trail for a series of predefined database events. The records generated from this facility are kept in an audit log file. The analysis of these records can reveal usage patterns that would identify system misuse. Once identified, actions can be taken to reduce or eliminate such system misuse.

The audit facility provides the ability to audit at both the instance and the individual database level, independently recording all instance and database level activities with separate logs for each. The system administrator (who holds SYSADM authority) can use the **db2audit** tool to configure audit at the instance level as well as to control when such audit information is collected. The system administrator can use the **db2audit** tool to archive both instance and database audit logs as well as to extract audit data from archived logs of either type.

The security administrator (who holds SECADM authority within a database) can use audit policies in conjunction with the SQL statement, AUDIT, to configure and control the audit requirements for an individual database. The security administrator can use the following audit routines to perform the specified tasks:

- The SYSPROC.AUDIT_ARCHIVE stored procedure archives audit logs.
- The SYSPROC.AUDIT_LIST_LOGS table function allows you to locate logs of interest.
- The SYSPROC.AUDIT_DELIM_EXTRACT stored procedure extracts data into delimited files for analysis.

The security administrator can grant EXECUTE privilege on these routines to another user, therefore enabling the security administrator to delegate these tasks, if required.

When working in a partitioned database environment, many of the auditable events occur at the database partition at which the user is connected (the coordinator partition) or at the catalog partition (if they are not the same database partition). The implication of this is that audit records can be generated by more than one database partition. Part of each audit record contains information identifying the coordinator partition and originating partition (the partition where audit record originated).

At the instance level, the audit facility must be stopped and started explicitly by use of the **db2audit start** and **db2audit stop** commands. When you start instance-level auditing, the audit facility uses existing audit configuration information. Since the audit facility is independent of the DB2 database server, it

will remain active even if the instance is stopped. In fact, when the instance is stopped, an audit record may be generated in the audit log. To start auditing at the database level, first you need to create an audit policy, then you associate this audit policy with the objects you want to monitor, such as, authorization IDs, database authorities, trusted contexts or particular tables.

Categories of audit records

There are different categories of audit records that may be generated. In the following description of the categories of events available for auditing, you should notice that following the name of each category is a one-word keyword used to identify the category type. The categories of events available for auditing are:

- Audit (AUDIT). Generates records when audit settings are changed or when the audit log is accessed.
- Authorization Checking (CHECKING). Generates records during authorization checking of attempts to access or manipulate DB2 database objects or functions.
- Object Maintenance (OBJMAINT). Generates records when creating or dropping data objects, and when altering certain objects.
- Security Maintenance (SECMAINT). Generates records when:
 - Granting or revoking object privileges or database authorities
 - Granting or revoking security labels or exemptions
 - Altering the group authorization, role authorization, or override or restrict attributes of an LBAC security policy
 - Granting or revoking the SETSESSIONUSER privilege
 - Modifying any of the SYSADM_GROUP, SYSCTRL_GROUP, SYSMAINT_GROUP, or SYSMON_GROUP configuration parameters.
- System Administration (SYSADMIN). Generates records when operations requiring SYSADM, SYSMAINT, or SYSCTRL authority are performed.
- User Validation (VALIDATE). Generates records when authenticating users or retrieving system security information.
- Operation Context (CONTEXT). Generates records to show the operation context when a database operation is performed. This category allows for better interpretation of the audit log file. When used with the log's event correlator field, a group of events can be associated back to a single database operation. For example, a query statement for dynamic queries, a package identifier for static queries, or an indicator of the type of operation being performed, such as CONNECT, can provide needed context when analyzing audit results.

Note: The SQL or XQuery statement providing the operation context might be very long and is completely shown within the CONTEXT record. This can make the CONTEXT record very large.

- Execute (EXECUTE). Generates records during the execution of SQL statements.

For any of the categories listed previously, you can audit failures, successes, or both.

Any operations on the database server may generate several records. The actual number of records generated in the audit log depends on the number of categories of events to be recorded as specified by the audit facility configuration. It also depends on whether successes, failures, or both, are audited. For this reason, it is important to be selective of the events to audit.

Audit policies

The security administrator can use audit policies to configure the audit facility to gather information only about the data and objects that are needed.

The security administrator can create audit policies to control what is audited within an individual database. The following objects can have an audit policy associated with them:

- The whole database

All auditable events that occur within the database are audited according to the audit policy.

- Tables

All data manipulation language (DML) and XQUERY access to the table (untyped), MQT (materialized query table), or nickname is audited. Only EXECUTE category audit events with or without data are generated when the table is accessed even if the policy indicates that other categories should be audited.

- Trusted contexts

All auditable events that happen within a trusted connection defined by the particular trusted context are audited according to the audit policy.

- Authorization IDs representing users, groups, or roles

All auditable events that are initiated by the specified user are audited according to the audit policy.

All auditable events that are initiated by users that are a member of the group or role are audited according to the audit policy. Indirect role membership, such as through other roles or groups, is also included.

You can capture similar data by using the Work Load Management event monitors by defining a work load for a group and capturing the activity details. You should be aware that the mapping to workloads can involve attributes in addition to just the authorization ID, which can cause you to not achieve the wanted granularity in auditing, or if those other attributes are modified, connections may map to different (possibly unmonitored) workloads. The auditing solution provides a guarantee that a user, group or role will be audited.

- Authorities (SYSADM, SECADM, DBADM, SQLADM, WLMADM, ACCESSCTRL, DATAACCESS, SYSCTRL, SYSMAINT, SYSMON)

All auditable events that are initiated by a user that holds the specified authority, even if that authority is unnecessary for the event, are audited according to the audit policy.

The security administrator can create multiple audit policies. For example, your company might want a policy for auditing sensitive data and a policy for auditing the activity of users holding DBADM authority. If multiple audit policies are in effect for a statement, all events required to be audited by each of the audit policies are audited (but audited only once). For example, if the database's audit policy requires auditing successful EXECUTE events for a particular table and the user's audit policy requires auditing failures of EXECUTE events for that same table, both successful and failed attempts at accessing that table are audited.

For a specific object, there can only be one audit policy in effect. For example, you cannot have multiple audit policies associated with the same table at the same time.

An audit policy cannot be associated with a view or a typed table. Views that access a table that has an associated audit policy are audited according to the underlying table's policy.

The audit policy that applies to a table does not automatically apply to a MQT based on that table. If you associate an audit policy with a table, associate the same policy with any MQT based on that table.

Auditing performed during a transaction is done based on the audit policies and their associations at the start of the transaction. For example, if the security administrator associates an audit policy with a user and that user is in a transaction at the time, the audit policy does not affect any remaining statements performed within that transaction. Also, changes to an audit policy do not take effect until they are committed. If the security administrator issues an ALTER AUDIT POLICY statement, it does not take effect until the statement is committed.

The security administrator uses the CREATE AUDIT POLICY statement to create an audit policy, and the ALTER AUDIT POLICY statement to modify an audit policy. These statements can specify:

- The status values for events to be audited: None, Success, Failure, or Both.
Only auditable events that match the specified status value are audited.
- The server behavior when errors occur during auditing.

The security administrator uses the AUDIT statement to associate an audit policy with the current database or with a database object, at the current server. Any time the object is in use, it is audited according to this audit policy.

To delete an audit policy, the security administrator uses the DROP statement. You cannot drop an audit policy if it is associated with any object. Use the AUDIT REMOVE statement to remove any remaining association with an object. To add metadata to an audit policy, the security administrator uses the COMMENT statement.

Events generated before a full connection has been established

For some events generated during connect and a switch user operation, the only audit policy information available is the policy that is associated with the database. These events are shown in the following table:

Table 44. Connection events

Event	Audit category	Comment
CONNECT	CONTEXT	
CONNECT_RESET	CONTEXT	
AUTHENTICATION	VALIDATE	This includes authentication during both connect and switch user within a trusted connection.
CHECKING_FUNC	CHECKING	The access attempted is SWITCH_USER.

These events are audited based only on the audit policy associated with the database and not with audit policies associated with any other object such as a user, their groups, or authorities. For the CONNECT and AUTHENTICATION events that occur during connect, the instance-level audit settings are used until

the database is activated. The database is activated either during the first connection or when the ACTIVATE DATABASE command is issued.

Effect of switching user

If a user is switched within a trusted connection, no remnants of the original user are left behind. In this case, the audit policies associated with the original user are no longer considered, and the applicable audit policies are re-evaluated according to the new user. Any audit policy associated with the trusted connection is still in effect.

If a SET SESSION USER statement is used, only the session authorization ID is switched. The audit policy of the authorization ID of the original user (the system authorization ID) remains in effect and the audit policy of the new user is used as well. If multiple SET SESSION USER statements are issued within a session, only the audit policies associated with the original user (the system authorization ID) and the current user (the session authorization ID) are considered.

Data definition language restrictions

The following data definition language (DDL) statements are called AUDIT exclusive SQL statements:

- AUDIT
- CREATE AUDIT POLICY, ALTER AUDIT POLICY, and DROP AUDIT POLICY
- DROP ROLE and DROP TRUSTED CONTEXT, if the role or trusted context being dropped is associated with an audit policy

AUDIT exclusive SQL statements have some restrictions in their use:

- Each statement must be followed by a COMMIT or ROLLBACK.
- These statements cannot be issued within a global transaction, for example an XA transaction.

Only one uncommitted AUDIT exclusive DDL statement is allowed at a time across all partitions. If an uncommitted AUDIT exclusive DDL statement is executing, subsequent AUDIT exclusive DDL statements wait until the current AUDIT exclusive DDL statement commits or rolls back.

Note: Changes are written to the catalog, but do not take effect until COMMIT, even for the connection that issues the statement.

Example of auditing any access to a specific table

Consider a company where the EMPLOYEE table contains extremely sensitive information and the company wants to audit any and all SQL access to the data in that table. The EXECUTE category can be used to track all access to a table; it audits the SQL statement, and optionally the input data value provided at execution time for that statement.

