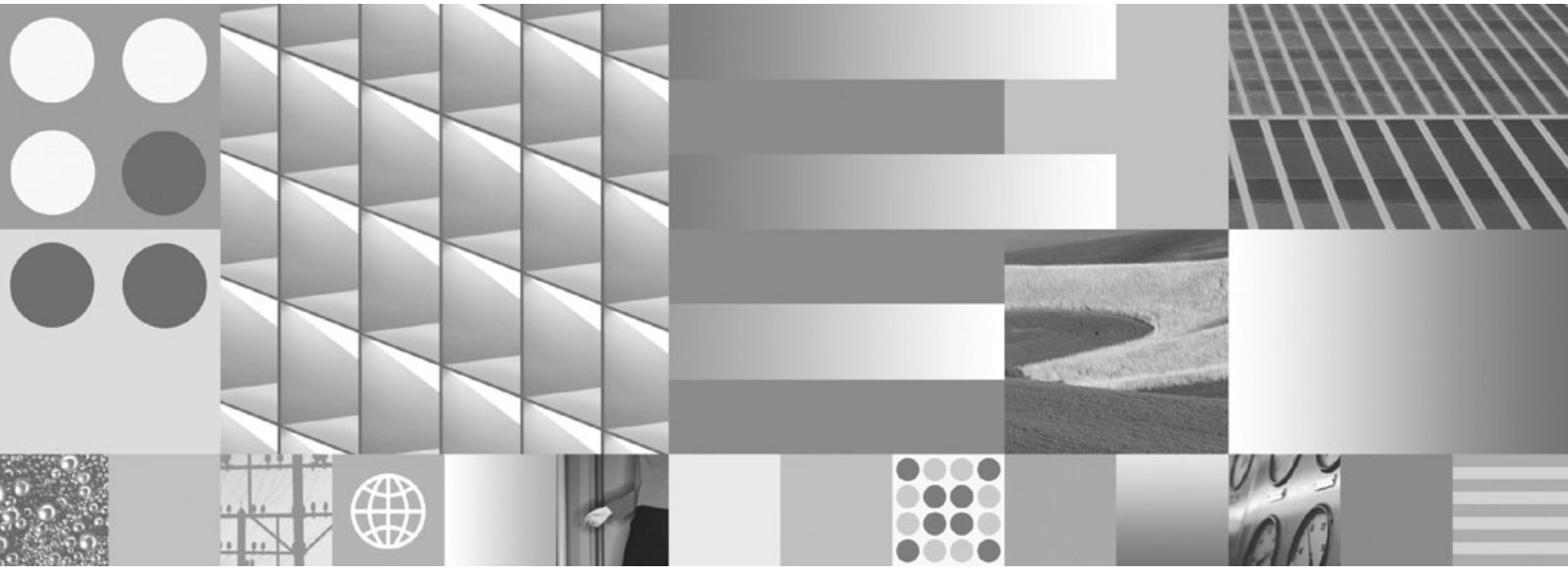
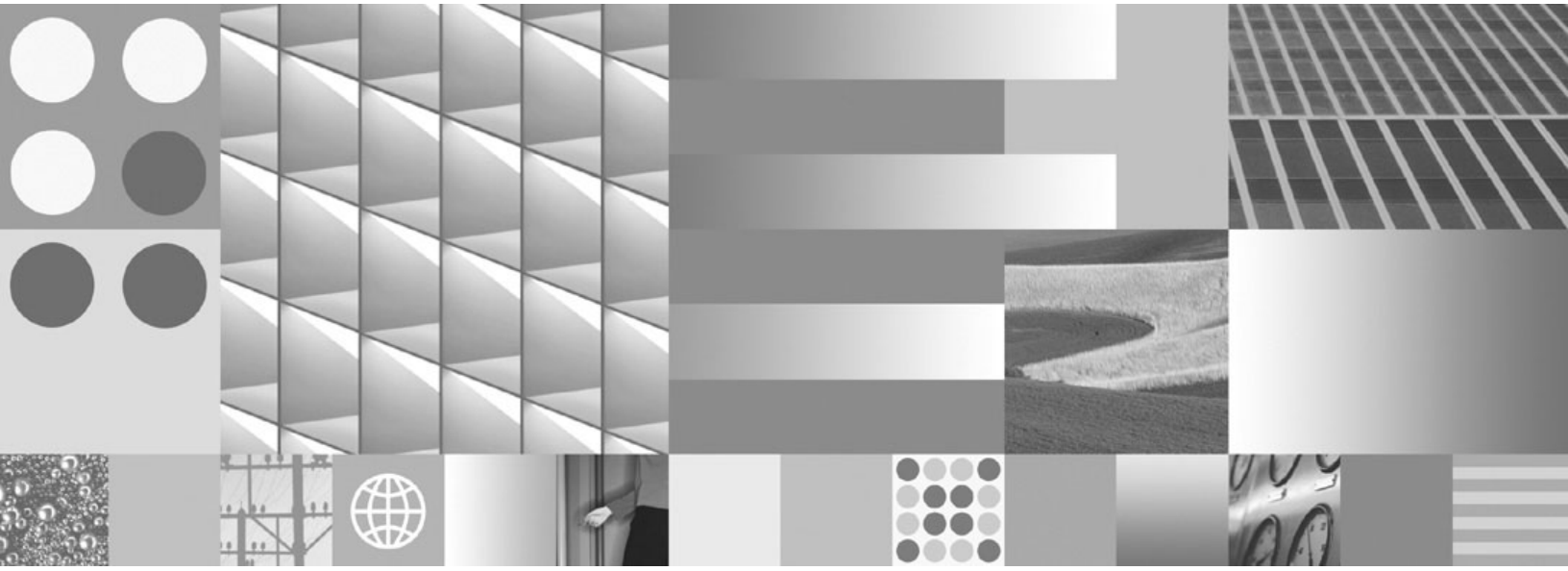
DB2 Version 9.5
for Linux, UNIX, and Windows



Partitioning and Clustering Guide
Updated April, 2009

SC23-5860-02

DB2 Version 9.5
for Linux, UNIX, and Windows



Partitioning and Clustering Guide
Updated April, 2009

SC23-5860-02

Note

Before using this information and the product it supports, read the general information under Appendix E, "Notices," on page 431.

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 www.ibm.com/shop/publications/order
- To find your local IBM representative, go to the IBM Directory of Worldwide Contacts at 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 International Business Machines Corporation 1993, 2009.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

About this book ix
Who should use this book ix
How this book is structured ix
Highlighting conventions xiii

Part 1. Planning and design considerations 1

Chapter 1. Partitioned databases and tables. 3
Setting up partitioned database environments . . . 3
Database partitioning across multiple database partitions 4
Partitioned database authentication considerations 5
Database partition groups 5
Database partition group design 7
Distribution maps 7
Distribution keys 8
Table collocation. 10
Partition compatibility 10
Partitioned tables 11
Table partitioning 11
Data partitions and ranges 13
Data organization schemes 14
Data organization schemes in DB2 and Informix databases 19
Table partitioning keys 24
Load considerations for partitioned tables . . . 26
Replicated materialized query tables 29
Table spaces in database partition groups . . . 29
Table partitioning and multidimensional clustering tables 29

Chapter 2. Range-clustered tables . . . 35
Advantages associated with a range-clustered table structure 35
Range-clustered table incompatibilities 35
Range-clustered tables and out-of-range record key values 36
Range-clustered table locks 37

Chapter 3. Multi-dimensional clustered (MDC) tables 39
Multidimensional clustering tables. 39
Comparison of regular and MDC tables 39
Choosing MDC table dimensions 41
Considerations when creating MDC tables 48
Load considerations for MDC tables 53
Logging considerations for MDC tables 54
Block index considerations for MDC tables . . . 54
Block indexes for MDC tables 55
Scenario: Multidimensional clustered (MDC) tables 57
Block indexes and query performance for MDC tables 60
Maintaining clustering automatically during INSERT operations 64
Block maps for MDC tables 66
Deleting from MDC tables 68
Updates to MDC tables 68
Table partitioning and multidimensional clustering tables 68

Chapter 4. Parallel database systems 73
Parallelism 73
Partitioned database environments 77
Database partition and processor environments . . 78

Part 2. Installation considerations 87

Chapter 5. Installation prerequisites . . 89
Installing DB2 servers (Windows) 89
Preparing the environment for a partitioned DB2 server (Windows) 91
Fast communications manager (Windows) . . . 93
An overview of installing your DB2 server product (Linux and UNIX) 93
DB2 installation methods 94
Installing DB2 servers using the DB2 Setup wizard (Linux and UNIX) 96

Chapter 6. Before you install 101
Additional partitioned database environment preinstallation tasks (Linux and UNIX) 101
Updating environment settings for a partitioned DB2 installation (AIX) 101
Setting up a working collective to distribute commands to ESE workstations (AIX) 103
Verifying that NFS is running (Linux and UNIX) 103
Verifying port range availability on participating computers (Linux and UNIX) 104
Creating a file system for a partitioned DB2 server (Linux) 105
Creating a DB2 home file system for a partitioned database system (AIX) 107
Creating required users for a DB2 server installation in a partitioned database environment (Linux) 109
Creating required users for a DB2 server installation in a partitioned database environment (AIX). 110

Chapter 7. Installing your DB2 server product 113
Setting up a partitioned database environment . . 113
Installing database partition servers on participating computers using a response file (Windows) 115

© Copyright IBM Corp. 1993, 2009

Installing database partition servers on participating computers using a response file (Linux and UNIX) 116

Chapter 8. After you install 119
Verifying the installation. 119
Verifying a partitioned database environment installation (Windows) 119
Verifying a partitioned database server installation (Linux and UNIX) 119

Part 3. Implementation and maintenance 121

Chapter 9. Before creating a database 123
Setting up partitioned database environments . . 123
Creating node configuration files 124
Format of the DB2 node configuration file. . . 126
Specifying the list of machines in a partitioned database environment 132
Eliminating duplicate entries from a list of machines in a partitioned database environment. 132
Updating the node configuration file (Linux and UNIX) 133
Setting up multiple logical nodes. 134
Configuring multiple logical partitions 135
Enabling inter-partition query parallelism 135
Enabling intra-partition parallelism for queries . . 137
Management of data server capacity. 138
Fast communications manager. 139
Fast communications manager (Windows). . . 139
Fast communications manager (Linux and UNIX) 139
Enabling communication between database partitions using FCM communications 140
Enabling communications between database partition servers (Linux and UNIX) 141

Chapter 10. Creating and managing partitioned database environments . . 143
Initial database partition groups 143
Creating database partition groups 143
Table spaces in database partition groups 144
Managing database partitions 144
Adding database partitions in partitioned database environments 146
Adding a database partition to a running database system 147
Adding a database partition to a stopped database system (Windows) 147
Adding a database partition to a stopped database system (UNIX). 148
Error recovery when adding database partitions 150
Dropping database partitions 151
Listing database partition servers in an instance 151
Adding database partition servers to an instance (Windows) 152
Adding database partitions to an instance using the Add Partitions wizard 153
Changing database partitions (Windows) . . . 153
Adding containers to SMS table spaces on database partitions 155
Dropping a database partition from an instance (Windows) 155
Dropping database partitions from the instance using the Drop Partitions launchpad 156
Scenario: Partitioning data in a database 157
Issuing commands in partitioned database environments 158
rah and db2_all commands overview 159
Specifying the rah and db2_all commands. . . 159
Running commands in parallel (Linux, UNIX) 160
Extension of the rah command to use tree logic (AIX and Solaris) 161
rah and db2_all commands. 161
rah command prefix sequences 162
Controlling the rah command 164
Specifying which . files run with rah (Linux and UNIX). 165
Determining problems with rah (Linux, UNIX) 166
Monitoring rah processes (Linux, UNIX) . . . 168
Setting the default environment profile for rah on Windows. 168

Chapter 11. Creating tables and other related table objects 169
Tables in partitioned database environments . . . 169
Large object behavior in partitioned tables. . . . 170
Creating partitioned tables 171
Defining ranges on partitioned tables 172
Migrating existing tables and views to partitioned tables 175
Partitioned materialized query table (MQT) behavior 177
Creating range-clustered tables 180
Algorithms used for range-clustered tables . . 180
Range-clustered table indexes 181
Differences from regular tables 181
Guidelines for using range-clustered tables . . 182
How the SQL compiler works with range-clustered tables 182
Scenarios: Range-clustered tables 182
Considerations when creating MDC tables. . . . 184

Chapter 12. Altering a database . . . 191
Altering an instance 191
Changing the database configuration across multiple database partitions 191
Altering a database 191
Altering a database partition group 191
Managing database partitions from the Control Center. 191

Chapter 13. Altering tables and other related table objects 193
Altering partitioned tables 193
Guidelines and restrictions on altering partitioned tables 194
Attaching data partitions 195

Guidelines for attaching data partitions to partitioned tables 196
Detaching data partitions 199
Attributes of detached data partitions 201
Adding data partitions to partitioned tables . . . 202
Dropping data partitions 203
Scenario: Rotating data in a partitioned table . . . 205
Scenarios: Rolling in and rolling out partitioned table data 206

Chapter 14. Load 211
Parallelism and loading 211
Multidimensional clustering considerations . . . 211
Load considerations for partitioned tables 212

Chapter 15. Loading data in a partitioned database environment . . 217
Load overview–partitioned database environments 217
Loading data in a partitioned database environment–hints and tips. 219
Loading data in a partitioned database environment. 220
Monitoring a load operation in a partitioned database environment using the LOAD QUERY command 225
Resuming, restarting, or terminating load operations in a partitioned database environment . 227
Load configuration options for partitioned database environments 228
Load sessions in a partitioned database environment - CLP examples 233
Migration and version compatibility. 236

Chapter 16. Migration of partitioned database environments 237
Migrating partitioned databases 237

Chapter 17. Using snapshot and event monitors 239
Using snapshot monitor data to monitor the reorganization of a partitioned table. 239
Global snapshots on partitioned database systems 246
Creating an event monitor for partitioned databases. 247