There are two steps to track activity on the table. First, the security administrator creates an audit policy that specifies the EXECUTE category, and then the security administrator associates that policy with the table:

```
CREATE AUDIT POLICY SENSITIVEDATAPOLICY
    CATEGORIES EXECUTE STATUS BOTH ERROR TYPE AUDIT
    COMMIT

AUDIT TABLE EMPLOYEE USING POLICY SENSITIVEDATAPOLICY
    COMMIT
```

Example of auditing any actions by SYSADM or DBADM

In order to complete their security compliance certification, a company must show that any and all activities within the database by those people holding system administration (SYSADM) or database administrative (DBADM) authority can be monitored.

To capture all actions within the database, both the EXECUTE and SYSADMIN categories should be audited. The security administrator creates an audit policy that audits these two categories. The security administrator can use the AUDIT statement to associate this audit policy with the SYSADM and DBADM authorities. Any user that holds either SYSADM or DBADM authority will then have any auditable events logged. The following example shows how to create such an audit policy and associate it with the SYSADM and DBADM authorities:

```
CREATE AUDIT POLICY ADMINSPOLICY CATEGORIES EXECUTE STATUS BOTH,
    SYSADMIN STATUS BOTH ERROR TYPE AUDIT
    COMMIT
AUDIT SYSADM, DBADM USING POLICY ADMINSPOLICY
    COMMIT
```

Example of auditing any access by a specific role

A company has allowed its web applications access to their corporate database. The exact individuals using the web applications are unknown. Only the role that is used is known and that role is used to manage the database authorizations. The company wants to monitor the actions of anyone who is a member of that role in order to examine the requests they are submitting to the database and to ensure that they only access the database through the web applications.

The EXECUTE category contains the necessary level of auditing to track the activity of the users for this situation. The first step is to create the appropriate audit policy and associate it with the roles that are used by the web applications (in this example, the roles are TELLER and CLERK):

```
CREATE AUDIT POLICY WEBAPPPOLICY CATEGORIES EXECUTE WITH DATA
    STATUS BOTH ERROR TYPE AUDIT
    COMMIT
AUDIT ROLE TELLER, ROLE CLERK USING POLICY WEBAPPPOLICY
    COMMIT
```

Example of enabling auditing for a database

A company wants to determine who is making DDL changes (example: ALTER TABLE) on the database named SAMPLE.

```
CONNECT TO SAMPLE
```

```
CREATE AUDIT POLICY ALTPOLICY CATEGORIES AUDIT STATUS BOTH,
    OBJMAINT STATUS BOTH, CHECKING STATUS BOTH,
    EXECUTE STATUS BOTH, ERROR TYPE NORMAL
```

```
AUDIT DATABASE USING POLICY ALTPOLICY
```

Storage and analysis of audit logs

Archiving the audit log moves the active audit log to an archive directory while the server begins writing to a new, active audit log. Later, you can extract data from the archived log into delimited files and then load data from these files into DB2 database tables for analysis.

Configuring the location of the audit logs allows you to place the audit logs on a large, high-speed disk, with the option of having separate disks for each member in a multiple member environment, such as a DB2 pureScale environment or a partitioned database environment. In a multiple member environment, the path for the active audit log can be a directory that is unique to each member. Having a unique directory for each member helps to avoid file contention, because each member is writing to a different disk.

The default path for the audit logs on Windows operating systems is *instance\security\auditdata* and on Linux and UNIX operating systems is *instance/security/auditdata*. If you do not want to use the default location, you can choose different directories (you can create new directories on your system to use as alternative locations, if they do not already exist). To set the path for the active audit log location and the archived audit log location, use the **db2audit configure** command with the **datapath** and **archivepath** parameters, as shown in this example:

```
db2audit configure datapath /auditlog archivepath /auditarchive
```

The audit log storage locations you set using **db2audit** apply to all databases in the instance.

Note: If there are multiple instances on the server, then each instance should each have separate data and archive paths.

The path for active audit logs (datapath) in a multiple member environment

In a multiple member environment, the same active audit log location (set by the **datapath** parameter) must be used on each member. There are two ways to accomplish this:

1. Use database member expressions when you specify the **datapath** parameter.
Using database member expressions allows the member number to be included in the path of the audit log files and results in a different path on each database member.
2. Use a shared drive that is the same on all members.

You can use database member expressions anywhere within the value you specify for the **datapath** parameter. For example, on a three member system, where the database member number is 10, the following command:

```
db2audit configure datapath '/pathForNode $N'
```

uses the following paths:

- /pathForMember10
- /pathForMember20
- /pathForMember30

Note: You cannot use database member expressions to specify the archive log file path (**archivepath** parameter).

Archiving active audit logs

The system administrator can use the **db2audit** tool to archive both instance and database audit logs as well as to extract audit data from archived logs of either type.

The security administrator, or a user to whom the security administrator has granted EXECUTE privilege on the audit routines, can archive the active audit log by running the SYSPROC.AUDIT_ARCHIVE stored procedure. To extract data from the log and load it into delimited files, they can use the SYSPROC.AUDIT_DELIM_EXTRACT stored procedure.

These are the steps to archive and extract the audit logs using the audit routines:

1. Schedule an application to perform regular archives of the active audit log using the stored procedure SYSPROC.AUDIT_ARCHIVE.
2. Determine which archived log files are of interest. Use the SYSPROC.AUDIT_LIST_LOGS table function to list all of the archived audit logs.
3. Pass the file name as a parameter to the SYSPROC.AUDIT_DELIM_EXTRACT stored procedure to extract data from the log and load it into delimited files.
4. Load the audit data into DB2 database tables for analysis.

The archived log files do not need to be immediately loaded into tables for analysis; they can be saved for future analysis. For example, they may only need to be looked at when a corporate audit is taking place.

If a problem occurs during archive, such as running out of disk space in the archive path, or the archive path does not exist, the archive process fails and an interim log file with the file extension .bk is generated in the audit log data path, for example, db2audit.instance.log.0.20070508172043640941.bk. After the problem is resolved (by allocating sufficient disk space in the archive path, or by creating the archive path) you must move this interim log to the archive path. Then, you can treat it in the same way as a successfully archived log.

Archiving active audit logs in a multiple member environment

In a multiple member environment, if the archive command is issued while the instance is running, the archive process automatically runs on every member. The same timestamp is used in the archived log file name on all members. For example, on a three member system, where the database member number is 10, the following command:

```
db2audit archive to /auditarchive
```

creates the following files:

- /auditarchive/db2audit.log.10.timestamp
- /auditarchive/db2audit.log.20.timestamp
- /auditarchive/db2audit.log.30.timestamp

If the archive command is issued while the instance is not running, you can control on which member the archive is run by one of the following methods:

- Use the **node** option with the **db2audit** command to perform the archive for the current member only.
- Use the **db2_all** command to run the archive on all members.

For example:

```
db2_all db2audit archive node to /auditarchive
```

This sets the **DB2NODE** environment variable to indicate on which members the command is invoked.

Alternatively, you can issue an individual archive command on each member separately. For example:

- On member 10:

```
db2audit archive node 10 to /auditarchive
```

- On member 20:

```
db2audit archive node 20 to /auditarchive
```

- On member 30:

```
db2audit archive node 30 to /auditarchive
```

Note: When the instance is not running, the timestamps in the archived audit log file names are not the same on each member.

Note: It is recommended that the archive path is shared across all members, but it is not required.

Note: The AUDIT_DELIM_EXTRACT stored procedure and AUDIT_LIST_LOGS table function can only access the archived log files that are visible from the current (coordinator) member.

Example of archiving a log and extracting data to a table

To ensure their audit data is captured and stored for future use, a company needs to create a new audit log every six hours and archive the current audit log to a WORM drive. The company schedules the following call to the SYSPROC.AUDIT_ARCHIVE stored procedure to be issued every six hours by the security administrator, or by a user to whom the security administrator has granted EXECUTE privilege on the AUDIT_ARCHIVE stored procedure. The path to the archived log is the default archive path, /auditarchive, and the archive runs on all members:

```
CALL SYSPROC.AUDIT_ARCHIVE( '/auditarchive', -2 )
```

As part of their security procedures, the company has identified and defined a number of suspicious behaviors or disallowed activities that it needs to watch for in the audit data. They want to extract all the data from the one or more audit logs, place it in a relational table, and then use SQL queries to look for these activities. The company has decided on appropriate categories to audit and has associated the necessary audit policies with the database or other database objects.

For example, they can call the SYSPROC.AUDIT_DELIM_EXTRACT stored procedure to extract the archived audit logs for all categories from all members that were created with a timestamp in April 2006, using the default delimiter:

```
CALL SYSPROC.AUDIT_DELIM_EXTRACT(
  '', '', '/auditarchive', 'db2audit.%..200604%', '' )
```

In another example, they can call the SYSPROC.AUDIT_DELIM_EXTRACT stored procedure to extract the archived audit records with success events from the EXECUTE category and failure events from the CHECKING category, from a file with the timestamp they are interested in:

```
CALL SYSPROC.AUDIT_DELIM_EXTRACT( '', '', '/auditarchive',
    'db2audit.%..20060419034937', 'category
execute status success, checking status failure );
```

The EXECUTE category for auditing SQL statements

Use the EXECUTE category to accurately track the SQL statements that are issued by a user. In Version 9.5 and earlier releases, you had to use the CONTEXT category to find this information.

As part of a comprehensive security policy, a company can require the ability to retroactively go back a set number of years and analyze the effects of any particular request against certain tables in their database. To do this, a company must institute a policy of archiving their weekly backups and associated log files such that they can reconstitute the database for any chosen moment in time. Also required, is sufficient database audit information captured about every request made against the database to allow, at any future time, the replay and analysis of any request against the relevant, restored database. This requirement can cover both static and dynamic SQL statements.

This EXECUTE category captures the SQL statement text as well as the compilation environment and other values that are needed to replay the statement at a later date. For example, replaying the statement can show you exactly which rows a SELECT statement returned. In order to re-run a statement, the database tables must first be restored to their state when the statement was issued.

When you audit using the EXECUTE category, the statement text for both static and dynamic SQL is recorded, as are input parameter markers and host variables. You can configure the EXECUTE category to be audited with or without input values.

Note: Global variables are not audited.

The auditing of EXECUTE events takes place at the completion of the event (for SELECT statements this is on cursor close). The status that the event completed with is also stored. Because EXECUTE events are audited at completion, long-running queries do not immediately appear in the audit log.

Note: The preparation of a statement is not considered part of the execution. Most authorization checks are performed at prepare time (for example, SELECT privilege). This means that statements that fail during prepare due to authorization errors do not generate EXECUTE events.

Statement Value Index, Statement Value Type and Statement Value Data fields may be repeated for a given execute record. For the report format generated by the extraction, each record lists multiple values. For the delimited file format, multiple rows are used. The first row has an event type of STATEMENT and no values. Following rows have an event type of DATA, with one row for each data value associated with the SQL statement. You can use the event correlator and application ID fields to link STATEMENT and DATA rows together. The columns Statement Text, Statement Isolation Level, and Compilation Environment Description are not present in the DATA events.

The statement text and input data values that are audited are converted into the database code page when they are stored on disk (all audited fields are stored in the database code page). No error is returned if the code page of the input data is not compatible with the database code page; the unconverted data will be logged instead. Because each database has its own audit log, databases having different code pages does not cause a problem.

ROLLBACK and COMMIT are audited when executed by the application, and also when issued implicitly as part of another command, such as BIND.

After an EXECUTE event has been audited due to access to an audited table, all statements that affect which other statements are executed within a unit of work, are audited. These statements are COMMIT, ROLLBACK, ROLLBACK TO SAVEPOINT and SAVEPOINT.

Savepoint ID field

You can use the Savepoint ID field to track which statements were affected by a ROLLBACK TO SAVEPOINT statement. An ordinary DML statement (such as SELECT, INSERT, and so on) has the current savepoint ID audited. However, for the ROLLBACK TO SAVEPOINT statement, the savepoint ID that is rolled back to will be audited instead. Therefore, every statement with a savepoint ID greater than or equal to that ID will be rolled back, as demonstrated by the following example. The table shows the sequence of statements run; all events with a Savepoint ID greater than or equal to 2 will be rolled back. Only the value of 3 (from the first INSERT statement) is inserted into the table T1.

Table 45. Sequence of statements to demonstrate effect of ROLLBACK TO SAVEPOINT statement

Statement	Savepoint ID
INSERT INTO T1 VALUES (3)	1
SAVEPOINT A	2
INSERT INTO T1 VALUES (5)	2
SAVEPOINT B	3
INSERT INTO T1 VALUES (6)	3
ROLLBACK TO SAVEPOINT A	2
COMMIT	

WITH DATA option

Not all input values are audited when you specify the WITH DATA option. LOB, LONG, XML and structured type parameters appear as NULL.

Date, time, and timestamp fields are recorded in ISO format.

If WITH DATA is specified in one policy, but WITHOUT DATA is specified in another policy associated with objects involved in the execution of the SQL statement, then WITH DATA takes precedence and data is audited for that particular statement. For example, if the audit policy associated with a user specifies WITHOUT DATA, but the policy associated with a table specifies WITH DATA, when that user accesses that table, the input data used for the statement is audited.

You are not able to determine which rows were modified on a positioned-update or positioned-delete statement. Only the execution of the underlying SELECT statement is logged, not the individual FETCH. It is not possible from the EXECUTE record to determine which row the cursor is on when the statement is issued. When replaying the statement at a later time, it is only possible to issue the SELECT statement to see what range of rows may have been affected.

Example of replaying past activities

Consider in this example that as part of their comprehensive security policy, a company requires that they retain the ability to retroactively go back up to seven years to analyze the effects of any particular request against certain tables in their database. To do this, they institute a policy of archiving their weekly backups and associated log files such that they can reconstitute the database for any chosen moment in time. They require that the database audit capture sufficient information about every request made against the database to allow the replay and analysis of any request against the relevant, restored database. This requirement covers both static and dynamic SQL statements.

This example shows the audit policy that must be in place at the time the SQL statement is issued, and the steps to archive the audit logs and later to extract and analyze them.

1. Create an audit policy that audits the EXECUTE category and apply this policy to the database:

```
CREATE AUDIT POLICY STATEMENTS CATEGORIES EXECUTE WITH DATA  
    STATUS BOTH ERROR TYPE AUDIT  
    COMMIT
```

```
AUDIT DATABASE USING POLICY STATEMENTS  
    COMMIT
```

2. Regularly archive the audit log to create an archive copy.

The following statement should be run by the security administrator, or a user to whom they grant EXECUTE privilege for the SYSPROC.AUDIT_ARCHIVE stored procedure, on a regular basis, for example, once a week or once a day, depending on the amount of data logged. These archived files can be kept for whatever period is required. The AUDIT_ARCHIVE procedure is called with two input parameters: the path to the archive directory and -2, to indicate that the archive should be run on all members:

```
CALL SYSPROC.AUDIT_ARCHIVE( '/auditarchive', -2 )
```

3. The security administrator, or a user to whom they grant EXECUTE privilege for the SYSPROC.AUDIT_LIST_LOGS table function, uses AUDIT_LIST_LOGS to examine all of the available audit logs from April 2006, to determine which logs may contain the necessary data:

```
SELECT FILE FROM TABLE(SYSPROC.AUDIT_LIST_LOGS('/auditarchive'))  
    AS T WHERE FILE LIKE 'db2audit dbname.log.0.200604%'  
FILE  
-----  
...  
db2audit dbname.log.0.20060418235612  
db2audit dbname.log.0.20060419234937  
db2audit dbname.log.0.20060420235128
```

4. From this output, the security administrator observes that the necessary logs should be in one file: db2audit dbname.log.0.20060419234937. The timestamp shows this file was archived at the end of the day for the day the auditors want to see.

The security administrator, or a user to whom they grant EXECUTE privilege for the SYSPROC.AUDIT_DELIM_EXTRACT stored procedure, uses this filename as input to AUDIT_DELIM_EXTRACT to extract the audit data into delimited files. The audit data in these files can be loaded into DB2 database tables, where it can be analyzed to find the particular statement the auditors are interested in. Even though the auditors are only interested in a single SQL statement, multiple statements from the unit of work may need to be examined in case they have any impact on the statement of interest.

5. In order to replay the statement, the security administrator must take the following actions:
 - Determine the exact statement to be issued from the audit record.
 - Determine the user who issued the statement from the audit record.
 - Re-create the exact permissions of the user at the time they issued the statement, including any LBAC protection.
 - Reproduce the compilation environment, by using the compilation environment column in the audit record in combination with the SET COMPILE ENVIRONMENT statement.
 - Restore the database to its exact state at the time the statement was issued.

To avoid disturbing the production system, any restore of the database and replay of the statement should be done on a second database system. The security administrator, running as the user who issued the statement, can reissue the statement as found in the statement text with any input variables that are provided in the statement value data elements.

Part 6. Appendixes

Appendix A. DB2 technical information

DB2 technical information is available in multiple formats that can be accessed in multiple ways.

DB2 technical information is available through the following tools and methods:

- Online DB2 documentation in IBM Knowledge Center:
 - Topics (task, concept, and reference topics)
 - Sample programs
 - Tutorials
- Locally installed DB2 Information Center:
 - Topics (task, concept, and reference topics)
 - Sample programs
 - Tutorials
- DB2 books:
 - PDF files (downloadable)
 - PDF files (from the DB2 PDF DVD)
 - Printed books
- Command-line help:
 - Command help
 - Message help

Important: The documentation in IBM Knowledge Center and the DB2 Information Center is updated more frequently than either the PDF or the hardcopy books. To get the most current information, install the documentation updates as they become available, or refer to the DB2 documentation in IBM Knowledge Center.

You can access additional DB2 technical information such as technotes, white papers, and IBM Redbooks® publications online at ibm.com. Access the DB2 Information Management software library site at <http://www.ibm.com/software/data/sw-library/>.

Documentation feedback

The DB2 Information Development team values your feedback on the DB2 documentation. If you have suggestions for how to improve the DB2 documentation, send an email to db2docs@ca.ibm.com. The DB2 Information Development team reads all of your feedback but cannot respond to you directly. Provide specific examples wherever possible to better understand your concerns. If you are providing feedback on a specific topic or help file, include the topic title and URL.

Do not use the db2docs@ca.ibm.com email address to contact DB2 Customer Support. If you have a DB2 technical issue that you cannot resolve by using the documentation, contact your local IBM service center for assistance.

DB2 technical library in hardcopy or PDF format

You can download the DB2 technical library in PDF format or you can order in hardcopy from the IBM Publications Center.

English and translated DB2 Version 10.1 manuals in PDF format can be downloaded from DB2 database product documentation at www.ibm.com/support/docview.wss?rs=71&uid=swg27009474.

The following tables describe the DB2 library available from the IBM Publications Center at <http://www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss>. Although the tables identify books that are available in print, the books might not be available in your country or region.

The form number increases each time a manual is updated. Ensure that you are reading the most recent version of the manuals, as listed below.

The DB2 documentation online in IBM Knowledge Center is updated more frequently than either the PDF or the hardcopy books.