Chapter 18. Developing a good backup and recovery strategy 251
Crash recovery 251
Recovering from transaction failures in a partitioned database environment 252
Recovering from the failure of a database partition server 255
Rebuilding partitioned databases 255
Recovering data using db2adutl 256
Synchronizing clocks in a partitioned database environment. 262

Chapter 19. Troubleshooting 265
Troubleshooting DB2 265
Troubleshooting partitioned database environments 265
Issuing commands in partitioned database environments 265

Part 4. Performance issues 267

Chapter 20. Performance issues in database design 269
Performance enhancing features 269
Table partitioning and multidimensional clustering tables 269
Optimization strategies for partitioned tables 273
Optimization strategies for MDC tables. 278

Chapter 21. Indexes 281
Indexes in partitioned tables 281
Understanding index behavior on partitioned tables 281
Understanding clustering index behavior on partitioned tables 284

Chapter 22. Design advisor 287
Using the Design Advisor to migrate from a single-partition to a multiple-partition database . . 287

Chapter 23. Managing concurrency 289
Lock modes for table and RID index scans of MDC tables 289
Locking for block index scans for MDC tables . . 292
Locking behavior on partitioned tables 296

Chapter 24. Agent management . . . 299
Agents in a partitioned database 299

Chapter 25. Optimizing access plans 301
Index access and cluster ratios. 301
Table and index management for MDC tables 301
Clustering 303
Optimization strategies for intra-partition parallelism 303
Examples of db2expln and dynexpln output . . . 305
Example one: no parallelism 305
Example two: single-partition plan with intra-partition parallelism 307
Example three: multipartition plan with inter-partition parallelism 309
Example four: multipartition plan with inter-partition and intra-partition parallelism . . 311
Joins 313
Database database partition group impact on query optimization 314
Join strategies in partitioned databases 314
Join methods in partitioned database environments 316
Replicated materialized query tables in partitioned database environments 321
Lesson 4. Improving an access plan in a partitioned database environment 323
Working with access plan graphs. 323

Running a query with no indexes and no statistics in a partitioned database environment . 324
Collecting current statistics for the tables and indexes using runstats in a partitioned database environment. 327
Creating indexes on columns used to join tables in a query in a partitioned database environment. 331
Creating additional indexes on table columns in a partitioned database environment 335

Chapter 26. Data redistribution 339
Restrictions on data redistribution 339
Determining the current data distribution within a nodegroup and for a table 341
Redistributing data across database partitions using the REDISTRIBUTE DATABASE PARTITION GROUP command. 342
Redistributing data in a database partition group 343
Log space requirements for data redistribution . . 344
Redistribution event log file 345
Redistributing database partition groups using the STEPWISE_REDISTRIBUTE_DBPG procedure . . 346

Chapter 27. Configuring self-tuning memory 349
Self tuning memory in partitioned database environments 349
Using self-tuning memory in partitioned database environments 351

Chapter 28. DB2 configuration parameters and variables 353
Configuring databases across multiple partitions 353
Partitioned database environment variables . . . 354
Partitioned database environment configuration parameters 355
Communications 355
Parallel processing. 359

Part 5. Administrative APIs, commands, SQL statements . . . 363

Chapter 29. Administrative APIs . . . 365
sqleaddn - Add a database partition server to the partitioned database environment 365
sqlecran - Create a database on a database partition server 367
sqledpan - Drop a database on a database partition server 368
sqledrpn - Check whether a database partition server can be dropped 369
sqlugrpn - Get the database partition server number for a row 371

Chapter 30. Commands 375
REDISTRIBUTE DATABASE PARTITION GROUP 375
db2nchg - Change database partition server configuration 384
db2ncrt - Add database partition server to an instance 385
db2ndrop - Drop database partition server from an instance 387

Chapter 31. SQL language elements 389
Data types 389
Database partition-compatible data types . . . 389
Special registers 390
CURRENT DBPARTITIONNUM 390

Chapter 32. SQL functions 393
DATAPARTITIONNUM 393
DBPARTITIONNUM 394

Chapter 33. SQL statements 397
ALTER DATABASE PARTITION GROUP 397
CREATE DATABASE PARTITION GROUP . . . 400

Chapter 34. Supported administrative SQL routines and views 403
ADMIN_CMD stored procedure and associated administrative SQL routines 403
GET STMM TUNING DBPARTITIONNUM command using the ADMIN_CMD procedure . 403
UPDATE STMM TUNING DBPARTITIONNUM command using the ADMIN_CMD procedure . 404
Configuration administrative SQL routines and views 404
DB_PARTITIONS 404
Stepwise redistribute administrative SQL routines 405
STEPWISE_REDISTRIBUTE_DBPG procedure - Redistribute part of database partition group . . 405

Part 6. Appendixes 409

Appendix A. Install as non-root user 411
Installing a DB2 product as a non-root user . . . 411

Appendix B. Using backup 413
Using backup 413

Appendix C. Partitioned database environment catalog views 417
SYSCAT.BUFFERPOOLDBPARTITIONS 417
SYSCAT.DATAPARTITIONEXPRESSION 417
SYSCAT.DATAPARTITIONS 417
SYSCAT.DBPARTITIONGROUPDEF 418
SYSCAT.DBPARTITIONGROUPS 419
SYSCAT.PARTITIONMAPS 420

Appendix D. Overview of the DB2 technical information 421
DB2 technical library in hardcopy or PDF format 421
Ordering printed DB2 books 424

Displaying SQL state help from the command line processor 425
Accessing different versions of the DB2 Information Center 425
Displaying topics in your preferred language in the DB2 Information Center 425
Updating the DB2 Information Center installed on your computer or intranet server 426
DB2 tutorials 428
DB2 troubleshooting information 428
Terms and Conditions 428

Appendix E. Notices 431

Index 435

About this book

Welcome to the Partitioning and Clustering Guide!

The functionality of the DB2® relational database management system is significantly impacted by the partitioning and clustering features which allow administrators and system operators to effectively enhance performance of the database and to distribute many of the database objects across hardware resources. Quicker data retrieval and the ability to distribute objects across ever-growing hardware resources, to take advantage of parallelism and storage capacity, ultimately results in greater productivity. This book contains an organized collection of topics from the DB2 library resulting in a single source of comprehensive information solely focused on the planning, design, implementation, use, and maintenance of database partitions, table partitions, table clusters, table range-clusters, multi-dimensional clustering tables, and parallelism.

Who should use this book

This book is intended primarily for database administrators, system administrators, security administrators, and system operators who need to design, implement and/or maintain a partitioned and/or clustered database to be accessed by local and remote clients. It can also be used by application developers and other users who require both a comprehensive information source and an understanding of the administration and operation of the DB2 relational database management system as it pertains to the partitioning, clustering, and parallelism features. For those contemplating a future implementation of any or all of the major features discussed here, this book will serve as an excellent informational resource.

How this book is structured

This collection of topics from the DB2 library provides a single source of comprehensive information that is solely focused on the DB2 partitioning, clustering, and parallelism features. This book, for reasons of convenience and efficiency, is divided into six major parts, the first five of which represent the main administrative themes that are of concern to administrators, system operators, and application developers. A topic, contained within a major part in this book, can be mapped to a theme that represents the content of another book in the DB2 library, allowing for easy cross-referencing to more general information as it relates to a host of other DB2 features and objects. For example, after reading a topic in Part 4, Chapter 20 about how optimization strategies for multi-dimensional clustered tables exhibit improved performance, you may wish to examine other general performance enhancements on regular tables that can be configured by consulting the *Tuning Database Performance* book to which that particular example topic maps. In Table 1 below, you'll find this book's major parts mapped to other books that can be consulted for additional information about other DB2 objects and features along a similar theme.

© Copyright IBM Corp. 1993, 2009

Table 1. The mapping of this book's Parts to other books in the DB2 library

Parts in the Partitioning and Clustering Guide	**Mapping to Books in the DB2 library**
Part 1. Planning and design considerations	*Data Servers, Databases, and Database Objects Guide* *Database Security Guide*
Part 2. Installation considerations	*Data Servers, Databases, and Database Objects Guide* *Quick Beginnings for DB2 Servers*
Part 3. Implementation and maintenance	*Data Movement Utilities Guide and Reference* *Data Recovery and High Availability Guide and Reference* *Data Servers, Databases, and Database Objects Guide* *Migration Guide* *System Monitor Guide and Reference* *Troubleshooting Guide* *Visual Explain Tutorial* *XQuery Reference*
Part 4. Performance issues	*Data Servers, Databases, and Database Objects Guide* *Tuning Database Performance* *Visual Explain Tutorial*
Part 5. Administrative APIs, commands, SQL statements	*Administrative API Reference* *Administrative Routines and Views* *Command Reference* *Developing ADO.NET and OLE DB Applications* *Developing Embedded SQL Applications* *Developing Java Applications* *Developing Perl and PHP Applications* *Developing User-defined Routines (SQL and External)* *Getting Started with Database Application Development* *SQL Reference, Volume 1* *SQL Reference, Volume 2*
Part 6. Appendixes	*Data Recovery and High Availability Guide and Reference* *Quick Beginnings for DB2 Servers* *SQL Reference, Volume 1*

The major subject areas discussed in the chapters of this book are as follows:

Part 1. Planning and design considerations

All of the following chapters contain conceptual information relevant to the planning and design of databases/tables that will either be partitioned, clustered, or used in parallel database systems.

- Chapter 1, "Partitioned databases and tables," introduces relevant concepts concerning the features and benefits of partitioning databases and tables.
- Chapter 2, "Range-clustered tables," provides general conceptual information about the features and advantages to using range-clustered tables.
- Chapter 3, "Multi-dimensional clustered (MDC) tables," describes the use of multi-dimensional clustering as an elegant method for clustering data in tables.
- Chapter 4, "Parallel database systems," describes how parallelism can be exploited to dramatically enhance performance.

Part 2. Installation considerations

The following chapters provide information about the preinstallation and installation tasks that are necessary in preparation for database partitioning.

- Chapter 5, "Installation prerequisites," describes the prerequisites and restrictions concerned with preparing a DB2 server that will be involved in a partitioned database environment.
- Chapter 6, "Before you install," discusses additional preinstallation tasks and considerations in the case of UNIX® and Linux® systems.
- Chapter 7, "Installing your DB2 server product," describes how to install database partition servers and set up a partitioned database environment.
- Chapter 8, "After you install," describes how to verify the installation on Windows®, UNIX and Linux systems.

Part 3. Implementation and maintenance

Once the planning, designing, and installation steps are complete, the following chapters discuss how to implement and maintain the features and/or objects for which preparations were made earlier.

- Chapter 9, "Before creating a database," describes what should be considered before creating a database, such as enabling parallelism, creating partitioned database environments, creating and configuring nodes/partitions, and establishing communications between database partitions.
- Chapter 10, "Creating and managing partitioned database environments," describes how to create and manage database partitions and partition groups.
- Chapter 11, "Creating tables and other related table objects," presents information on how to create and set up partitioned tables, range-clustered tables, and MDC tables.
- Chapter 12, "Altering a database," describes how to alter an instance and/or a database.
- Chapter 13, "Altering tables and other related table objects," provides information on how to modify partitioned tables.
- Chapter 14, "Load," discusses load considerations in cases of parallelism, multi-dimensional clustering, and partitioned tables.

- Chapter 15, “Loading data in a partitioned database environment,” describes how to start, resume, restart or terminate data load operations in partitioned database environments.
- Chapter 16, “Migration of partitioned database environments,” briefly gives an overview about migrating partitioned databases and a reference for more detailed information.
- Chapter 17, “Using snapshot and event monitors,” gives relevant information about using snapshot monitor results to monitor table reorganization or to assess the global status of a partitioned database system, in addition to describing how to use the CREATE EVENT MONITOR statement.
- Chapter 18, “Developing a good backup and recovery strategy,” describes the concepts behind crash recovery in a partitioned database environment which will help to develop backup and recovery strategies before a failure occurs.
- Chapter 19, “Troubleshooting,” gives a brief overview of troubleshooting and useful information about how to issue commands useful in troubleshooting, such as db2trc, across all computers in the instance, or on all database partition servers.

Part 4. Performance issues

The following chapters contain pertinent information that will allow you to enhance the performance of your partitioned and/or clustered environment.

- Chapter 20, “Performance issues in database design,” describes performance enhancing features of table partitioning and multi-dimensional clustering, including optimization strategies for both.
- Chapter 21, “Indexes,” presents conceptual information that is helpful in understanding indexes on partitioned tables.
- Chapter 22, “Design advisor,” describes how to use the design advisor to obtain information about migrating from a single-partition to a multi-partition database, as well as recommendations about distributing your data and creating new indexes, materialized query tables, and multi-dimensional clustering tables.
- Chapter 23, “Managing concurrency,” provides information about lock modes.
- Chapter 24, “Agent management,” describes how to optimize database agents that are used to service application requests.
- Chapter 25, “Optimizing access plans,” describes how to improve an access plan, how the optimizer uses information from various scans to optimize data access strategies, and includes information about join strategies, all to improve performance in partitioned database environments, clustered tables, and/or systems using parallelism.
- Chapter 26, “Data redistribution,” helps you to determine if data redistribution should be done, and, if so, it describes how to redistribute data across database partitions.
- Chapter 27, “Configuring self-tuning memory,” discusses the use of the self-tuning memory feature in a partitioned database environment and provides configuration recommendations.
- Chapter 28, “DB2 configuration parameters and variables,” presents information about how to set database configuration parameters and

environmental variables across multiple partitions, and lists the parameters and variables related to partitioned database environments and the parallelism feature.

Part 5. Administrative APIs, commands, SQL statements

The following chapters collectively consolidate information about administrative APIs, commands, and SQL elements that is pertinent to partitioned database environments.