Table 46. DB2 technical information

Name	Form Number	Available in print	Last updated
<i>Administrative API Reference</i>	SC27-3864-00	Yes	April, 2012
<i>Administrative Routines and Views</i>	SC27-3865-01	No	January, 2013
<i>Call Level Interface Guide and Reference Volume 1</i>	SC27-3866-01	Yes	January, 2013
<i>Call Level Interface Guide and Reference Volume 2</i>	SC27-3867-01	Yes	January, 2013
<i>Command Reference</i>	SC27-3868-01	Yes	January, 2013
<i>Database Administration Concepts and Configuration Reference</i>	SC27-3871-01	Yes	January, 2013
<i>Data Movement Utilities Guide and Reference</i>	SC27-3869-01	Yes	January, 2013
<i>Database Monitoring Guide and Reference</i>	SC27-3887-01	Yes	January, 2013
<i>Data Recovery and High Availability Guide and Reference</i>	SC27-3870-01	Yes	January, 2013
<i>Database Security Guide</i>	SC27-3872-01	Yes	January, 2013
<i>DB2 Workload Management Guide and Reference</i>	SC27-3891-01	Yes	January, 2013
<i>Developing ADO.NET and OLE DB Applications</i>	SC27-3873-01	Yes	January, 2013
<i>Developing Embedded SQL Applications</i>	SC27-3874-01	Yes	January, 2013

Table 46. DB2 technical information (continued)

Name	Form Number	Available in print	Last updated
<i>Developing Java Applications</i>	SC27-3875-01	Yes	January, 2013
<i>Developing Perl, PHP, Python, and Ruby on Rails Applications</i>	SC27-3876-00	No	April, 2012
<i>Developing RDF Applications for IBM Data Servers</i>	SC27-4462-00	Yes	January, 2013
<i>Developing User-defined Routines (SQL and External)</i>	SC27-3877-01	Yes	January, 2013
<i>Getting Started with Database Application Development</i>	GI13-2046-01	Yes	January, 2013
<i>Getting Started with DB2 Installation and Administration on Linux and Windows</i>	GI13-2047-00	Yes	April, 2012
<i>Globalization Guide</i>	SC27-3878-00	Yes	April, 2012
<i>Installing DB2 Servers</i>	GC27-3884-01	Yes	January, 2013
<i>Installing IBM Data Server Clients</i>	GC27-3883-00	No	April, 2012
<i>Message Reference Volume 1</i>	SC27-3879-01	No	January, 2013
<i>Message Reference Volume 2</i>	SC27-3880-01	No	January, 2013
<i>Net Search Extender Administration and User's Guide</i>	SC27-3895-01	No	January, 2013
<i>Partitioning and Clustering Guide</i>	SC27-3882-01	Yes	January, 2013
<i>Preparation Guide for DB2 10.1 Fundamentals Exam 610</i>	SC27-4540-01	No	January, 2013
<i>Preparation Guide for DB2 10.1 DBA for Linux, UNIX, and Windows Exam 611</i>	SC27-4541-01	No	January, 2013
<i>pureXML Guide</i>	SC27-3892-01	Yes	January, 2013
<i>Spatial Extender User's Guide and Reference</i>	SC27-3894-00	No	April, 2012
<i>SQL Procedural Languages: Application Enablement and Support</i>	SC27-3896-01	Yes	January, 2013
<i>SQL Reference Volume 1</i>	SC27-3885-01	Yes	January, 2013
<i>SQL Reference Volume 2</i>	SC27-3886-01	Yes	January, 2013
<i>Text Search Guide</i>	SC27-3888-01	Yes	January, 2013

Table 46. DB2 technical information (continued)

Name	Form Number	Available in print	Last updated
<i>Troubleshooting and Tuning Database Performance</i>	SC27-3889-01	Yes	January, 2013
<i>Upgrading to DB2 Version 10.1</i>	SC27-3881-01	Yes	January, 2013
<i>What's New for DB2 Version 10.1</i>	SC27-3890-01	Yes	January, 2013
<i>XQuery Reference</i>	SC27-3893-01	No	January, 2013

Table 47. DB2 Connect-specific technical information

Name	Form Number	Available in print	Last updated
<i>DB2 Connect Installing and Configuring DB2 Connect Personal Edition</i>	SC27-3861-00	Yes	April, 2012
<i>Installing and Configuring DB2 Connect Servers</i>	SC27-3862-01	Yes	January, 2013
<i>DB2 Connect User's Guide</i>	SC27-3863-01	Yes	January, 2013

Displaying SQL state help from the command line processor

DB2 products return an SQLSTATE value for conditions that can be the result of an SQL statement. SQLSTATE help explains the meanings of SQL states and SQL state class codes.

Procedure

To start SQL state help, open the command line processor and enter:

`? sqlstate or ? class code`

where *sqlstate* represents a valid five-digit SQL state and *class code* represents the first two digits of the SQL state.

For example, `? 08003` displays help for the 08003 SQL state, and `? 08` displays help for the 08 class code.

Accessing different versions of the DB2 Information Center

Documentation for other versions of DB2 products is found in separate information centers on ibm.com®.

About this task

For DB2 Version 10.1 topics, the *DB2 Information Center* URL is <http://publib.boulder.ibm.com/infocenter/db2luw/v10r1>.

For DB2 Version 9.8 topics, the *DB2 Information Center* URL is <http://publib.boulder.ibm.com/infocenter/db2luw/v9r8/>.

For DB2 Version 9.7 topics, the *DB2 Information Center* URL is <http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/>.

For DB2 Version 9.5 topics, the *DB2 Information Center* URL is <http://publib.boulder.ibm.com/infocenter/db2luw/v9r5>.

For DB2 Version 9.1 topics, the *DB2 Information Center* URL is <http://publib.boulder.ibm.com/infocenter/db2luw/v9/>.

For DB2 Version 8 topics, go to the *DB2 Information Center* URL at: <http://publib.boulder.ibm.com/infocenter/db2luw/v8/>.

Updating the DB2 Information Center installed on your computer or intranet server

A locally installed DB2 Information Center must be updated periodically.

Before you begin

A DB2 Version 10.1 Information Center must already be installed. For details, see the “Installing the DB2 Information Center using the DB2 Setup wizard” topic in *Installing DB2 Servers*. All prerequisites and restrictions that applied to installing the Information Center also apply to updating the Information Center.

About this task

An existing DB2 Information Center can be updated automatically or manually:

- Automatic updates update existing Information Center features and languages. One benefit of automatic updates is that the Information Center is unavailable for a shorter time compared to during a manual update. In addition, automatic updates can be set to run as part of other batch jobs that run periodically.
- Manual updates can be used to update existing Information Center features and languages. Automatic updates reduce the downtime during the update process, however you must use the manual process when you want to add features or languages. For example, a local Information Center was originally installed with both English and French languages, and now you want to also install the German language; a manual update will install German, as well as, update the existing Information Center features and languages. However, a manual update requires you to manually stop, update, and restart the Information Center. The Information Center is unavailable during the entire update process. In the automatic update process the Information Center incurs an outage to restart the Information Center after the update only.

This topic details the process for automatic updates. For manual update instructions, see the “Manually updating the DB2 Information Center installed on your computer or intranet server” topic.

Procedure

To automatically update the DB2 Information Center installed on your computer or intranet server:

- On Linux operating systems,

- a. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the /opt/ibm/db2ic/V10.1 directory.
 - b. Navigate from the installation directory to the doc/bin directory.
 - c. Run the update-ic script:
update-ic
2. On Windows operating systems,
 - a. Open a command window.
 - b. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the <Program Files>\IBM\DB2 Information Center\Version 10.1 directory, where <Program Files> represents the location of the Program Files directory.
 - c. Navigate from the installation directory to the doc\bin directory.
 - d. Run the update-ic.bat file:
update-ic.bat

Results

The DB2 Information Center restarts automatically. If updates were available, the Information Center displays the new and updated topics. If Information Center updates were not available, a message is added to the log. The log file is located in doc\eclipse\configuration directory. The log file name is a randomly generated number. For example, 1239053440785.log.

Manually updating the DB2 Information Center installed on your computer or intranet server

If you have installed the DB2 Information Center locally, you can obtain and install documentation updates from IBM.

About this task

Updating your locally installed *DB2 Information Center* manually requires that you:

1. Stop the *DB2 Information Center* on your computer, and restart the Information Center in stand-alone mode. Running the Information Center in stand-alone mode prevents other users on your network from accessing the Information Center, and allows you to apply updates. The Workstation version of the DB2 Information Center always runs in stand-alone mode. .
2. Use the Update feature to see what updates are available. If there are updates that you must install, you can use the Update feature to obtain and install them

Note: If your environment requires installing the *DB2 Information Center* updates on a machine that is not connected to the internet, mirror the update site to a local file system by using a machine that is connected to the internet and has the *DB2 Information Center* installed. If many users on your network will be installing the documentation updates, you can reduce the time required for individuals to perform the updates by also mirroring the update site locally and creating a proxy for the update site.

If update packages are available, use the Update feature to get the packages. However, the Update feature is only available in stand-alone mode.

3. Stop the stand-alone Information Center, and restart the *DB2 Information Center* on your computer.

Note: On Windows 2008, Windows Vista (and higher), the commands listed later in this section must be run as an administrator. To open a command prompt or graphical tool with full administrator privileges, right-click the shortcut and then select **Run as administrator**.

Procedure

To update the *DB2 Information Center* installed on your computer or intranet server:

1. Stop the *DB2 Information Center*.
 - On Windows, click **Start > Control Panel > Administrative Tools > Services**. Then right-click **DB2 Information Center** service and select **Stop**.
 - On Linux, enter the following command:
`/etc/init.d/db2icdv10 stop`
2. Start the Information Center in stand-alone mode.
 - On Windows:
 - a. Open a command window.
 - b. Navigate to the path where the Information Center is installed. By default, the *DB2 Information Center* is installed in the *Program_Files\IBM\DB2 Information Center\Version 10.1* directory, where *Program_Files* represents the location of the Program Files directory.
 - c. Navigate from the installation directory to the doc\bin directory.
 - d. Run the `help_start.bat` file:
`help_start.bat`
 - On Linux:
 - a. Navigate to the path where the Information Center is installed. By default, the *DB2 Information Center* is installed in the */opt/ibm/db2ic/V10.1* directory.
 - b. Navigate from the installation directory to the doc/bin directory.
 - c. Run the `help_start` script:
`help_start`

The systems default Web browser opens to display the stand-alone Information Center.

3. Click the **Update** button (). (JavaScript must be enabled in your browser.) On the right panel of the Information Center, click **Find Updates**. A list of updates for existing documentation displays.
4. To initiate the installation process, check that the selections you want to install, then click **Install Updates**.
5. After the installation process has completed, click **Finish**.
6. Stop the stand-alone Information Center:
 - On Windows, navigate to the doc\bin directory within the installation directory, and run the `help_end.bat` file:
`help_end.bat`

Note: The `help_end` batch file contains the commands required to safely stop the processes that were started with the `help_start` batch file. Do not use **Ctrl-C** or any other method to stop `help_start.bat`.

- On Linux, navigate to the doc/bin directory within the installation directory, and run the `help_end` script:
`help_end`

Note: The `help_end` script contains the commands required to safely stop the processes that were started with the `help_start` script. Do not use any other method to stop the `help_start` script.