- Chapter 29, "Administrative APIs," provides information about the APIs pertinent only to partitioned database environments.
- Chapter 30, "Commands," provides information about the commands pertinent only to partitioned database environments.
- Chapter 31, "SQL language elements," presents database partition-compatible data types and special registers.
- Chapter 32, "SQL functions," describes SQL functions pertinent only to partitioned database environments.
- Chapter 33, "SQL statements," describes SQL statements pertinent only to partitioned database environments.
- Chapter 34, "Supported administrative SQL routines and views," describes SQL routines and views pertinent only to partitioned database environments.

Part 6. Appendixes

- Appendix A, "Install as non-root user," describes installing the DB2 product as a non-root user on UNIX and Linux systems.
- Appendix B, "Using backup," describes how to use the BACKUP DATABASE command.
- Appendix C, "Partitioned database environment catalog views," lists the catalog views particular to a partitioned database environment.

Highlighting conventions

The following highlighting conventions are used in this book.

Bold	Indicates commands, keywords, and other items whose names are predefined by the system.
Italics	Indicates one of the following: • Names or values (variables) that must be supplied by the user • General emphasis • The introduction of a new term • A reference to another source of information
`Monospace`	Indicates one of the following: • Files and directories • Information that you are instructed to type at a command prompt or in a window • Examples of specific data values • Examples of text similar to what might be displayed by the system • Examples of system messages • Samples of programming code

Part 1. Planning and design considerations

© Copyright IBM Corp. 1993, 2009

Chapter 1. Partitioned databases and tables

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.

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.

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 sqldbdir and is located in the sqllib 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 sqldbdir 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.

© Copyright IBM Corp. 1993, 2009

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 **stop_start_time**.

Database partitioning across multiple database partitions

The database manager allows great flexibility in spreading data across multiple database partitions (nodes) 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 should 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 may 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 collocated 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 may be using different distribution maps – they are not collocated.)
- 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.

Partitioned database authentication considerations

In a partitioned database, each partition of the database must have the same set of users and groups defined. If the definitions are not the same, the user may be authorized to do different things on different partitions. Consistency across all partitions is recommended.

Database partition groups

A database partition group is a set of one or more database partitions defined as belonging to a database. When you want to create tables for the database, you first create the database partition group where the table spaces will be stored, then you create the table space where the tables will be stored.

You can define named subsets of one or more database partitions in a database. Each subset you define is known as a *database partition group*. Each subset that contains more than one database partition is known as a *multipartition database partition group*. Multipartition database partition groups can only be defined with database partitions that belong to the same instance. A database partition group can contain as few as one database partition, or span all of the database partitions in the database.

Figure 1 on page 6 shows an example of a database with five database partitions in which:

- A database partition group spans all but one of the database partitions (Database Partition Group 1).
- A database partition group contains one database partition (Database Partition Group 2).
- A database partition group contains two database partitions. (Database Partition Group 3).
- The database partition within Database Partition Group 2 is shared (and overlaps) with Database Partition Group 1.

- There is a single database partition within Database Partition Group 3 that is shared (and overlaps) with Database Partition Group 1.

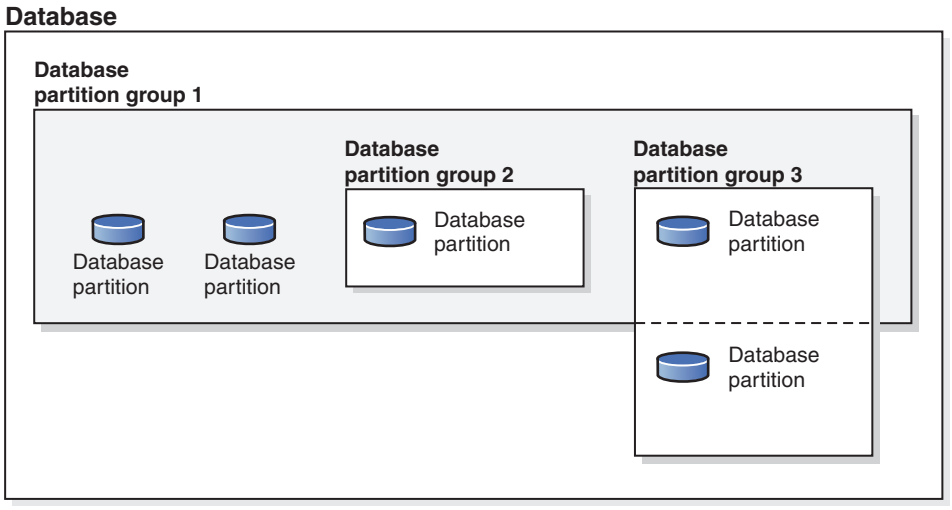


Figure 1. Database partition groups in a database

You create a new database partition group using the CREATE DATABASE PARTITION GROUP statement. You can modify it using the ALTER DATABASE PARTITION GROUP statement. Data is divided across all the database partitions in a database partition group, and you can add or drop one or more database partitions from a database partition group. If you are using a multipartition database partition group, you must look at several database partition group design considerations.

Each database partition that is part of the database system configuration must already be defined in a *database partition configuration file* called `db2nodes.cfg`. A database partition group can contain as few as one database partition, or as many as the entire set of database partitions defined for the database system.

When a database partition group is created or modified, a *distribution map* is associated with it. A distribution map, in conjunction with a *distribution key* and a hashing algorithm, is used by the database manager to determine which database partition in the database partition group will store a given row of data.

In a non-partitioned database, no distribution key or distribution map is required. A database partition is a part of the database, complete with user data, indexes, configuration files, and transaction logs. Default database partition groups that were created when the database was created are used by the database manager. IBMCATGROUP is the default database partition group for the table space containing the system catalogs. IBMTEMPGROUP is the default database partition group for system temporary table spaces. IBMDEFAULTGROUP is the default database partition group for the table spaces containing the user defined tables that you may choose to put there. A user temporary table space for a declared temporary table can be created in IBMDEFAULTGROUP or any user-created database partition group but not in IBMTEMPGROUP.

When working with database partition groups you can:

- Create a database partition group.
- Change the comment associated with a database partition group.

- Add database partitions to a database partition group.
- Drop database partitions from a database partition group.
- Redistribute table data within a database partition group.

Database partition group design

There are no database partition group design considerations if you are using a single-partition database. The DB2 Design Advisor is a tool that can be used to recommend database partition groups. The DB2 Design Advisor can be accessed from the Control Center and using db2advis from the command line processor.

If you are using a multiple partition database partition group, consider the following design points:

- In a multipartition database partition group, you can only create a unique index if it is a superset of the distribution key.
- Depending on the number of database partitions in the database, you may have one or more single-partition database partition groups, and one or more multipartition database partition groups present.
- Each database partition must be assigned a unique number. The same database partition may be found in one or more database partition groups.
- To ensure fast recovery of the database partition containing system catalog tables, avoid placing user tables on the same database partition. This is accomplished by placing user tables in database partition groups that do not include the database partition in the IBMCATGROUP database partition group.

You should 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 allow DB2 Database for Linux, UNIX, and Windows to utilize more efficient join strategies. Collocated tables can reside in a single-partition database partition group. Tables are considered collocated if they reside in a multipartition database partition group, have the same number of columns in the distribution key, and if the data types of the 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.

You should avoid extending medium-sized tables across too many database partitions. For example, a 100 MB table may 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, to ensure that the performance of OLTP transactions is not adversely affected.

Distribution maps

In a partitioned database environment, the database manager must know where to find the data 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 `4 096` 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 created on four database partitions (numbered 0–3). The distribution map for the IBMDEFAULTGROUP database partition group of this database would be:

```
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 would be:

```
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 `4 095`. That number is used as an index into the distribution map to select the database partition for that row.

Figure 2 shows how the row with the distribution key value (c1, c2, c3) is mapped to number 2, which, in turn, references database partition n5.

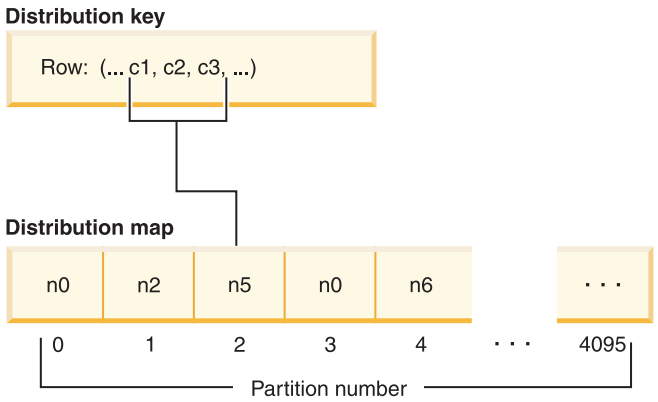


Figure 2. Data distribution using a distribution map

A distribution map is a flexible way of controlling where data is stored in a multi-partition database. If you need to 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 sqlugtpi API to obtain a copy of a distribution map that you can view.

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. You should 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. Columns with unevenly distributed data, and columns with a small number of distinct values should not be chosen as distribution keys. 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. Unnecessary columns should not be included in the distribution key.

The following points should be considered when defining distribution keys:

- Creation of a multiple-partition table that contains only long data types (LONG VARCHAR, LONG VARGRAPHIC, BLOB, CLOB, or DBCLOB) is not supported.
- The distribution key definition cannot be altered.
- The distribution key should include the most frequently joined columns.
- The distribution key should be made up of columns that often participate in a GROUP BY clause.
- Any unique key or primary key must contain all of the distribution key columns.
- In an online transaction processing (OLTP) environment, all columns in the distribution key should participate in the transaction by using equal (=) predicates with constants or host variables. For example, assume you have an employee number, *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 would make 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 4095.
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

You may discover that two or more tables frequently contribute data in response to certain queries. In this case, you will want related data from such tables to be located as close together as possible. In an environment where the database is physically divided among two or more database partitions, there must be a way to keep the related pieces of the divided tables as close together as possible. The ability to do this is called *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 may 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*.

DB2 Database for Linux, UNIX, and Windows can recognize, when accessing more than one table for a join or a subquery, that the data to be joined is located at the same database partition. When this happens, DB2 can perform the join or subquery 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 may 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.

Partitioned tables

Table partitioning functionality is available with the DB2 Version 9.1 Enterprise Server Edition for Linux, UNIX, and Windows, and later. 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, then the table is partitioned.

All the table spaces specified must have the same: pagesize, extentsize, storage mechanism (DMS, SMS), and type (REGULAR or LARGE) and all 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 32767 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.

Restrictions: Partitioned hierarchical or temporary tables, range-clustered tables, and partitioned views are not supported for use in partitioned tables. XML column types are also not supported for use in partitioned tables.

Table partitioning

Table partitioning is 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 columns. Each data partition is stored separately. These storage objects can be in different table spaces, in the same table space, or a combination of both.

Storage objects behave much like individual tables, making it easy to accomplish fast roll-in by incorporating an existing table into a partitioned table using the ALTER TABLE ... ATTACH statement. Likewise, easy roll-out is accomplished with the ALTER TABLE ... DETACH statement. Query processing can also take advantage of the separation of the data to avoid scanning irrelevant data, resulting in better query performance for many data warehouse style queries.

Table data is partitioned as specified in the PARTITION BY clause of the CREATE TABLE statement. The columns used in this definition are referred to as the table partitioning key columns.

This organization scheme can be used in isolation or in combination with other organization schemes. 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. The organization schemes include:

- DISTRIBUTE BY HASH
- PARTITION BY RANGE
- ORGANIZE BY DIMENSIONS

Table partitioning functionality is available with the DB2 Version 9.1 Enterprise Server Edition for Linux, UNIX, and Windows, and later.

Benefits of table partitioning

If any of the following circumstances apply to you and your organization, consider the numerous benefits of table partitioning:

- You have a data warehouse that would benefit from easier roll-in and roll-out of table data
- You have a data warehouse that includes large tables
- You are considering a migration to a Version 9.1 database from a previous release or a competitive database product
- You need to use Hierarchical Storage Management (HSM) solutions more effectively

Table partitioning offers easy roll-in and roll-out of table data, easier administration, flexible index placement and better query processing.

Efficient roll-in and roll-out
: Table partitioning allows for the efficient roll-in and roll-out of table data. You can achieve this by using the ATTACH PARTITION and DETACH PARTITION clauses of the ALTER TABLE statement. Rolling in partitioned table data allows a new range to be easily incorporated into a partitioned table as an additional data partition. Rolling out partitioned table data allows you to easily separate ranges of data from a partitioned table for subsequent purging or archiving.

Easier administration of large tables
: Table level administration is more flexible because you can perform administrative tasks on individual data partitions. These tasks include: detaching and reattaching of a data partition, backing up and restoring individual data partitions, and reorganizing individual indexes. Time

consuming maintenance operations can be shortened by breaking them down into a series of smaller operations. For example, backup operations can work data partition by data partition when the data partitions are placed in separate table spaces. Thus, it is possible to backup one data partition of a partitioned table at a time.

Flexible index placement
Indexes can now be placed in different table spaces allowing for more granular control of index placement. Some benefits of this new design include:

- Improved performance of drop index and online index create.
- The ability to use different values for any of the table space characteristics between each index on the table (for example, different page sizes for each index may be appropriate to ensure better space utilization).
- Reduced IO contention providing more efficient concurrent access to the index data for the table.
- When individual indexes are dropped space will immediately become available to the system without the need for an index reorganization.
- If you choose to perform index reorganization, an individual index can be reorganized.

Both DMS and SMS table spaces support the use of indexes in a different location than the table.

Improved performance for business intelligence style queries
Query processing is enhanced to automatically eliminate data partitions based on predicates of the query. This functionality, known as *Data Partition Elimination*, benefits many decision support queries.

The following example creates a table *customer* where rows with l_shipdate >= '01/01/2006' and l_shipdate <= '03/31/2006' are stored in table space ts1, rows with l_shipdate >= '04/01/2006' and l_shipdate <= '06/30/2006' are in table space ts2, etc.

```
CREATE TABLE customer (l_shipdate DATE, l_name CHAR(30))
IN ts1, ts2, ts3, ts4, ts5
PARTITION BY RANGE(l_shipdate) (STARTING FROM ('01/01/2006')
ENDING AT ('12/31/2006') EVERY (3 MONTHS))
```

Data partitions and ranges

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. 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 may 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.