7. Restart the *DB2 Information Center*.

- On Windows, click **Start > Control Panel > Administrative Tools > Services**. Then right-click **DB2 Information Center** service and select **Start**.
- On Linux, enter the following command:
`/etc/init.d/db2icdv10 start`

Results

The updated *DB2 Information Center* displays the new and updated topics.

DB2 tutorials

The DB2 tutorials help you learn about various aspects of DB2 database products. Lessons provide step-by-step instructions.

Before you begin

You can view the XHTML version of the tutorial from the Information Center at <http://publib.boulder.ibm.com/infocenter/db2luw/v10r1/>.

Some lessons use sample data or code. See the tutorial for a description of any prerequisites for its specific tasks.

DB2 tutorials

To view the tutorial, click the title.

“pureXML®” in *pureXML Guide*

Set up a DB2 database to store XML data and to perform basic operations with the native XML data store.

DB2 troubleshooting information

A wide variety of troubleshooting and problem determination information is available to assist you in using DB2 database products.

DB2 documentation

In the Troubleshooting and Tuning Database Performance or the Database fundamentals section of the online DB2 documentation, you can find the following troubleshooting information:

- Information about how to isolate and identify problems by using DB2 diagnostic tools and utilities
- Solutions to some of the most common problems
- Advice to help solve other problems that you might encounter with your DB2 database products

IBM Support Portal

See the IBM Support Portal if you are experiencing problems and want help finding possible causes and solutions. The Technical Support site has links to the latest DB2 publications, TechNotes, Authorized Program Analysis Reports (APARs or bug fixes), fix packs, and other resources. You can search through this knowledge base to find possible solutions to your problems.

Access the IBM Support Portal at http://www.ibm.com/support/entry/portal/Overview/Software/Information_Management/DB2_for_Linux,_UNIX_and_Windows

Terms and conditions

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.

IBM Trademarks: IBM, the IBM logo, and ibm.com are trademarks or registered trademarks 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 www.ibm.com/legal/copytrade.shtml

Appendix B. Notices

This information was developed for products and services offered in the U.S.A. Information about non-IBM products is based on information available at the time of first publication of this document and is subject to change.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information about 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 grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte character set (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

The following paragraph does not apply to the United Kingdom or any other country/region where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements, changes, or both in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to websites not owned by IBM 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.

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 that has been exchanged, should contact:

IBM Canada Limited
U59/3600
3600 Steeles Avenue East
Markham, Ontario L3R 9Z7
CANADA

Such information may be available, subject to appropriate terms and conditions, including, in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems, and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements, or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious, and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating

platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. 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 follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. *_enter the year or years_*. All rights reserved.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks 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 www.ibm.com/legal/copytrade.shtml.

The following terms are trademarks or registered trademarks of other companies

- Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
- Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle, its affiliates, or both.
- UNIX is a registered trademark of The Open Group in the United States and other countries.
- Intel, Intel logo, Intel Inside, Intel Inside logo, Celeron, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
- Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Index

A

access plan diagrams
 description 108
 example 108
 setting preferences 109
access plans
 diagramming 106
activities
 canceling
 scenarios 422, 424
 long-running
 scenario 422
adaptive compression
 details 617
 dictionaries 623
ADC (automatic dictionary creation)
 details 624
administration notification log
 first occurrence data capture (FODC) 514
administrative tools
 Replication Center 668
agents
 configuration 585
AIX
 backups 288
 restores 288
AIX Workload Manager
 integrating DB2 workload management 409
ALTER DATABASE statement
 compatibility with online backups 318
ALTER STOGROUP statement
 compatibility with online backups 318
ALTER TABLE statement
 enabling compression 621
archive logging
 overview 285
archivepath parameter 685
archiving
 audit log files 685
 log files
 compression 287
ASYNC synchronization mode 431
asynchronous index cleanup 609
asynchronous processing 253
audit facility
 actions 679
 authorities 679
 events 679
 EXECUTE events 688
 overview 679
 policies 681
 privileges 679
 troubleshooting 525
audit logs
 archiving 685
 location 685
authorities
 audit policy 681
AUTOCONFIGURE command
 sample output 606

automatic backups
 enabling 315
automatic client reroute
 high availability disaster recovery (HADR) 475
automatic dictionary creation (ADC)
 details 624
automatic incremental restore
 limitations 298
automatic maintenance
 AUTOMAINT_SET_POLICY procedure 316
 AUTOMAINT_SET_POLICYFILE procedure 316
 backups 281, 315
 configuring 316
 overview 595
 windows 596
automatic memory tuning 590
automatic reorganization
 details 597
automatic storage
 overview 3
automatic storage databases
 use by default 4
automatic storage table spaces
 adding storage 34
 altering 34
 container names 11
 converting 13
 details 9
 dropping 34
 dropping storage paths 73
 reducing size 35
autorestart database configuration parameter
 high availability disaster recovery (HADR) 485

B

BACKUP DATABASE command
 backing up data 301
 DB2 pureScale environments 310
backup images 294
backup utility
 monitoring progress 317
 performance 317
 restrictions 301
backups
 automatic 281
 compression 630
databases
 automatic 281, 315
 incremental 295
 named pipes 308
 operating system restrictions 288
 partitioned databases 309
 storage considerations 286
 tape 306
 user exit program 286
best practices
 data redistribution 170
block indexes
 insert time clustering (ITC) tables 205
 multidimensional clustering (MDC) tables 205

block-structured devices 28
buffer pool hit ratios 61
buffer pools
 creating 65
 DB2 pureScale environments
 calculating hit ratios 62
 hit ratio overview 59
 monitoring overview 58
 temporary buffer pools 62
designing 56
dropping 68
hit ratios 61, 62
memory
 protection 57
modifying 66
overview 55
query optimization 572
temporary in DB2 pureScale instances 62

C

catalog database partitions 125
catalog tables
 stored on catalog database partition 125
chains
 job manager 102
changed functionality
 Replication Center 669
character serial devices 28
circular logging 284
classic row compression
 details 615
 dictionaries 623
clone databases
 creating
 using different storage group paths 351
CLP (command line processor)
 federated functions 656
clustering
 data
 insert time clustering tables 185
 multidimensional clustering tables 185
 tables
 insert time clustering tables 185
 multidimensional clustering tables 185
clusters
 managing
 high availability disaster recovery (HADR) 501
collocation
 table 118, 124
column expressions 199
comm_bandwidth database manager configuration parameter
 query optimization 572
command line processor (CLP)
 examples
 database rebuild sessions 365
 redirected restore sessions 344
 rollforward sessions 384
 federated functions 656
command management
 Administration Explorer 89, 96
 Object List 89, 96
commands
 db2adutl
 cross-node recovery examples 323
 db2cklog 526

commands (*continued*)
 db2dart
 INSPECT command comparison 547
 overview 546
 db2ls
 listing DB2 products and features 529
 db2pd
 examples 531
 db2pdcfg
 overview 515
 INSPECT
 db2dart command comparison 547
 invoking 91
 running 91
 running in parallel 151
 support in an Object List 89, 96
 support in the Administration Explorer 89, 96
 types of database administration 89, 96
compatibility
 partition 125
compiler rewrites
 adding implied predicates 563
 correlated subqueries 559
 merge view 557
compression
 adaptive 617
 backup 630
 classic row 615
 default system values 614
 estimating storage savings 619
 index
 details 628
 NULL values 614
 overview 613
 row
 adaptive 617
 classic 615
 overview 615
table
 column values 614
 overview 613
tables
 changing 622
 creating 619
 disabling 622
 enabling 621
 temporary tables 615, 617
 value 614
compression dictionaries
 adaptive compression 617
 automated creation 624
 classic row compression 615
 forcing creation 626
 KEEPDICTIONARY parameter 626
 multiple 627
 overview 623
 rebuilding 626
 RESETDICTIONARY parameter 626
 size reporting 627
configuration
 agent and process model 585
 databases
 HADR 493
 file system caching 22
 high availability 485
 memory 583

Configuration Advisor
 defining the scope of configuration parameters 605
 details 605
 sample output 606

configuration parameters
 Configuration Advisor for defining scope 605
 hadr_peer_window
 setting 493
 hadr_timeout
 setting 493
 logarchopt1
 cross-node recovery examples 323
 partitioned database 125
 query optimization 572
 vendoropt
 cross-node recovery examples 323

connections
 failures
 parameter setting 493

constructs
 multiple query blocks 108

containers
 SMS table spaces
 adding 144

converting tables 96

coordinator partitions
 details 117

cpuspeed configuration parameter
 query optimization effect 572

CREATE SERVER statement 656

CREATE STOGROUP statement
 compatibility with online backups 318

cross-node database recovery examples 323

D

data
 accessing
 optimization 595
 compressing 627
 distribution
 organization schemes 111
 partitioned database environments 117

organization
 overview 111

redistribution
 best practices 170
 database partition groups 148
 database partitions 162
 determining need 165
 error recovery 177
 event logging 176
 log file entries 177
 log space requirements 167
 mechanism 171
 methods 162
 overview 162, 172
 recovery 176

data defragmentation
 overview 595

data management tools
 Data Studio 83

data movement
 multidimensional tables 199

data partition elimination 224

data partitions
 adding
 procedure 234
 altering 262
 attaching
 overview 236, 261
 scenario 267
 attributes 249
 creating 257
 detach phases 251
 detaching
 overview 246, 261
 scenario 267
 dropping 255
 overview 111, 217, 233
 range definition 257
 rolling in data
 overview 236, 261
 scenario 267
 rolling out data
 overview 246, 261
 scenario 267
 rotating
 scenario 265

data recovery
 log replay delay 463

data sources 654
 description 654
 performance 663

data storage
 multi-temperature 69
 storage groups 69

Data Studio
 interface for federated systems 656
 key tasks 83

Data Studio web console
 overview 89

database administration
 commands 89, 96
 invoking commands 91
 running commands 91
 support in an Object List 89, 96
 support in the Administration Explorer 89, 96

database administration commands
 cluster caching facilities 90
 Database Partitioning Facility (DPF) 90
 DB2 pureScale Feature 90
 DB2 pureScale members 90
 partitioned databases 90

database analysis and reporting tool command
 overview 546

database configuration file
 -changing 144

database engine processes 546

database manager
 shared memory 577

database objects
 recovery history file 281
 recovery log file 281
 table space change history file 281

database partition expressions
 details 144

database partition groups
 data location determination 122
 IBMDEFAULTGROUP 160
 overview 119
 tables 160

database partition servers
dropping 160
issuing commands 148
multiple logical partitions 128
specifying 143

database partitions
adding
 overview 134
 restrictions 136
 running system 135
 stopped system (UNIX) 136
 stopped system (Windows) 138

catalog 125

changing (Windows) 146

database configuration updates 144

managing 142

overview 117

redistributing data 162

spreading data across multiple partitions 118

database_memory database configuration parameter
self-tuning 587

database-managed space (DMS)
page sizes 25
table sizes 25
table spaces
 automatic storage 13
 creating 28
 sizes 25

databases
automatic storage
 overview 4
backing up 89, 96
backups
 automated 315, 595
 strategy 281

configuring 89, 96
 high availability disaster recovery (HADR) 493

connections
 high availability disaster recovery (HADR) 493

creating 89, 96
 partitioned database environments 125

data partitioning enabling 125

designing
 overview 1
dropping 89, 96

forcing applications off 89, 96

High Availability Disaster Recovery (HADR) 89, 96

logging
 circular 284
 overview 284

nonrecoverable 281

quiescing 89, 96

rebuilding
 examples 365
 incremental backup images 362
 overview 352
 partitioned databases 363
 restrictions 364
 table space containers 356
 target image selection 357

recovering 89, 96
 strategy 281

redistributing data 162

restarting 89, 96

restoring 89, 96

rolling forward log files 89, 96

starting 89, 96

databases (*continued*)
stopping 89, 96
temporary table spaces 357
transporting schemas
 examples 377
 overview 373
 transportable objects 376
 troubleshooting 379
 unquiescing 89, 96

datapath parameter 685

DB2 documentation
available formats 695

DB2 Governor
troubleshooting 525

DB2 Information Center
updating 699, 700
versions 698

DB2 products
listing 529

DB2 pureScale 656

DB2 pureScale environments
backups 310
buffer pools
 calculating hit ratios 62
 hit rates 59
 hit ratios 59
 monitoring 58

database rollforward 392

log file management 289

log record identifiers (LRIs) 293

log sequence numbers (LSNs) 293

log stream merges 289

log streams 289

monitoring
 buffer pool hit rates 59
 buffer pool hit ratios 59
 buffer pools 58

restoring 310

DB2 pureScale Feature
application transparency 674
capacity 671
continuous availability 673
database administration commands 90
deploying 676
getting started 676
overview 671
scaling 671

DB2 pureScale instances
managing 677
monitoring
 status 677

DB2 pureScale members
cluster caching facilities
 database administration commands 90
 database administration commands 90

DB2 workload management
activities
 identifying long-running activities (example) 422

AIX Workload Manager integration 409

canceling
 activities 424

DB2 Governor
relationship 399

domains 397

frequently asked questions 399

Linux workload management integration 414

overview 397

DB2 workload management (*continued*)
 Query Patroller 399
 sample application 420
 workloads
 analyzing system slowdown (example) 421

db2_all command
 overview 149
 partitioned database environments 148
 specifying 150

db2_call_stack command 149

db2_kill command 149

DB2_PEER_WAIT_LIMIT registry variable
 high availability disaster recovery (HADR) 485

db2adutl command
 cross-node recovery examples 323

db2audit.log file 679

db2Backup API
 backing up data 301

db2cklog command
 troubleshooting 526

db2dart command
 INSPECT command comparison 547
 troubleshooting overview 546

db2diag logs
 details 509
 first occurrence data capture (FODC) information 514
 interpreting
 informational record 513
 overview 510

db2fodc command
 collecting diagnostic information 515

db2inldb command
 creating split mirror 304, 305

db2ls command
 listing installed products and features 529

db2nchg command
 changing database partition server configurations 146

db2ncrt command
 adding database partition servers 127

db2ndrop command
 dropping database partition servers 160

db2nlist command 142

db2pd command
 troubleshooting examples 531

db2pdcfg command
 collecting diagnostic information 515

db2Recover API
 recovering data 321

db2Restore API
 recovering data 339

db2Rollforward API
 applying transactions to restored backup image 383

db2val command
 validating DB2 copy 546

declustering
 partial 117, 118

deep compression
 See adaptive compression 617
 See classic row compression 615

default storage groups
 overview 69

deferred index cleanup
 monitoring 611

defragmentation
 index 643

detached data partitions
 attributes 249

detached data partitions (*continued*)
 detach phases 251
 details 246

detached table partitions
 asynchronous partition detach 253

dft_degree configuration parameter
 effect on query optimization 572

diaglevel configuration parameter
 updating 514

diagnostic information
 first occurrence data capture (FODC)
 configuring 517
 details 515
 files 514

dictionaries
 compression 623

dimensions of MDC tables 206

disaster recovery
 high availability disaster recovery (HADR)
 overview 429
 requirements 471

distributed database management system 653

distribution keys
 details 123
 partitioned database environments 160

distribution maps
 details 122

documentation
 PDF files 696
 printed 696
 terms and conditions of use 703

DPF
 database administration commands 90

DROP STOGROUP statement
 compatibility with online backups 318

dynamic queries
 setting the optimization class 568

E

environment variables
 \$RAHBUFDIR 151
 \$RAHBUFSIZE 151
 \$RAHENV 155
 rah command 155
 RAHDOTFILES 156

error messages
 partitioned databases 139

event log file 177

event monitors
 troubleshooting 525

ExampleBANK reclaiming space scenario
 converting to insert time clustering tables 216
 insert time clustering table 216

EXECUTE category
 overview 688

explain facility
 federated database queries 661, 665

export utility
 online backup compatibility 318

extents
 insert time clustering (ITC) tables 198
 multidimensional clustering tables 198
 sizes in table spaces 24

F

failback operations 504
failover
 performing 501
FCM
 memory requirements 580
 service entry syntax 140
federated databases
 description 654
 determining where queries are evaluated 661
 global analysis of queries 665
 global optimization 663
 pushdown analysis 657
 server options 657
 wrapper modules 655
 wrappers 655
federated servers 654
 description 656
federated systems
 overview 653
file systems
 caching for table spaces 22
first occurrence data capture
 see FODC 514
FODC
 data generation 525
 details 514
 subdirectories 520
formulas
 buffer pool hit ratios 61
fragment elimination
 see data partition elimination 224

G

GBPs
 relationship to local buffer pools 59

H

HADR 89, 96
 active standby database
 isolation level 459
 replay-only window 460
 cluster managers 501
 commands 475
 configuring 485
 converting to multiple standby mode 438
 data concurrency 459
 databases
 initializing 477
 failback 504
 failover
 multiple standbys 447
 performing 501
 initializing
 multiple standbys 436
 single standby 477
 load operations 485
 log archiving 494
 log flushes 459
 managing 474
 monitoring
 methods 505
 multiple standby mode 444

HADR (*continued*)

 multiple standby mode
 enabling 438
 multiple standbys 435
 overview 429
 performance 498
 primary reintegration 504
 reads on standby
 replay-only window 460
 requirements 469, 471
 restrictions 473
 rolling updates 443, 466
 rolling upgrades
 multiple standby mode 443
 setting up
 multiple standbys 436
 single standby 477
 standby databases
 initializing 480
 stopping 508
 switching database roles 504
 synchronization modes
 ASYNC 431
 effective 431, 441
 NEARSYNC 431
 operational 431, 441
 SUPERASYNC 431
 SYNC 431
 takeover
 multiple standbys 447
HADR multiple standbys
 adding auxiliary standbys 440
 changing the principal standby 440
 configuring 448
 enabling 436
 example 448
 modifying your setup 440
 monitoring 444
 NAT support 472
 overview 435
 restrictions 436
 setting up 448
 takeover
 examples 453
HADR reads on standby
 enabling 458
 overview 458
HADR standby
 log spooling 496
hadr_peer_window database configuration parameter
 automatic reconfiguration 441
 high availability disaster recovery (HADR) 485
 setting parameter 493
hadr_remote_host configuration parameter
 automatic reconfiguration 441
hadr_remote_inst configuration parameter
 automatic reconfiguration 441
hadr_remote_svc configuration parameter
 automatic reconfiguration 441
hadr_replay_delay database configuration parameter
 HADR delayed replay 463
hadr_spool_limit database configuration parameter 496
hadr_syncmode configuration parameter
 automatic reconfiguration 441
hadr_syncmode database configuration parameter
 high availability disaster recovery (HADR) 485

hadr_timeout configuration parameter
 setting parameter 493
 hadr_timeout database configuration parameter
 high availability disaster recovery (HADR) 485
 hash partitioning 118
 heaps
 configuring 583
 help
 SQL statements 698
 high availability
 configuring
 NAT 472
 designing 279
 outages
 overview 279
 high availability disaster recovery
 see HADR 429
 High Availability Disaster Recovery
 see HADR 429
 High Availability Disaster Recovery (HADR) 89, 96
 high water marks
 lowering
 automatic storage table spaces 16, 35
 DMS table spaces 16
 overview 14
 historical compression dictionary
 overview 627
 history
 job manager 102
 HP-UX
 backups 288
 restores 288