Data organization schemes

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 should 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 intra-query 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)

This syntax allows consistency between the clauses as well as allowing 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 functionality.

In a single table, you can combined the clauses used in each data organization scheme to create more sophisticated partitioning schemes. For example, DB2 Database Partitioning Feature (DPF) is not only compatible, but also complementary to table partitioning.

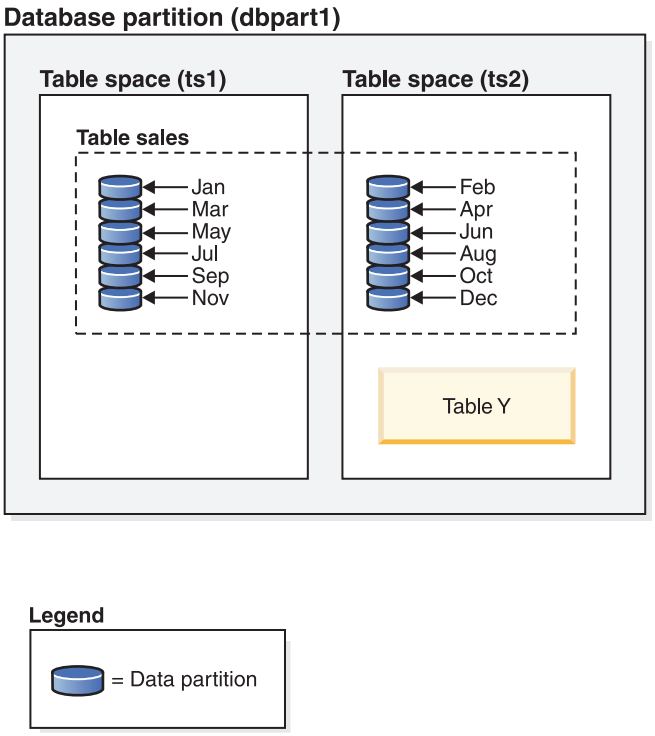


Figure 3. Demonstrating the table partitioning organization scheme where a table representing monthly sales data is partitioned into multiple data partitions. The table also spans two table spaces (ts1 and ts2).

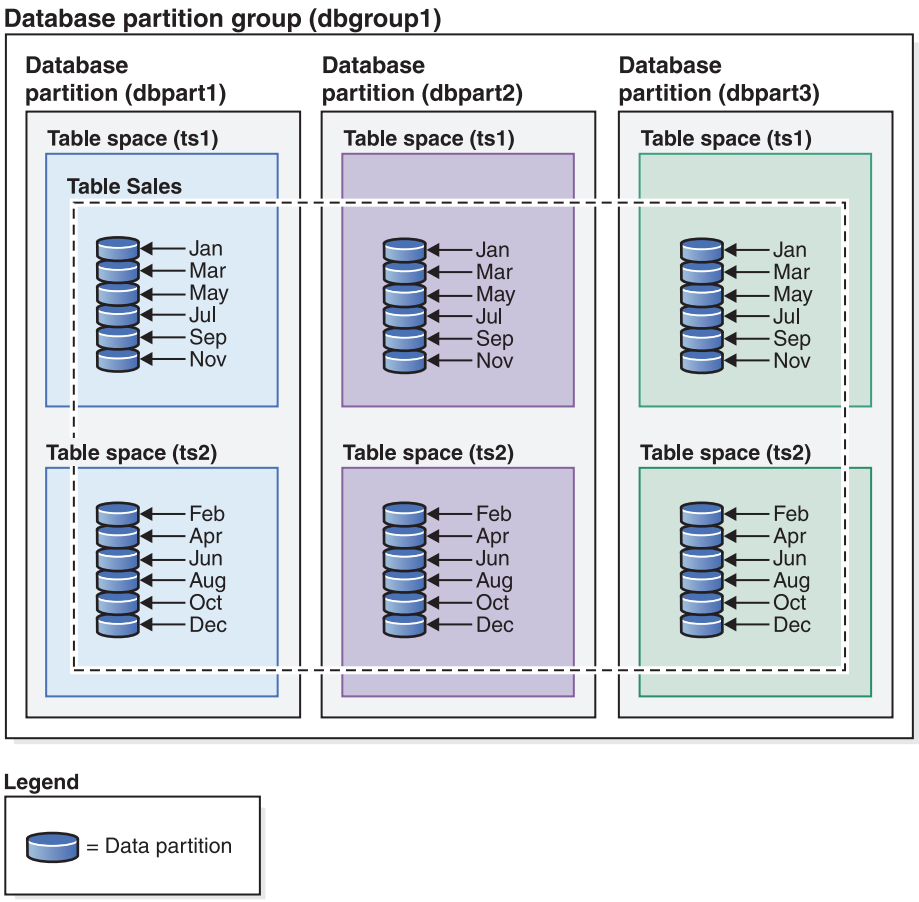


Figure 4. Demonstrating the complementary organization schemes of database partitioning and table partitioning. A table representing monthly sales data is partitioned into multiple data partitions, spanning two table spaces (ts1 and ts2) that are distributed across multiple database partitions (dbpart1, 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), while 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 5 on page 17.

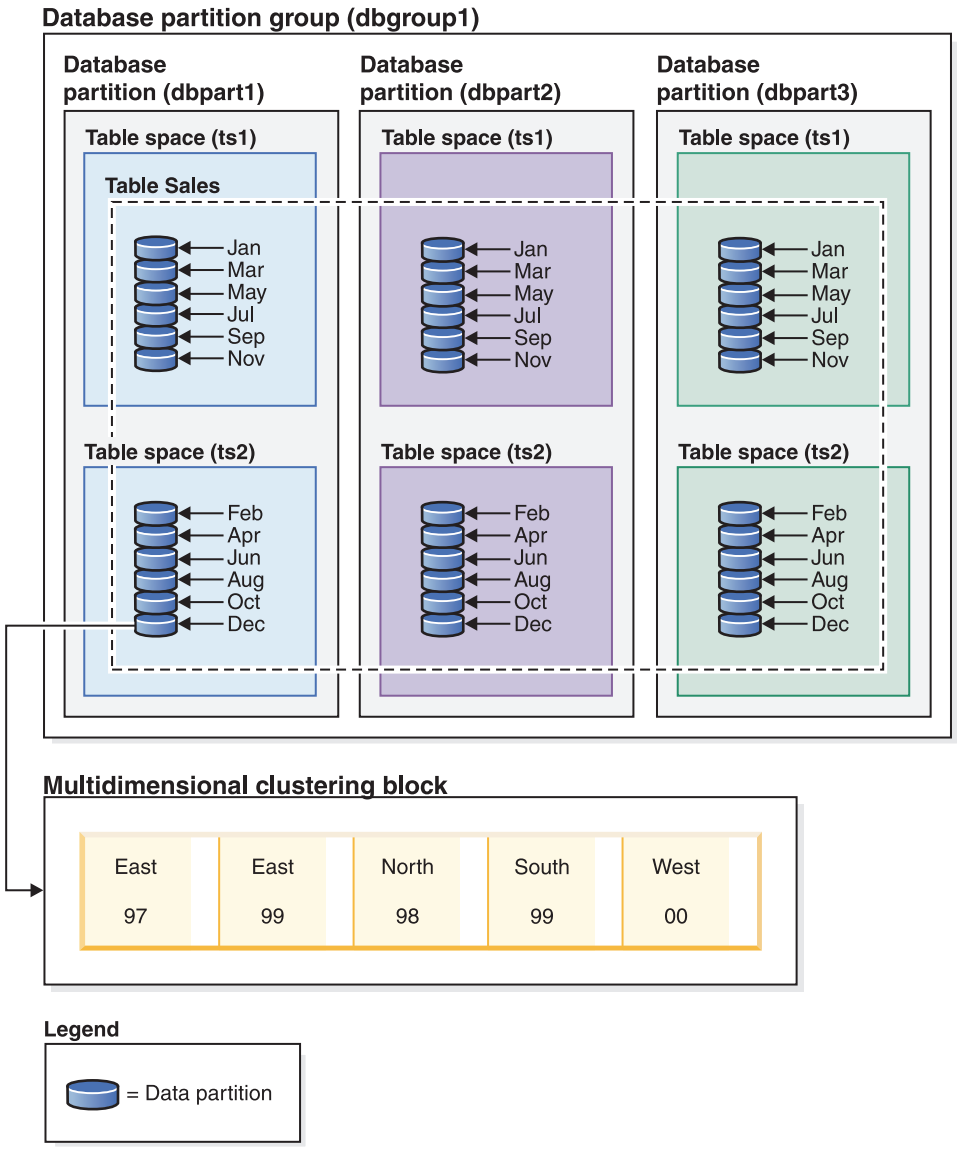


Figure 5. A representation of the database partitioning, table partitioning and multidimensional organization schemes where data from table SALES is not only distributed across multiple database partitions, partitioned across table spaces ts1 and ts2, but also groups rows with similar values on both the date and region dimensions.

There is another data organization scheme which cannot be used in conjunction with any of those listed above. 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.

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 2 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 2. 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 3. 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.

Table 3. Using table partitioning with MDC tables (continued)

Issue	Recommended scheme	Explanation
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.

Data organization schemes in DB2 and Informix databases

Table partitioning is 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 columns. Each data partition is stored separately. These storage objects can be in different table spaces, in the same table space, or a combination of both.

Table data is partitioned as specified in the PARTITION BY clause of the CREATE TABLE statement. The columns used in this definition are referred to as the table partitioning key columns. DB2 table partitioning maps to the data fragmentation approach to data organization offered by Informix Dynamic Server and Informix Extended Parallel Server.

The Informix approach

Informix supports several data organization schemes, which are called *fragmentation* in the Informix products. One of the more commonly used types of fragmentation is FRAGMENT BY EXPRESSION. This type of fragmentation works much like a CASE statement, where there is an expression associated with each fragment of the table. These expressions are checked in order to determine where to place a row.

An Informix and DB2 database system comparison

DB2 database provides a rich set of complementary features that map directly to the Informix data organization schemes, making it relatively easy for customers to convert from the Informix syntax to the DB2 syntax. The DB2 database manager handles complicated Informix schemes using a combination of generated columns and the PARTITION BY RANGE clause of the CREATE TABLE statement. Table 4 compares data organizations schemes used in Informix and DB2 database products.

Table 4. A mapping of all Informix and DB2 data organization schemes

Data organization scheme	Informix syntax	DB2 Version 9.1 syntax
• Informix: expression-based • DB2: table partitioning	FRAGMENT BY EXPRESSION	PARTITION BY RANGE
• Informix: round-robin • DB2: default	FRAGMENT BY ROUND ROBIN	No syntax: DB2 database manager automatically spreads data among containers

Table 4. A mapping of all Informix and DB2 data organization schemes (continued)

Data organization scheme	Informix syntax	DB2 Version 9.1 syntax
• Informix: range distribution • DB2: table partitioning	FRAGMENT BY RANGE	PARTITION BY RANGE
• Informix: system defined-hash • DB2: database partitioning	FRAGMENT BY HASH	DISTRIBUTE BY HASH
• Informix: HYBRID • DB2: database partitioning with table partitioning	FRAGMENT BY HYBRID	DISTRIBUTE BY HASH, PARTITION BY RANGE
• Informix: n/a • DB2: Multidimensional clustering	n/a	ORGANIZE BY DIMENSION

Examples

The following examples provide details on how to accomplish DB2 database equivalent outcomes for any Informix fragment by expression scheme.

Example 1: The following basic create table statement shows Informix fragmentation and the equivalent table partitioning syntax for a DB2 database system:

Informix syntax:

```
CREATE TABLE demo(a INT) FRAGMENT BY EXPRESSION
 a = 1 IN db1,
 a = 2 IN db2,
 a = 3 IN db3;
```

DB2 syntax:

```
CREATE TABLE demo(a INT) PARTITION BY RANGE(a)
 (STARTING(1) IN db1,
 STARTING(2) IN db2,
 STARTING(3) ENDING(3) IN db3);
```

Informix XPS supports a two-level fragmentation scheme known as hybrid where data is spread across co-servers with one expression and within the co-server with a second expression. This allows all co-servers to be active on a query (that is, there is data on all co-servers) as well as allowing the query to take advantage of data partition elimination.

The DB2 database system achieves the equivalent organization scheme to the Informix hybrid using a combination of the DISTRIBUTE BY and PARTITION BY clauses of the CREATE TABLE statement.

*Example 2:*The following example shows the syntax for the combined clauses:

Informix syntax

```
CREATE TABLE demo(a INT, b INT) FRAGMENT BY HYBRID HASH(a)
 EXPRESSION b = 1 IN dbsl1,
   b = 2 IN dbsl2;
```

DB2 syntax

```
CREATE TABLE demo(a INT, b INT) IN dbsl1, dbsl2
 DISTRIBUTE BY HASH(a),
 PARTITION BY RANGE(b) (STARTING 1 ENDING 2 EVERY 1);
```

In addition, you can use multidimensional clustering to gain an extra level of data organization:

```
CREATE TABLE demo(a INT, b INT, c INT) IN dbsl1, dbsl2
 DISTRIBUTE BY HASH(a),
 PARTITION BY RANGE(b) (STARTING 1 ENDING 2 EVERY 1)
 ORGANIZE BY DIMENSIONS(c);
```

Thus, all rows with the same value of column **a** are in the same database partition. All rows with the same value of column **b** are in the same table space. For a given value of **a** and **b**, all rows with the same value **c** are clustered together on disk. This approach is ideal for OLAP-type drill-down operations, because only one or several extents (blocks)in a single table space on a single database partition must be scanned to satisfy this type of query.

Table partitioning applied to common application problems

The following sections discuss how to apply the various features of DB2 table partitioning to common application problems. In each section, particular attention is given to best practices for mapping various Informix fragmentation schemes into equivalent DB2 table partitioning schemes.

Considerations for creating simple data partition ranges

One of the most common applications of table partitioning is to partition a large fact table based on a date key. If you need to create uniformly sized ranges of dates, consider using the automatically generated form of the CREATE TABLE syntax.