I

I/O
 table space design 26
 IBM Data Studio
 key tasks 83
 overview 83
 IBM Relational Data Replication Tools 667
 incremental backups
 details 295
 images for rebuilding databases 362
 incremental recovery
 overview 295
 incremental restores
 overview 342
 restoring from incremental backup images 297
 index compression
 details 628
 restrictions 628
 indexes
 advantages 633
 asynchronous cleanup 609, 611
 block
 insert time clustering (ITC) tables 205
 multidimensional clustering (MDC) tables 205
 clustering
 block-based comparison 185
 deferred cleanup 611
 logging for high availability disaster recovery
 (HADR) 496
 matching source table index with target table partitioned
 index 244
 migrating 272
 online defragmentation 643

indexes (*continued*)
 partitioned tables
 details 634
 performance tips 642
 planning 640
 reorganizing 89, 96
 XML
 partition changes 264
 insert time clustering (ITC) tables
 block indexes 205
 block maps 195
 creating 199
 deleting from 197
 logging 205
 moving data to 199
 overview 185
 updating 197
 INSPECT command
 db2dart comparison 547
 installation
 listing DB2 database products 529
 instances
 adding partition servers 127
 configuring 89, 96
 listing database partition servers 142
 partition servers
 changing 146
 dropping 160
 quiescing 89, 96
 starting 89, 96
 stopping 89, 96
 unquiescing 89, 96
 inter-partition query parallelism 129
 intrapartition parallelism
 details 645
 enabling 131
 enhancements 646
 optimization strategies 647
 invalid pages
 DB2 pureScale environments 59

J

job manager
 chains 102
 create jobs 102, 103
 history 102
 manage jobs 102
 notifications 102
 schedules 102
 job type
 DB2 CLP scripts 100
 SSH 100
 Executable/shell scripts 100
 ssh 100
 SQL-only scripts 100
 jobs
 job manager 102
 job type 100
 joins
 shared aggregation 557
 subquery transformation by optimizer 557

K

keys
 distribution 123
 table partitioning 217

L

large objects (LOBs)
 partitioned tables 233

LBPs
 relationship to group buffer pools 59

licenses
 partitioned database environments 117

Linux
 backup and restore operations between different operating systems and hardware platforms 288
 listing DB2 database products 529
 workload management integration with DB2 workload management 414

locklist configuration parameter
 query optimization 572

log files
 checking validity 526

log record identifiers (LRIs)
 DB2 pureScale environments 293

log sequence numbers (LSNs)
 DB2 pureScale environments 293

log spooling
 HADR configuration 496

log stream merges
 overview 289

log streams
 overview 289

logarchmeth1 configuration parameter
 high availability disaster recovery (HADR) 494

logarchmeth2 configuration parameter
 high availability disaster recovery (HADR) 494

logarchopt1 configuration parameter
 cross-node recovery examples 323

logfilsiz database configuration parameter
 high availability disaster recovery (HADR) 485

logical database partitions
 database partition servers 128, 143

logical partitions
 multiple 128

logs
 active 284
 archive logging 285
 archived
 compression 287
 audit 679
 circular logging 284
 control files 286
 databases
 overview 284
 DB2 pureScale environments 289
 including in backup image 294
 indexes 496
 log archiving 494
 log control files 286
 offline archived 285
 online archived 285
 redistribute events 177
 space requirements
 data redistribution 167
 recovery 286

logs (*continued*)

 user exit programs 286

LRIs (log record identifiers)
 DB2 pureScale environments 293

LSNs (log sequence numbers)
 DB2 pureScale environments 293

M

maintenance
 automatic 595
 windows 596

maxappls configuration parameter
 effect on memory use 575

maxcoordagents configuration parameter 575

maximum query degree of parallelism configuration parameter
 effect on query optimization 572

MDC tables
 block indexes 187, 189
 block maps 195
 column expressions as dimensions 199
 creating 199
 deferred index cleanup 611
 deleting from 197
 density of values 206
 details 185
 dimensions 206
 DMS table spaces 199
 loading 204
 logging 205
 maintaining clustering automatically 193
 moving data to 199
 partitioned tables 220
 scenarios 213
 updating 197

media failures
 logs 286

memory
 allocating
 overview 575
 parameters 580

configuring
 details 583

database manager 577

FCM buffer pool 580

partitioned database environments 593

self-tuning 587, 588

migration
 indexes 272

MON_GET_REBALANCE_STATUS table function
 monitoring progress 44

monitoring
 backups 317

buffer pools
 DB2 pureScale environments (hit rates) 59
 DB2 pureScale environments (hit ratios) 59
 DB2 pureScale environments (overview) 58

data redistribution 175

DB2 pureScale environments
 buffer pool hit rates 59
 buffer pool hit ratios 59

high availability disaster recovery (HADR)
 multiple standby mode 444
 overview 505

rah processes 152

rebalance operations 44

restores 380, 394

monotonicity 199
MQTs
behavior 229
partitioned tables 229
multi-temperature storage
overview 69, 79
scenario 427
multidimensional clustering (MDC) tables
block indexes 205
multidimensional clustering tables
See MDC tables 185
multiple logical partitions
configuring 129
multiple query blocks 108
multiple-partition databases
database partition groups 119

N

named pipes
backing up 308
NEARSYNC synchronization mode 431
Nodes
setting preferences 109
nonrecoverable databases
backup and recovery strategy 281
notices 705
notifications
job manager 102
numdb database manager configuration parameter
effect on memory use 575

O

offline archived logs 285
offline backups
compatibility with online backups 318
offline loads
compatibility with online backups 318
offline maintenance 596
online archived logs 285
online backups
compatibility with other utilities 318
online index creation
compatibility with online backups 318
online index reorganization
compatibility with online backups 318
online inspect
compatibility with online backups 318
online loads
compatibility with online backups 318
online maintenance 596
online table reorganization
compatibility with online backups 318
operations
merged by optimizer 556
moved by optimizer 556
operators
REBAL 646
optimization
backup performance 317
classes
choosing 567
details 564
setting 568
intra-partition parallelism 647

optimization (*continued*)
partitioned tables 224
query rewriting methods 556
restore performance 381
optimization classes
overview 564
optimization guidelines
overview 569
optimization profiles
overview 569
optimizer
tuning 569
overview
Data Studio web console 89

P

page validity
DB2 pureScale environments 59
pages
sizes
table spaces 25
tables 25
parallelism
intra-partition
optimization strategies 647
overview 645
intrapartition
enabling 131
non-SMP environments 645
partitioned database environments 117
recovery 321
parameters
memory allocation 580
partial declustering
overview 117
partitioned databases
backing up 309
creating 125
data redistribution 180
database administration commands 90
database partition groups 119
decorrelation of queries 559
dropping partitions 159
duplicate machine entries 143
errors when adding database partitions 139
machine list
duplicate entry elimination 143
specifying 143
overview 117, 118
partition compatibility 125
rebuilding databases 363
redistributing data 172, 176
self-tuning memory 591, 593
setting up 125
table spaces 28
partitioned tables
adding data partitions
procedure 234, 261
altering 261, 262
attaching partitions 236, 261
converting 241
creating 256, 257
data ranges 257
detached data partitions 249
detaching data partitions 246, 251, 255, 261
indexes 634

partitioned tables (*continued*)
 large objects (LOBs) 233
 loading 270
 materialized query tables (MQTs) 229
 migrating
 pre-Version 9.1 241
 tables 270
 views 270
 mismatches 241
 multidimensional clustering (MDC) tables 220
 optimization strategies 224
 overview 217
 restrictions 217, 262
 rolling in data partitions 236, 261
 rolling out data partitions 261
 scenarios
 attaching and detaching data partitions 267
 rolling in and rolling out data partitions 267
 rotating data 265
 see partitioned tables 217
partitioning keys
 overview 217
paths
 adding 72
performance
 catalog information 125
 enhancements
 relational indexes 642
 summary 553
 high availability disaster recovery (HADR) 498
 recovery 321
port number ranges
 defining
 Windows 127
predicate pushdown query optimization
 combined SQL/XQuery statements 561
predicates
 implied
 example 563
 translation by optimizer 556
prefix sequences 153
primary database connections
 disconnect 493
primary database reintegration after takeover 504
problem determination
 information available 702
 tutorials 702
procedures
 STEPWISE_REDISTRIBUTE_DBPG 173
process model
 configuration simplification 585
proxy nodes
 Tivoli Storage Manager (TSM)
 example 323
pureScale feature 656
pushdown analysis
 federated database queries 657

Q

queries
 multidimensional clustering 206
query optimization
 catalog statistics 571
 classes 564, 567
 configuration parameters 572
query rewrite
 examples 559

R

rah command
 controlling 155
 determining problems 157
 environment variables 155
 monitoring processes 152
 overview 148, 149
 prefix sequences 153
 RAHCHECKBUF environment variable 151
 RAHDOTFILES environment variable 156
 RAHOSTFILE environment variable 143
 RAHOSTLIST environment variable 143
 RAHWAITTIME environment variable 152
 recursively invoked 152
 running commands in parallel 151
 setting default environment profile 157
 specifying
 database partition server list 143
 parameter or response to prompt 150
 RAHCHECKBUF environment variable 151
 RAHDOTFILES environment variable 156
 RAHOSTFILE environment variable 143
 RAHOSTLIST environment variable 143
 RAHTREETHRESH environment variable 152
 RAHWAITTIME environment variable 152
range partitioning
 see data partitions 233
range-clustered tables
 advantages 275
 guidelines 276
 restrictions 277
 scenarios 276
ranges
 defining for data partitions 257
 restrictions 257
raw devices
 creating table spaces 28
REBAL operator 646
rebalancing
 compatibility with online backups 318
 rebalance utility
 monitoring progress 44
rebuilding compression dictionaries 626
reclaimable storage
 automatic storage table spaces 35
 compressed tables 615, 617
 details 16
records
 audit 679
RECOVER DATABASE command
 recovering data 321
recoverable databases
 details 281
recovery
 cross-node examples 323
 data redistribution errors 177
 databases
 rebuilding 352
 dropped tables 336
 incremental 295
 operating system restrictions 288
 parallel 321
 performance 321

recovery (*continued*)