Examples

Example 1: The following example shows the automatically generated form of the syntax:

```
CREATE TABLE orders
(
 l_orderkey  DECIMAL(10,0) NOT NULL,
 l_partkey  INTEGER,
 l_suppkey  INTEGER,
 l_linenumber  INTEGER,
 l_quantity  DECIMAL(12,2),
 l_extendedprice  DECIMAL(12,2),
 l_discount  DECIMAL(12,2),
 l_tax   DECIMAL(12,2),
 l_returnflag  CHAR(1),
 l_linestatus  CHAR(1),
 l_shipdate  DATE,
 l_commitdate  DATE,
 l_receiptdate  DATE,
 l_shipinstruct  CHAR(25),
 l_shipmode  CHAR(10),
 l_comment  VARCHAR(44))
  PARTITION BY RANGE(l_shipdate)
  (STARTING '1/1/1992' ENDING '12/31/1993' EVERY 1 MONTH);
```

This creates 24 ranges, one for each month in 1992-1993. Attempting to insert a row with l_shipdate outside of that range results in an error.

Example 2: Compare the preceding example to the following Informix syntax:

```
create table orders
(
 l_orderkey  decimal(10,0) not null,
 l_partkey  integer,
 l_suppkey  integer,
 l_linenumber integer,
 l_quantity  decimal(12,2),
 l_extendedprice decimal(12,2),
 l_discount  decimal(12,2),
 l_tax   decimal(12,2),
 l_returnflag char(1),
 l_linestatus char(1),
 l_shipdate  date,
 l_commitdate date,
 l_receiptdate date,
 l_shipinstruct char(25),
 l_shipmode  char(10),
 l_comment  varchar(44)
) fragment by expression
l_shipdate < '1992-02-01' in ldbs1,
l_shipdate >= '1992-02-01' and l_shipdate < '1992-03-01' in ldbs2,
l_shipdate >= '1992-03-01' and l_shipdate < '1992-04-01' in ldbs3,
l_shipdate >= '1992-04-01' and l_shipdate < '1992-05-01' in ldbs4,
l_shipdate >= '1992-05-01' and l_shipdate < '1992-06-01' in ldbs5,
l_shipdate >= '1992-06-01' and l_shipdate < '1992-07-01' in ldbs6,
l_shipdate >= '1992-07-01' and l_shipdate < '1992-08-01' in ldbs7,
l_shipdate >= '1992-08-01' and l_shipdate < '1992-09-01' in ldbs8,
l_shipdate >= '1992-09-01' and l_shipdate < '1992-10-01' in ldbs9,
l_shipdate >= '1992-10-01' and l_shipdate < '1992-11-01' in ldbs10,
l_shipdate >= '1992-11-01' and l_shipdate < '1992-12-01' in ldbs11,
l_shipdate >= '1992-12-01' and l_shipdate < '1993-01-01' in ldbs12,
l_shipdate >= '1993-01-01' and l_shipdate < '1993-02-01' in ldbs13,
l_shipdate >= '1993-02-01' and l_shipdate < '1993-03-01' in ldbs14,
l_shipdate >= '1993-03-01' and l_shipdate < '1993-04-01' in ldbs15,
l_shipdate >= '1993-04-01' and l_shipdate < '1993-05-01' in ldbs16,
l_shipdate >= '1993-05-01' and l_shipdate < '1993-06-01' in ldbs17,
l_shipdate >= '1993-06-01' and l_shipdate < '1993-07-01' in ldbs18,
l_shipdate >= '1993-07-01' and l_shipdate < '1993-08-01' in ldbs19,
l_shipdate >= '1993-08-01' and l_shipdate < '1993-09-01' in ldbs20,
l_shipdate >= '1993-09-01' and l_shipdate < '1993-10-01' in ldbs21,
l_shipdate >= '1993-10-01' and l_shipdate < '1993-11-01' in ldbs22,
l_shipdate >= '1993-11-01' and l_shipdate < '1993-12-01' in ldbs23,
l_shipdate >= '1993-12-01' and l_shipdate < '1994-01-01' in ldbs24,
l_shipdate >= '1994-01-01' in ldbs25;
```

Notice that the Informix syntax provides an open ended range at the top and bottom to catch dates that are not in the expected range. The DB2 syntax can be modified to match the Informix syntax by adding ranges that make use of MINVALUE and MAXVALUE.

Example 3: The following example modifies Example 1 to mirror the Informix syntax::

```
CREATE TABLE orders
(
 l_orderkey  DECIMAL(10,0) NOT NULL,
 l_partkey  INTEGER,
 l_suppkey  INTEGER,
 l_linenumber INTEGER,
 l_quantity  DECIMAL(12,2),
 l_extendedprice DECIMAL(12,2),
 l_discount  DECIMAL(12,2),
 l_tax   DECIMAL(12,2),
 l_returnflag CHAR(1),
 l_linestatus CHAR(1),
```

```
 l_shipdate  DATE,
 l_commitdate DATE,
 l_receiptdate DATE,
 l_shipinstruct CHAR(25),
 l_shipmode  CHAR(10),
 l_comment  VARCHAR(44)
) PARTITION BY RANGE(l_shipdate)
 (STARTING MINVALUE,
  STARTING '1/1/1992' ENDING '12/31/1993' EVERY 1 MONTH,
  ENDING MAXVALUE);
```

This technique allows any date to be inserted into the table.

Partition by expression using generated columns

Although DB2 database does not directly support partitioning by expression, partitioning on a generated column is supported, making it possible to achieve the same result.

Consider the following usage guidelines before deciding whether to use this approach:

- The generated column is a real column that occupies physical disk space. Tables that make use of a generated column can be slightly larger.
- Altering the generated column expression for the column on which a partitioned table is partitioned is not supported. Attempting to do so will result in the message SQL0190. Adding a new data partition to a table that uses generated columns in the manner described in the next section generally requires you to alter the expression that defines the generated column. Altering the expression that defines a generated column is not currently supported.
- There are limitations on when you can apply data partition elimination when a table uses generated columns.

Examples

Example 1: The following uses the Informix syntax, where it is appropriate to use generated columns. In this example, the column to be partitioned on holds Canadian provinces and territories. Because the list of provinces is unlikely to change, the generated column expression is unlikely to change.

```
CREATE TABLE customer (
 cust_id  INT,
 cust_prov CHAR(2))
 FRAGMENT BY EXPRESSION
  cust_prov = "AB" IN dbspace_ab
  cust_prov = "BC" IN dbspace_bc
  cust_prov = "MB" IN dbspace_mb
  ...
  cust_prov = "YT" IN dbspace_yt
  REMAINDER IN dbspace_remainder;
```

Example 2: In this example, the DB2 table is partitioned using a generated column:

```
CREATE TABLE customer (
 cust_id     INT,
 cust_prov   CHAR(2),
 cust_prov_gen GENERATED ALWAYS AS (CASE
  WHEN cust_prov = 'AB' THEN 1
  WHEN cust_prov = 'BC' THEN 2
  WHEN cust_prov = 'MB' THEN 3
  ...
  WHEN cust_prov = 'YT' THEN 13
```

```
 ELSE 14 END))
IN tbspace_ab, tbspace_bc, tbspace_mb, .... tbspace_remainder
PARTITION BY RANGE (cust_prov_gen)
 (STARTING 1 ENDING 14 EVERY 1);
```

Here the expressions within the case statement match the corresponding expressions in the FRAGMENT BY EXPRESSION clause. The case statement maps each original expression to a number, which is stored in the generated column (cust_prov_gen in this example). This column is a real column stored on disk, so the table could occupy slightly more space than would be necessary if DB2 supported partition by expression directly. This example uses the short form of the syntax. Therefore, the table spaces in which to place the data partitions must be listed in the IN clause of the CREATE TABLE statement. Using the long form of the syntax requires a separate IN clause for each data partition.

Note: This technique can be applied to any FRAGMENT BY EXPRESSION clause.

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 ranges to match the data roll-in size. It is most common to partition data on a date or time column.
- Define range granularity to match data roll-out. It is most common to use month or quarter.
- Partition on a column that provides advantages in partition elimination.

Supported data types

Table 5 shows the data types (including synonyms) that are supported for use as a table partitioning key column:

Table 5. Supported data types

Data type column 1	Data type column 2
SMALLINT	INTEGER
INT	BIGINT
FLOAT	REAL
DOUBLE	DECIMAL
DEC	NUMERIC
NUM	CHARACTER
CHAR	VARCHAR
DATE	TIME
GRAPHIC	VARGRAPHIC
CHARACTER VARYING	TIMESTAMP

Table 5. Supported data types (continued)

Data type column 1	Data type column 2
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 appear 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

The XML data type is not supported in a partitioned table.

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 “Considerations when creating MDC tables” on page 48.

Load considerations for partitioned tables

All of the existing load features are supported when the target table is partitioned with the exception of the following general restrictions:

- Consistency points are not supported when the number of partitioning agents is greater than one.
- Loading data into a subset of data partitions while the remaining data partitions remain fully online is not supported.
- The exception table used by a load operation cannot be partitioned.
- A unique index cannot be rebuilt when the load utility is running in insert mode or restart mode, and the load target table has any detached dependents.
- Similar to loading MDC tables, exact ordering of input data records is not preserved when loading partitioned tables. Ordering is only maintained within the cell or data partition.
- Load operations utilizing multiple formatters on each database partition only preserve approximate ordering of input records. Running a single formatter on each database partition, groups the input records by cell or table partitioning key. To run a single formatter on each database partition, explicitly request CPU_PARALLELISM of 1.

General load behavior
The load utility inserts data records into the correct data partition. There is no requirement to use an external utility, such as a splitter, to partition the input data before loading.

The load utility does not access any detached or attached data partitions. Data is inserted into visible data partitions only. Visible data partitions are neither attached nor detached. In addition, a load replace operation does not truncate detached or attached data partitions. Since the load utility acquires locks on the catalog system tables, the load utility waits for any uncommitted ALTER TABLE transactions. Such transactions acquire an exclusive lock on the relevant rows in the catalog tables, and the exclusive lock must terminate before the load operation can proceed. This means that there can be no uncommitted ALTER TABLE ...ATTACH, DETACH, or ADD PARTITION transactions while load operation is running. Any input source records destined for an attached or detached data partition are rejected, and can be retrieved from the exception table if one is specified. An informational message is written to the message file to indicate some of the target table data partitions were in an attached or detached state. Locks on the relevant catalog table rows corresponding to the target table prevent users from changing the partitioning of the target table by issuing any ALTER TABLE ...ATTACH, DETACH, or ADD PARTITION operations while the load utility is running.

Handling of invalid rows
When the load utility encounters a record that does not belong to any of the visible data partitions the record is rejected and the load utility continues processing. The number of records rejected because of the range constraint violation is not explicitly displayed, but is included in the overall number of rejected records. Rejecting a record because of the range violation does not increase the number of row warnings. A single message (SQL0327N) is written to the load utility message file indicating that range violations are found, but no per-record messages are logged. In addition to all columns of the target table, the exception table includes columns describing the type of violation that had occurred for a particular row. Rows containing invalid data, including data that cannot be partitioned, are written to the dump file.

Because exception table inserts are expensive, you can control which constraint violations are inserted into the exception table. For instance, the default behavior of the load utility is to insert rows that were rejected because of a range constraint or unique constraint violation, but were otherwise valid, into the exception table. You can turn off this behavior by specifying, respectively, NORANGEEXC or NOUNIQUEEXC with the FOR EXCEPTION clause. If you specify that these constraint violations should not be inserted into the exception table, or you do not specify an exception table, information about rows violating the range constraint or unique constraint is lost.

History file
If the target table is partitioned, the corresponding history file entry does not include a list of the table spaces spanned by the target table. A different operation granularity identifier ('R' instead of 'T') indicates that a load operation ran against a partitioned table.

Terminating a load operation
Terminating a load replace completely truncates all visible data partitions, terminating a load insert truncates all visible data partitions to their lengths before the load. Indexes are invalidated during a termination of an ALLOW READ ACCESS load operation that failed in the load copy phase. Indexes are also

invalidated when terminating an ALLOW NO ACCESS load operation that touched the index (It is invalidated because the indexing mode is rebuild, or a key was inserted during incremental maintenance leaving the index in an inconsistent state). Loading data into multiple targets does not have any effect on load recovery operations except for the inability to restart the load operation from a consistency point taken during the load phase In this case, the SAVECOUNT load option is ignored if the target table is partitioned. This behavior is consistent with loading data into a MDC target table.

Generated columns
If a generated column is in any of the partitioning, dimension, or distribution keys, the `generatedoverride` file type modifier is ignored and the load utility generates values as if the `generatedignore` file type modifier is specified. Loading an incorrect generated column value in this case can place the record in the wrong physical location, such as the wrong data partition, MDC block or database partition. For example, once a record is on a wrong data partition, set integrity has to move it to a different physical location, which cannot be accomplished during online set integrity operations.

Data availability
The current ALLOW READ ACCESS load algorithm extends to partitioned tables. An ALLOW READ ACCESS load operation allows concurrent readers to access the whole table, including both loading and non-loading data partitions.

Data partition states
After a successful load, visible data partitions might change to either or both Set Integrity Pending or Read Access Only table state, under certain conditions. Data partitions might be placed in these states if there are constraints on the table which the load operation cannot maintain. Such constraints might include check constraints and detached materialized query tables. A failed load operation leaves all visible data partitions in the Load Pending table state.

Error isolation
Error isolation at the data partition level is not supported. Isolating the errors means continuing a load on data partitions that did not run into an error and stopping on data partitions that did run into an error. Errors can be isolated between different database partitions, but the load utility cannot commit transactions on a subset of visible data partitions and roll back the remaining visible data partitions.

Other considerations

- Incremental indexing is not supported if any of the indexes are marked invalid. An index is considered invalid if it requires a rebuild or if detached dependents require validation with the SET INTEGRITY statement.
- Loading into tables partitioned using any combination of partitioned by range, distributed by hash, or organized by dimension algorithms is also supported.
- For log records which include the list of object and table space IDs affected by the load, the size of these log records (LOAD START and COMMIT (PENDING LIST)) could grow considerably and hence reduce the amount of active log space available to other applications.
- When a table is both partitioned and distributed, a partitioned database load might not affect all database partitions. Only the objects on the output database partitions are changed.

- During a load operation, memory consumption for partitioned tables increases with the number of tables. Note, that the total increase is not linear as only a small percentage of the overall memory requirement is proportional to the number of data partitions.

Replicated materialized query tables

A *materialized query table* is a table that is defined by a query that is also used to determine the data in the table. Materialized query tables can be used to improve the performance of queries. If DB2 Database for Linux, UNIX, and Windows determines that a portion of a query could be resolved using a materialized query table, the query may be rewritten by the database manager to use the materialized query table.

In a partitioned database environment, you can replicate materialized query tables and use them to improve query performance. A *replicated materialized query table* is based on a table that may have been created in a single-partition database partition group, but that you want replicated across all of the database partitions in another database partition group. To create the replicated materialized query table, invoke the CREATE TABLE statement with the REPLICATED keyword.

By using replicated materialized query tables, you can obtain collocation between tables that are not typically collocated. Replicated materialized query tables are particularly useful for joins in which you have a large fact table and small dimension tables. To minimize the extra storage required, as well as the impact of having to update every replica, tables that are to be replicated should be small and updated infrequently.

Note: You should also consider replicating larger tables that are updated infrequently: the one-time cost of replication is offset by the performance benefits that can be obtained through collocation.

By specifying a suitable predicate in the subselect clause used to define the replicated table, you can replicate selected columns, selected rows, or both.

Table spaces in database partition groups

By placing a table space in a multiple-partition database partition group, all of the tables within the table space are divided or partitioned across each database partition in the database partition group.

The table space is created into a database partition group. Once in a database partition group, the table space must remain there; it cannot be changed to another database partition group. The CREATE TABLESPACE statement is used to associate a table space with a database partition group.

Table partitioning and multidimensional clustering tables

A table can be both multi-dimensional clustered and partitioned. In a table that is both multi-dimensional clustered and partitioned, columns can be used both in the table partitioning range-partition-spec and in the MDC key. This is useful for achieving 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 multi-column, while MDC is multi-dimension.

Characteristics of a mainstream DB2 V9.1 data warehouse

The following recommendations were focused on typical, mainstream warehouses that are new for DB2 V9.1. The following characteristics are assumed:

- The database runs on multiple machines or multiple AIX logical partitions.
- Database partitioning feature (DPF) is 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 million to 100 billion 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 6. Using table partitioning with MDC tables

Issue	Recommended scheme	Recommendation
Data availability during roll-out	Table partitioning	Use the DETACH PARTITION clause to roll out large amounts of data with minimal disruption.
Query performance	Table partitioning and MDC	MDC is best for querying multiple dimensions. Table partitioning helps through data partition elimination.

Table 6. Using table partitioning with MDC tables (continued)

Issue	Recommended scheme	Recommendation
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 (either ALLOW READ ACCESS or ALLOW NO ACCESS)	Table partitioning and MDC	MDC provides most of the benefit here. Table partitioning provides incremental benefits.
Load data ″continually″ (ALLOW READ ACCESS)	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, range partitioning helps reduce the need for reorg by maintaining some course 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 6 on page 32.

```
CREATE TABLE orders (YearAndMonth INT, Province CHAR(2))
PARTITION BY RANGE (YearAndMonth)
(STARTING 9901 ENDING 9904 EVERY 2)
ORGANIZE BY (Province);
```

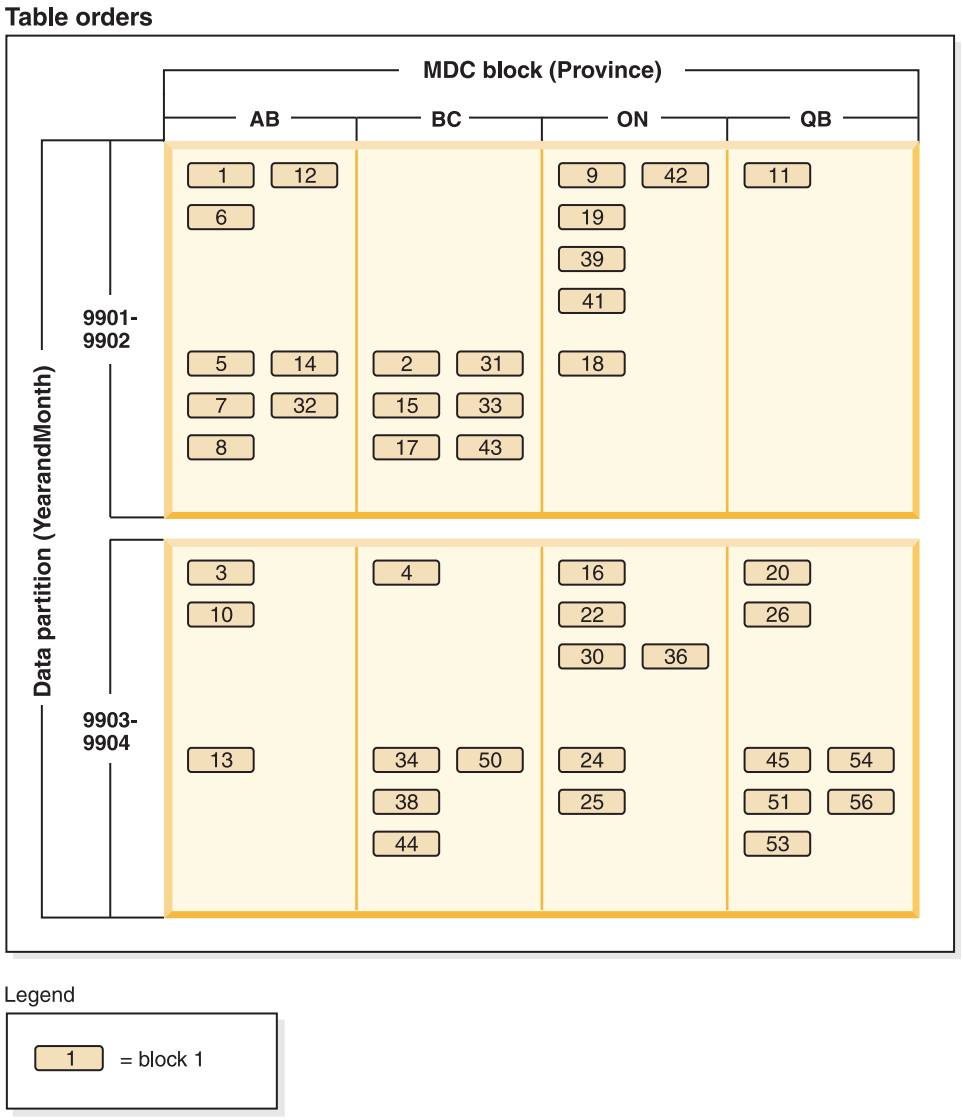


Figure 6. A table partitioned by YearAndMonth and organized by Province

Example 2:

Finer granularity can be achieved by adding YearAndMonth to the ORGANIZE BY clause, as shown in Figure 7 on page 33.

```
CREATE TABLE orders (YearAndMonth INT, Province CHAR(2))
PARTITION BY RANGE (YearAndMonth)
(STARTING 9901 ENDING 9904 EVERY 2)
ORGANIZE BY (YearAndMonth, Province);
```

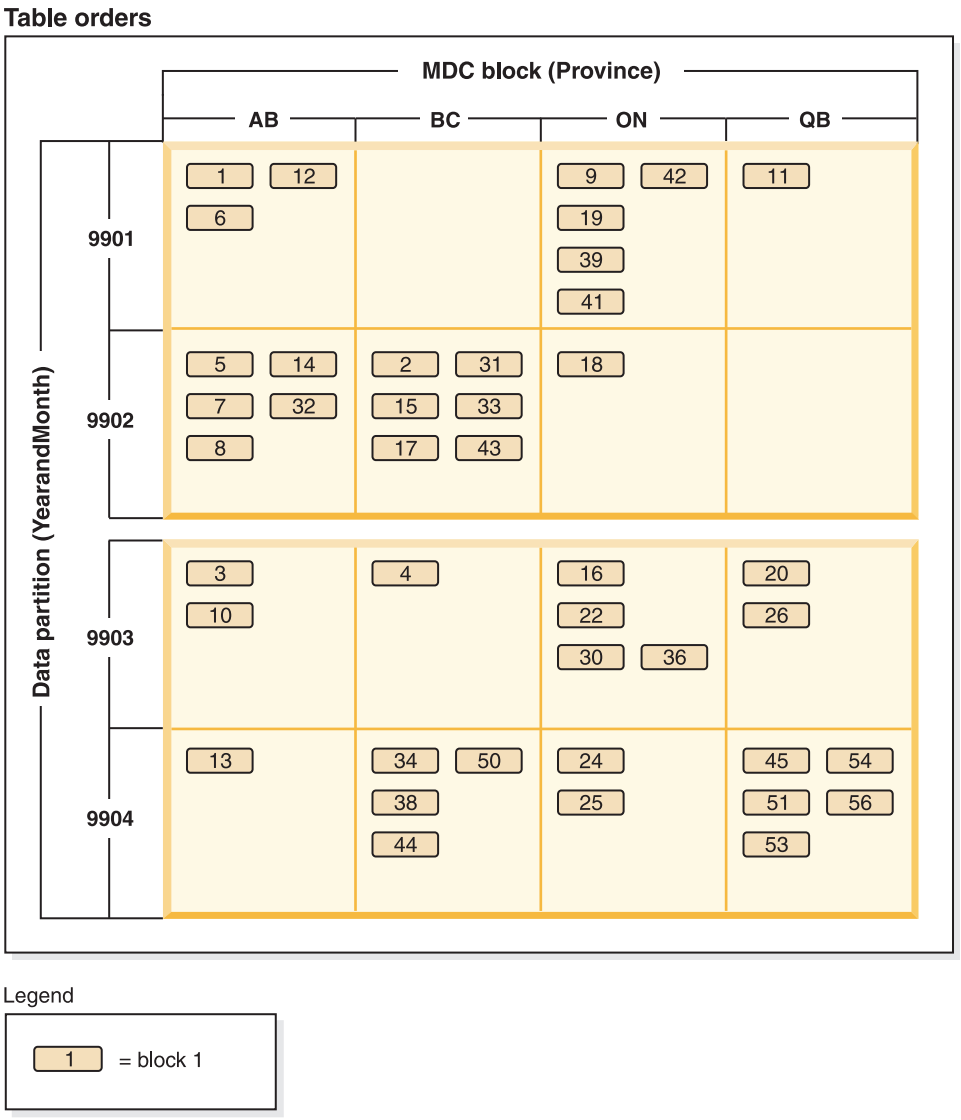


Figure 7. A table partitioned by YearAndMonth and organized by Province and YearAndMonth

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.

Chapter 2. Range-clustered tables

A range-clustered table (RCT) is a table layout scheme where each record in the table has a predetermined record ID (RID) which is an internal identifier used to locate a record in a table.

For each table that holds your data, consider which of the possible table types would best suit your needs. For example, if you have data records that will be loosely clustered (not monotonically increasing), consider using a regular table and indexes. If you have data records that will have duplicate (not unique) values in the key, you should not use a range-clustered table. If you cannot afford to preallocate a fixed amount of storage on disk for the range-clustered tables you might want, you should not use this type of table. These factors will help you to determine whether you have data that can be used as a range-clustered table.

Advantages associated with a range-clustered table structure

There are some advantages to using range-clustered tables.

- Direct access

  Access is through a range-clustered table key-to-RID mapping function.
- Less maintenance

  A secondary structure such as a B+ tree does not need to be updated for every INSERT, UPDATE, or DELETE.
- Less logging

  There is less logging done for range-clustered tables when compared to a similarly sized regular table and associated B+ tree index.
- Less buffer pool memory required

  There is no additional memory required to store a secondary structure.
- Order properties of B+ tree tables

  The ordering of the records is the same as what was achieved by B+ tree tables without requiring extra levels or B+ tree next-key locking schemes. With RCT, the code path length is reduced compared to regular B+ tree indexes. To obtain this advantage, however, the range-clustered table must be created with DISALLOW OVERFLOW and the data must be dense, not sparse.
- One less index

  Mapping each key to a location on disk means that the table can be created with one less index than would have been necessary otherwise. With range-clustered tables, the application requirements for accessing the data in the table might make a second, separate index unnecessary. You may still choose to create regular indexes, especially if the application requires it.

Range-clustered table incompatibilities

In addition to those considerations, there are some incompatibilities that either limit places where range-clustered tables can be used, or other utilities that do not work with these tables.

The limitations on range-clustered tables include:

- Range clustered tables on partitioned tables are not supported.

© Copyright IBM Corp. 1993, 2009

If you attempt to create a partitioned table with range clustering, the error message SQL0270 rc=87 is returned.

- Declared global temporary tables (DGTT) are not supported.

  These temp tables are not allowed to use the range cluster property.
- Automatic summary tables (AST) are not supported.

  These tables are not allowed to use the range cluster property.
- Load utility is not supported.

  Rows must be inserted one at a time through an import operation or a parallel inserting application.
- REORG TABLE utility is not supported.

  Range-clustered tables that are defined to DISALLOW OVERFLOW will not need to be reorganized. Those range-clustered tables defined to ALLOW OVERFLOW are still not permitted to have the data in this overflow region reorganized.
- Range-clustered tables on one logical machine only.

  On the Enterprise Server Edition (ESE) with the Database Partitioning Feature (DPF), a range-clustered table cannot exist in a database containing more than one database partition.
- The design advisor will not recommend range-clustered tables.
- Range-clustered tables are, by definition, already clustered.