- storage considerations 286
- strategy overview 281
- Tivoli Storage Manager (TSM) proxy nodes example 323

 redefining table space containers

- redirected restore operations
 - using script 348
- redirected restores
 - overview 344
 - using generated script 350
 - using script 348

 redistribution of data

- across database partitions 162
- database partition groups 148, 173
- event log file 176
- methods 162
- necessity 165
- prerequisites 166
- procedures 173
- restrictions 168

 redistribution utility

- monitoring progress 175

 registry variables

- `DB2_HADR_PEER_WAIT_LIMIT` 498
- `DB2_HADR_SORCVBUF` 498
- `DB2_HADR_SOSNDBUF` 498
- `DB2_NO_MPFA_FOR_NEW_DB` 199

 relational indexes

- advantages 633

 RENAME STOGROUP statement

- compatibility with online backups 318
- renaming storage groups 75

 REORG TABLE command

- compression dictionary maintenance options 626

 reorganization

- automatic
 - details 597
- tables
 - compatibility with online backups 318

 replay delay

- HADR configuration 463
- HADR standby 463, 464

 replication

- compression dictionaries for source tables 627
- tools 667

 Replication Center

- administrative tools 668
- changes 669

 RESTORE DATABASE command

- DB2 pureScale environments 310
- restoring data 339

 restore utility

- compatibility with online backups 318
- examples 344
- monitoring progress 380, 394
- performance 381
- redefining table space containers 344
- redirected restores
 - overview 344
- restrictions 339

 restoring

- automatic incremental
 - limitations 298
- from snapshot backup 341
- incremental 295, 297, 342
- transporting database schemas
 - examples 377
- restoring (*continued*)
 - transporting database schemas (*continued*)
 - overview 373
 - transportable objects 376
 - troubleshooting 379
 - ROLLFORWARD DATABASE command
 - applying transactions to restored backup image 383
 - DB2 pureScale environment 392
 - rollforward recovery
 - minimum recovery time 388
 - table spaces 388
 - rollforward utility
 - compatibility with online backups 318
 - examples 384
 - recovering dropped table 336
 - restrictions 383
 - rolling updates
 - performing
 - HADR environments 466
 - multiple standby mode 443
 - rolling upgrades
 - performing
 - multiple standby mode 443
 - rollout
 - deferred detaching 253
 - rollout deletion
 - deferred cleanup 611
 - routines
 - `WLM_CANCEL_ACTIVITY` example 422
 - row compression
 - estimating storage savings 619
 - overview 615
 - rebuilding compression dictionaries 626
 - See classic row compression 615
 - RUNSTATS command
 - automatic statistics collection 599
 - RUNSTATS utility
 - compatibility with online backups 318

S

Savepoint ID field 688
 scenario

- create jobs 103

 scenarios

- adding storage paths 37
- cancelling
 - activities 424
- moving a table space to a new storage group 77
- multidimensional clustering (MDC) tables 213
- rebalancing
 - after adding and dropping storage paths 42
 - after adding storage paths 37
 - after dropping storage paths 40
 - overview 37
- removing storage paths 37

 schedules

- job manager 102

 scripts

- troubleshooting 546

 security

- enhancements summary 653

 SELECT statement

- eliminating DISTINCT clauses 559

 self-tuning memory

- details 587
- disabling 589

self-tuning memory (*continued*)
 enabling 588
 monitoring 590
 overview 588
 partitioned database environments 591, 593

self-tuning memory manager
 see self-tuning memory 588

SET CURRENT QUERY OPTIMIZATION statement
 setting query optimization class 568

SET WRITE command
 compatibility with online backups 318

SIGTTIN message 150

site failures
 high availability disaster recovery (HADR) 429

snapshot backups
 performing 303
 restoring from 341

Solaris operating systems
 backups 288
 restores 288

sortheap database configuration parameter
 effect on query optimization 572

split mirrors
 backup images
 DB2 pureScale environment 305
 procedure 304
 standby databases 480
 DB2 pureScale environment 482

SQL compiler
 process details 553

SQL statements
 diagramming access plans 106
 help
 displaying 698
 invoking 91
 optimization configuration parameters 572
 rewriting 556
 running 91
 support in an Object List 89, 96
 support in the Administration Explorer 89, 96

ssh
 DB2 CLP scripts 100
 Executable/shell scripts 100

START HADR command
 starting HADR 475

Statement Value Data field 688

Statement Value Index field 688

Statement Value Type field 688

static queries
 setting optimization class 568

statistics
 collection
 automatic 599
 profiling
 overview 595
 query optimization 571

stdin 150

STEPWISE_REDISTRIBUTE_DBPG procedure
 redistributing data 173

STMM
 see self-tuning memory 588

stmtheap database configuration parameter
 effect on query optimization 572

STOP HADR command
 overview 475

stopping
 high availability disaster recovery (HADR) 508

storage
 automatic
 adding 34
 overview 4
 table spaces 9, 13

compression
 classic row 615
 indexes 628
 reclaiming storage freed 615, 617
 row 617
 table 613
 estimating savings offered by compression 619
 media failures 286
 reclaimable
 details 16
 reclaiming storage in automatic storage table spaces 35
 removing from automatic storage table spaces 73

requirements
 backup and recovery 286

storage groups
 altering 72
 attributes 70
 creating 72
 default 69
 dropping 76
 overview 69, 79
 paths
 replacing 75
 replacing paths 75

scenarios
 associating a table space 77
 moving a table space 77

storage paths 75
 adding 72
 monitoring 74

scenarios
 adding 37
 rebalancing table spaces after adding 37
 rebalancing table spaces after adding and dropping 42
 rebalancing table spaces after dropping 40
 removing 37

subqueries
 correlated 559

SUPERASYNC synchronization mode 431

switching
 database roles 504

SYNC synchronization mode 431

synchronization
 modes 431

SYSCATSPACE table spaces 33

SYSPROC.AUDIT_ARCHIVE stored procedure 685

SYSPROC.AUDIT_DELIM_EXTRACT stored procedure 685

system requirements
 high availability disaster recovery (HADR) 469

system-managed space (SMS)
 page size 25
 table spaces
 adding containers 144
 creating 28
 size 25

T

table compression
 compression dictionaries 627
 creating tables 619
 enabling 621

table compression (*continued*)
 overview 613
 removing 622

table partitions
 detaching 253
 placement 260

table space containers
 redefining in redirected restore operation 344

table space states 44

table spaces
 altering
 automatic storage 34
 associating with storage groups 77
 attributes 70
 automatic storage
 converting to use 13
 overview 9
 reducing size 35
 backing up 89, 96
 containers
 file example 28
 rebuilding databases 356
 creating
 procedure 28
 designing 7
 details 5
 device container example 28
 disk I/O considerations 26
 dropping
 procedure 53
 dropping storage paths 73
 extent sizes 24
 initial 33
 page sizes 25
 partitioned database environments 28
 rebalancing 73
 rebuilding 352, 361
 reducing size of automatic storage 35
 restoring 89, 96
 rollforward recovery 388
 rolling forward log files 89, 96
 scenarios
 moving to a new storage group 77
 rebalancing (after adding and dropping storage paths) 42
 rebalancing (after adding storage paths) 37
 rebalancing (after dropping storage paths) 40
 rebalancing (overview) 37
 states 44
 storage expansion 9
 storage management 8
 switching states 53
 temporary
 creating 32
 types
 overview 8
 without file system caching 22

tables
 adaptive compression 617
 altering
 partitioned tables 234, 255
 audit policy 681
 classic row compression 615
 clustering
 insert time 185
 collocation 118, 124

tables (*continued*)
 compression
 column value 614
 NULLS 614
 converting 270
 creating
 partitioned databases 160
 decompressing 622
 exporting data 89, 96
 importing data 89, 96
 loading data 89, 96
 materialized query 229
 migrating to partitioned tables 270
 multidimensional clustering (MDC) 185, 220
 page sizes 25
 partitioned
 materialized query tables (MQTs) 229
 multidimensional clustering (MDC) tables 220
 overview 217
 range-clustered 275
 guidelines 276
 restrictions 277
 scenarios 276
 recovering dropped tables 336
 regular
 multidimensional clustering (MDC) comparison 185
 setting integrity 89, 96
 unloading data 89, 96

TAKEOVER HADR command
 overview 475
 performing failover operations 501
 switching database roles 504

tape backups
 procedure 306

target images
 database rebuilds 357

temporary table spaces
 creating 32
 database rebuilds 357

temporary tables
 adaptive compression 617
 classic row compression 615

TEMPSPACE1 table space 33

terms and conditions
 publications 703

threads
 troubleshooting scripts 546

Tivoli Storage Manager
 recovery example 323

transports
 database schemas
 examples 377
 overview 373
 transportable objects 376
 troubleshooting 379

troubleshooting
 db2diag log file entry interpretation 510
 diagnostic data
 automatic collection 515
 configuring collection 517
 manual collection 515
 diagnostic logs 509
 gathering information 531
 log files 526
 resources 702
 tutorials 702

TRUNCATE
compatibility with online backups 318
trusted contexts
audit policies 681
tuning partition
determining 593
tutorials
list 702
problem determination 702
pureXML 702
troubleshooting 702

U

UNION ALL views
converting 270
UNIX
listing DB2 database products 529
updates
DB2 Information Center 699, 700
user exit programs
backups 286
logs 286
USERSPACE1 table space 33
utilities
invoking 91
running 91
support in an Object List 89, 96
support in the Administration Explorer 89, 96
utility management
Administration Explorer 89, 96
Object List 89, 96
utility throttling
details 609

V

validation
DB2 copies 546
value compression 614
vendoropt configuration parameter
cross-node recovery examples 323
views
merging by optimizer 557
predicate pushdown by optimizer 559
Visual Explain
appearance 109
constructs 106
diagramming access plans 106
explain data 109
nodes
appearance 109
purpose 106
running traces 106
setting preferences 109
special registers 106, 109
terminator 106
working directory 106

W

Windows
database partition additions 138
WITH DATA option
details 688

WLM_GET_SERVICE_SUBCLASS_STATS table function
examples
analyzing system slowdown 421
WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table
function
examples
identifying long-running activities 422
workloads
examples
analyzing system slowdown 421
wrappers
description 655

X

XML column path indexes
altering tables 264
XML data
partitioned indexes 634
XML indexes
altering table 264
XML region indexes
altering table 264
XPath statements
diagramming access plans 106
XQuery compiler
process details 553
XQuery statements
optimization configuration parameters 572
rewriting 556

IBM[®]

Printed in USA

SC27-5574-00



Spine information:

IBM DB2 10.1 for Linux, UNIX, and Windows

Preparation Guide for DB2 10.1 Advanced DBA for Linux, UNIX, and Windows Exam 614