  This means that the following clustering schemes are incompatible with range-clustered tables:

  – Multi-dimensional clustered (MDC) table

  – Clustering indexes
- Value and default compression are not supported.
- Reverse scans on the range-clustered table are not supported.
- The REPLACE option on the IMPORT command is not supported.
- The WITH EMPTY TABLE option on the ALTER TABLE ... ACTIVATE NOT LOGGED INITIALLY statement is not supported.

Range-clustered tables and out-of-range record key values

You control the behavior of a range-clustered table (RCT) that allows overflow records by using the CREATE TABLE statement and the ALLOW OVERFLOW option. In this way, you ensure that all of the pages required by the table within the defined range are allocated immediately.

Once created, any records with keys that fall into the defined range work the same way, regardless of whether the table is created with the overflow option allowed or disallowed. The difference occurs when there is a record with a key that falls outside of the defined range. In this case, when the table allows overflow records, the record is placed in the overflow area, which is dynamically allocated. As more records are added from outside the defined range, they are placed into the growing overflow area. Actions against the table that involve this overflow area will require longer processing time because the overflow area must be accessed as part of the action. The larger the overflow area, the longer it will take to access the overflow area. After prolonged use of the overflow area, consider reducing its size by exporting the data from the table to a new range-clustered table that you have defined using new, extended ranges.

There might be times when you do not want records placed into a range-clustered table to have record key values falling outside of an allowed or defined range. For this type of RCT to exist, you must use the DISALLOW OVERFLOW option on the CREATE TABLE statement. Once you have created this type of RCT, you might have to accept error messages if a record key value falls outside of the allowed or defined range.

Range-clustered table locks

Within normal processing, locking of records takes place to ensure that only one application or user has access to a record or group of records at any given time. With range-clustered tables, instead of key and next-key locking, "discrete locking" is used.

This method locks all records that are effected by, or might be effected by, the operation requested by the application or user. The number of locks that are obtained depends on the isolation level.

Qualifying rows in range-clustered tables that are currently empty but have been preallocated are locked. This avoids the need for next-key locking. As a result, fewer locks are required for a dense, range-clustered table.

Chapter 3. Multi-dimensional clustered (MDC) tables

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.

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.

© Copyright IBM Corp. 1993, 2009

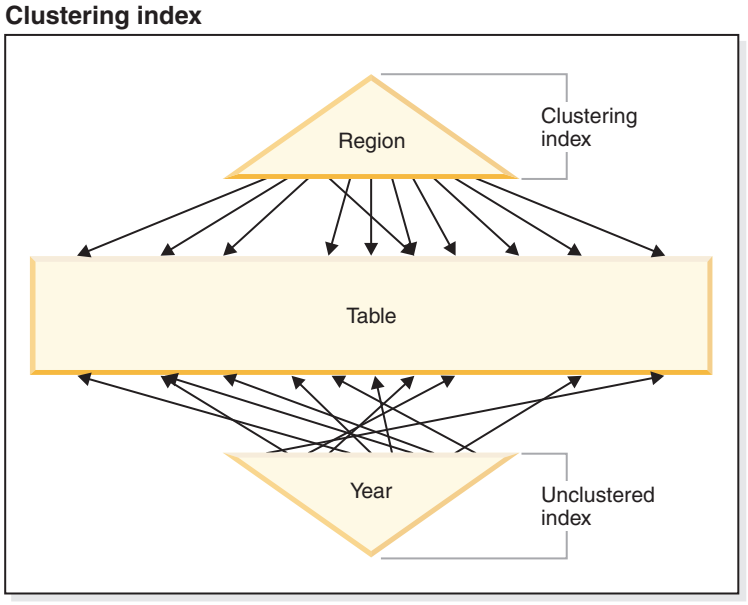


Figure 8. A regular table with a clustering index

The table in Figure 8 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.

Choosing MDC table dimensions

Once 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. Aspects of designing MDC tables and some guidance regarding the selection of appropriate dimensions and block sizes can be seen using the related concepts links.

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, especially those with low cardinalities, that are involved in equality or range predicate queries will show the greatest benefit from, and should be considered as candidates for, clustering dimensions. You will also want to consider creating dimensions for foreign keys in an MDC fact table involved in star joins with dimension tables. You should keep in mind the performance benefits of automatic and continuous clustering on more than one dimension, and of clustering at an 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'
```

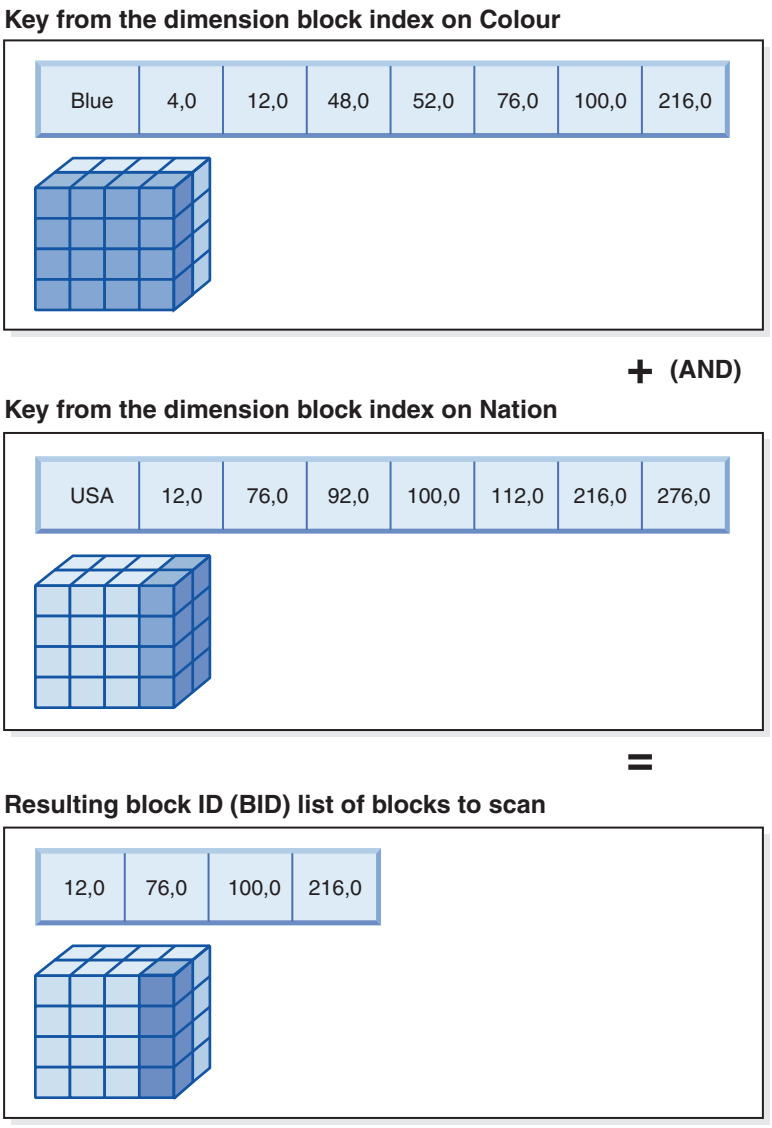


Figure 9. A query request that uses a logical AND operation with two block indexes

To carry out this query request, the following is done (and is shown in Figure 9):

- 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 and a IN predicate on the c2 dimension, as well as 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
```

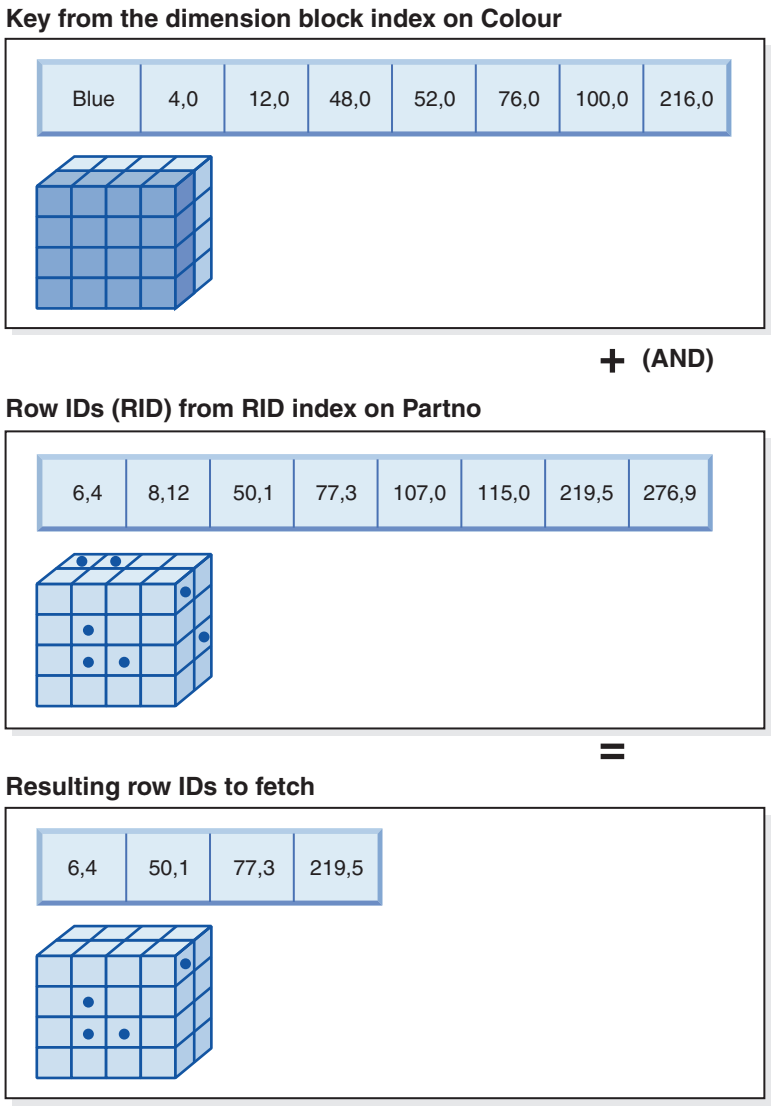


Figure 10. A query request that uses a logical AND operation on a block index and a row ID (RID) index

To carry out this query request, the following is done (and is shown in Figure 10):

- 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
```

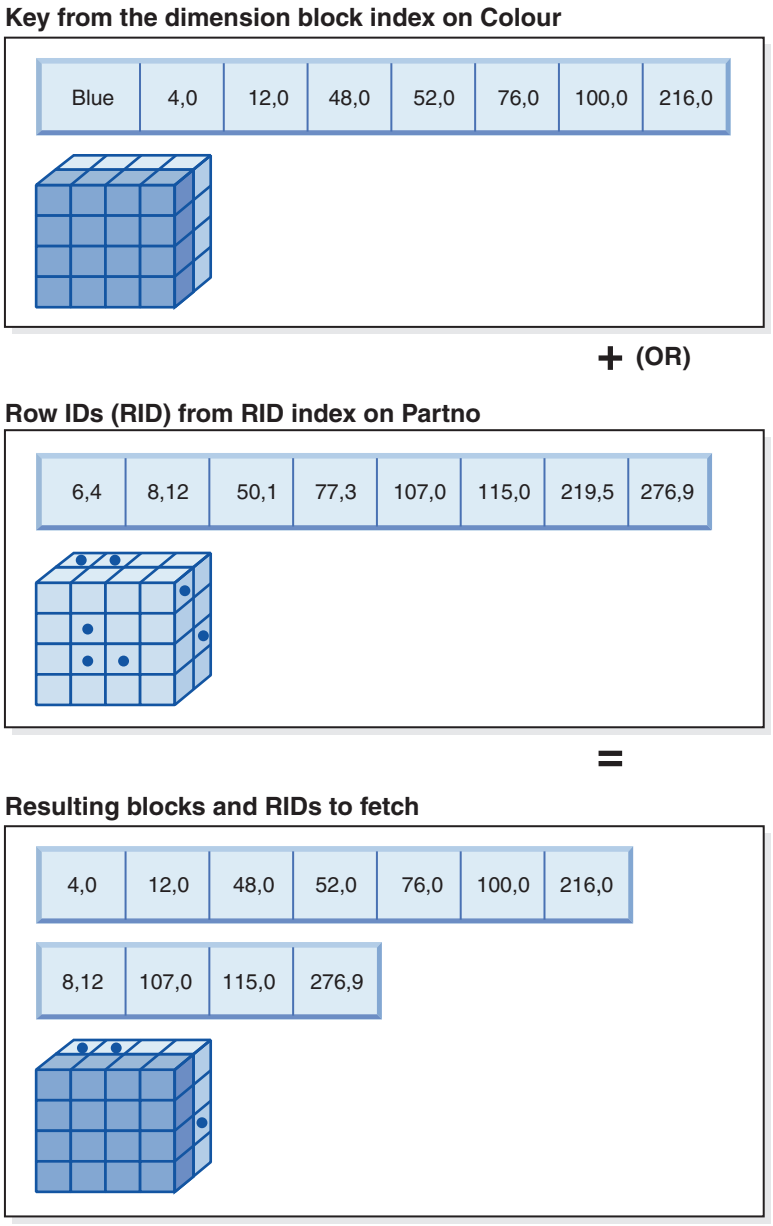


Figure 11. How block index and row ID using a logical OR operation works

To carry out this query request, the following is done (and is shown in Figure 11):

- 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 and an equality predicate on a non-dimension column c4, as well as a logical OR operation. If there is a RID

index on the c4 column, this may 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 may be chosen instead, since all records must be checked. The logical OR operation would use a block index scan on c1 for values less than 4, as well as 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 have to be MDC tables. Region, year, and product are columns of the respective dimension tables which 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 will still require an entire block of pages allocated to it, in order to store the records for that cell. If the block size is large, this table could end up being much larger than it really 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 simply 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, you will have to 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, you need to consider how many records there are per cell on each database partition, as blocks are allocated for data on a database partition basis. When estimating the space occupancy and density in a partitioned database environment, you need to consider the number of records per cell on average on each database partition, not across the entire table. See the section called "Multidimensional clustering (MDC) table creation, placement, and use" for more information.

   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 in order 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.

You should also 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 Database for Linux, UNIX, and Windows can use the block index on YearAndMonth to quickly narrow down which months contain the days desired 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 may 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.

Considerations when creating MDC tables

There are many factors that should be considered when creating MDC tables. The following sections discuss how your decisions on how to create, place, and use your MDC tables could be influenced by your current database environment (for example, whether you have a partitioned database or not), and by your choices of dimensions for your MDC table. Also discussed is the DB2 Design Advisor, and how it can be used to provide advice on some of these issues.

Moving data from existing tables to MDC tables

To improve query performance and reduce the overhead 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: export your data, drop the original table (optional), create a multidimensional clustering (MDC) table (using the CREATE TABLE statement with the ORGANIZE BY DIMENSIONS clause), and 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 does the following when altering a table:

- Drop all dependent objects of the table
- Rename the table
- Create the table using the new definition
- Recreate all dependent objects of the table
- 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 may be used to transform from a old data type to a new data type.

MDC tables in SMS table spaces

If you plan to store MDC tables in an SMS table space, you need to use multipage file allocation. (Multipage file allocation is the default for newly created databases in Version 8.2 and later.) The reason for this is that MDC tables are always extended by whole extents, and it is important that all the pages in these extents are physically consecutive. Therefore, there is no space advantage to disabling multipage file allocation; and furthermore, enabling it will significantly increase the chances that the pages in each extent are physically consecutive.

MDC Advisor feature on the DB2 Design Advisor

The DB2 Design Advisor (db2advis), formerly known as the Index Advisor, 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 mathematics expression to reduce the cardinality (the number of distinct values) of a clustering dimension. A common example of a coarsification is the date where coarsification could be 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 will 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 will be created in the same table space as the existing table, and will therefore have the same extent size. The recommendations for MDC dimensions would change depending on the extent size of the table space since the extent size impacts the number of records that can fit into a block or cell. This directly affects the density of the cells.

Only single-column dimensions, and not composite-column dimensions, are considered, although single or multiple dimensions may be recommended for the table. The MDC feature will recommend 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 VARGRAPH 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 impact 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 impact of any table expansion resulting from the organization of data along particular MDC dimensions.

The MDC feature is enabled 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 will not explore tables that are less than 12 extents in size.

The advisor will analyze 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 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 conjunction with 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 nodes in order to:

- Take advantage of multiple machines to increase processing requests in parallel.
- Increase the physical size of the table beyond a single database partition's limits.
- 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 will contain 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 will be used to divide the data. As a result, the Red and Blue slices may 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 will have a subset of data from each slice. When choosing dimensions and estimating cell occupancy (see the section called "Density of cells"), 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 will be heavily used in queries, you can cluster the table on the columns involved, 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 native 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 will be processed by the relational operators involved. Note that the size of a block (the number of pages) will be 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 will be 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 will have the same number of blocks as the table in Example 1, but one less dimension block index. In Example 1, there will be three dimension block indexes, one for each of the columns c1, c3, and c4. In Example 2, there will be two dimension block indexes, one on the column c1 and the other on the columns c3 and c4. The main differences between these two approaches is that, in Example 1, queries involving just 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 just c4 involve more processing. However, in Example 2 there will be 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. In order to implement the rolling up of dimensions in this way, you can use generated columns. This type of column definition will allow 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, while column c6 rolls up column c1 to a coarser granularity in time. This statement will cluster 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 will be 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 we create an MDC table with dimensions on the generated column month, where `month = INTEGER (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 will only allow 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 usual way, with a starting key of 4 and a stop key of 6.

To make this function monotonic, you would have to include the year as the high order part of the month. Version 9.2 provides 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 DATENUM( ), DAYS( ), 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 will only support the use of equality predicates on its base column.

Load considerations for MDC tables

If you roll data in to your data warehouse on a regular basis, you can use MDC tables to your advantage. In MDC tables, load will first reuse previously emptied blocks in the table before extending the table and adding new blocks for the remaining data.

After you have deleted 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 have been emptied after the (committed) deletion. In Version 9.5, the MDC rollout feature is added 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 will be freed and can be reused.

When loading data into MDC tables, the input data can be either sorted or unsorted. If unsorted, consider doing the following:

- Increase the *util_heap_sz* configuration parameter.

  To improve the performance of the load utility when loading MDC tables, the *util_heap_sz* database configuration parameter value should be increased. The mdc-load algorithm performs significantly 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* should be increased accordingly.
- Increase the value given with the DATA BUFFER clause of the LOAD command.

  Increasing this value will affect a single load request. The utility heap size must be large enough to accommodate the possibility of multiple concurrent load requests. When the DATA BUFFER option of the LOAD command is specified, its value should also be increased.
- 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 should be set to a value that takes this into account.

The following restrictions apply when loading data into multidimensional clustering (MDC) tables:

- The SAVECOUNT option of 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 tables. If a load is executed into an MDC table without the `anyorder` modifier, it will be explicitly enabled by the utility.

When using the LOAD command with an MDC table, violations of unique constraints are be handled as follows:

- If the table included a unique key prior to 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 or for the initial populating of a table.

MDC load operations always have a build phase since all MDC tables have block indexes.

Logging considerations for MDC tables

In cases where columns previously or otherwise indexed by RID indexes are now dimensions and so are indexed with block indexes, index maintenance and logging are significantly reduced.

Only when the last record in an entire block is deleted does the database manager need to remove the BID from the block indexes and log this index operation. Similarly, only when a record is inserted to a new block (if it is the first record of a logical cell or an insert to a logical cell of currently full blocks) does the database manager need to insert a BID in the block indexes and log that operation. Because blocks can be between 2 and 256 pages of records, this block index maintenance and logging will be relatively small. Inserts and deletes to the table and to RID indexes will still be 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 index considerations for MDC tables

When you define dimensions for an MDC table, dimension block indexes are created. In addition, a composite block index may also be created when multiple dimensions are defined. If you have defined only one dimension for your MDC table, however, DB2 will create only one block index, which will serve both as the dimension block index and as the composite block index.

Similarly, if you create an MDC table that has dimensions on column A, and on (column A, column B), DB2 will create 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 will also serve as the composite block index.

The composite block index is also used in query processing to access data in the table having specific dimension values. Note that the order of key parts in the composite block index may affect its use or applicability for query processing. The order of its key parts is determined by the order of columns found in the entire ORGANIZE BY DIMENSIONS clause used when creating the MDC table. For example, if a table is created using the statement

```
CREATE TABLE t1 (c1 int, c2 int, c3 int, c4 int)
   ORGANIZE BY DIMENSIONS (c1, c4, (c3,c1), c2)
```

then the composite block index will be created on columns (c4,c3,c1,c2). Note that although c1 is specified twice in the dimensions clause, it is used only once as a key part for the composite block index, and in the order in which it is first found. The order of key parts in the composite block index makes no difference for insert processing, but may do so for query processing. Therefore, if it is more desirable to have the composite block index with column order (c1,c2,c3,c4), then the table should be created using the statement

```
CREATE TABLE t1 (c1 int, c2 int, c3 int, c4 int)
   ORGANIZE BY DIMENSIONS (c1, c2, (c3,c1), c4)
```

Block indexes for MDC tables

This topic shows how records are organized in MDC tables using block indexes.

The MDC table shown in Figure 12 on page 56 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.

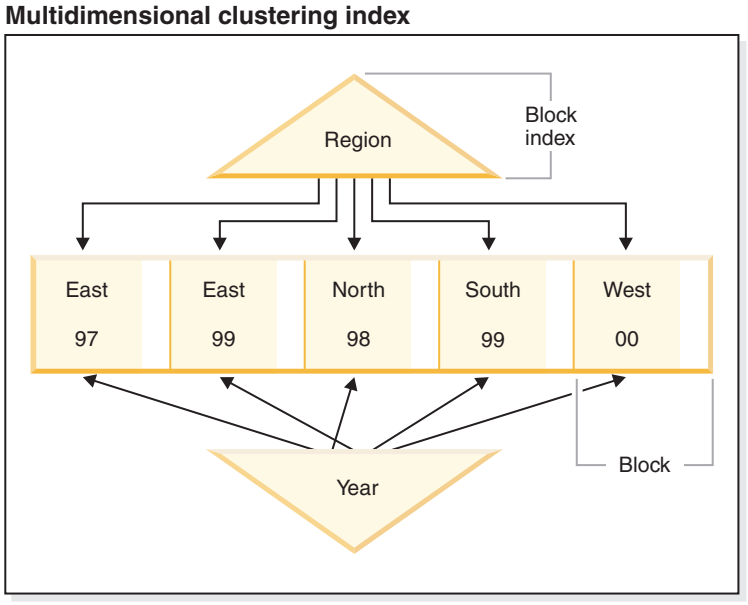


Figure 12. A multidimensional clustering table

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 41 and *The Design Advisor*.

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 clause of the CREATE TABLE statement. Refer to "Block index considerations for MDC tables" on page 54 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.

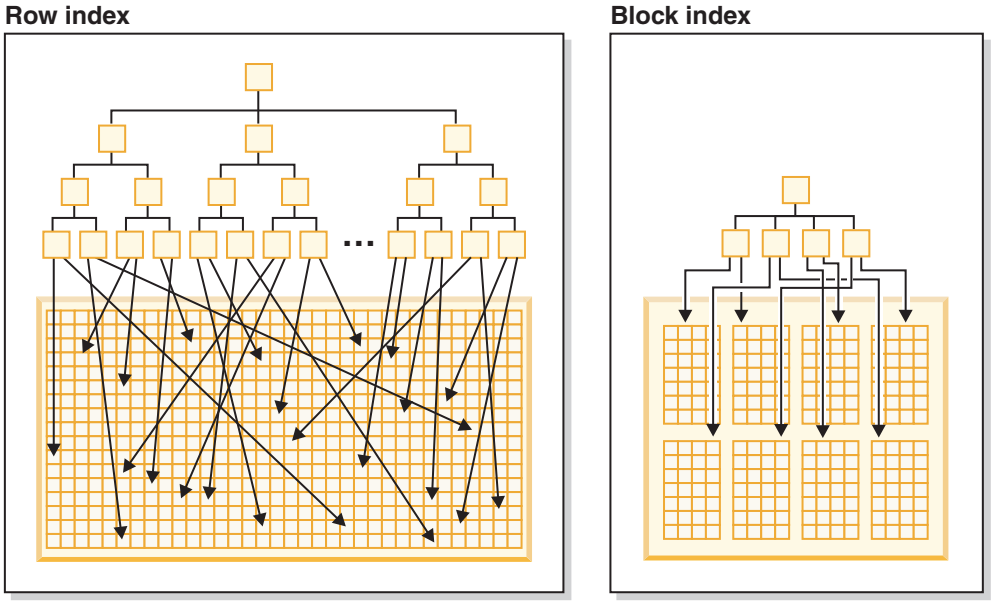


Figure 13. How row indexes differ from block indexes

As seen in Figure 13, 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*.

Scenario: Multidimensional clustered (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 14 on page 58, 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.

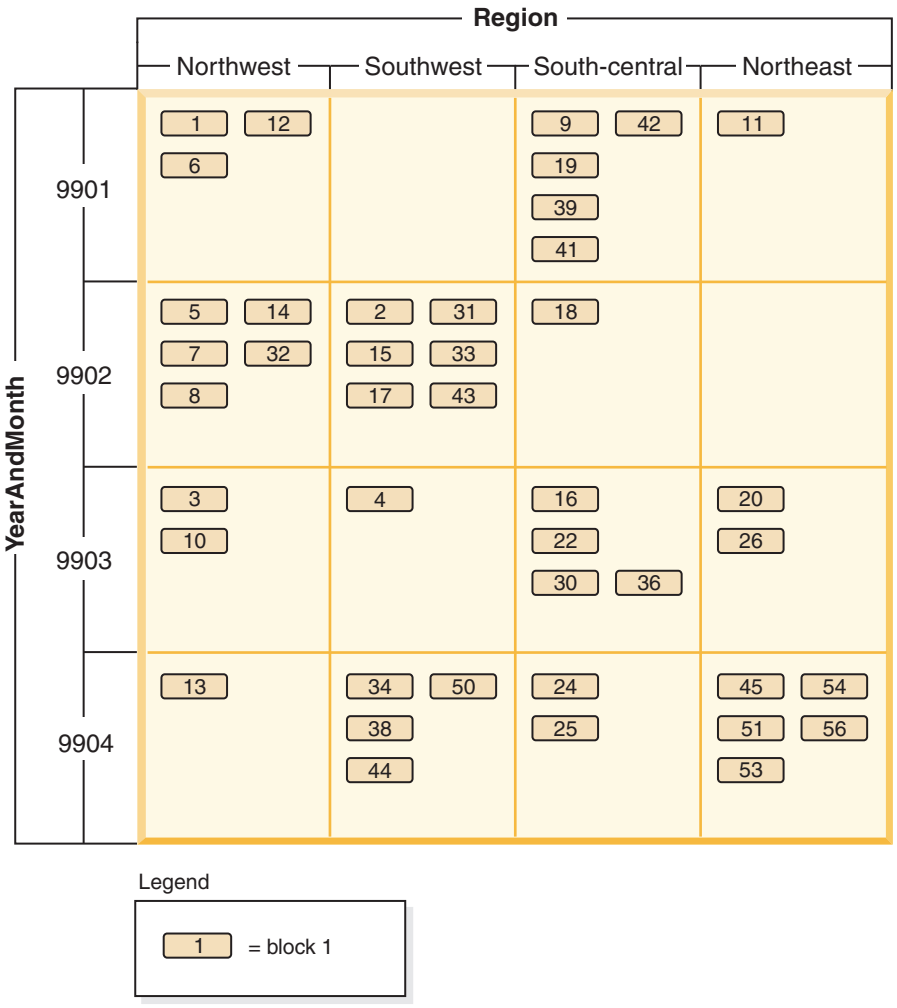


Figure 14. Multidimensional table with dimensions of 'Region' and 'YearAndMonth' that is called Sales

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 15 on page 59, 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.

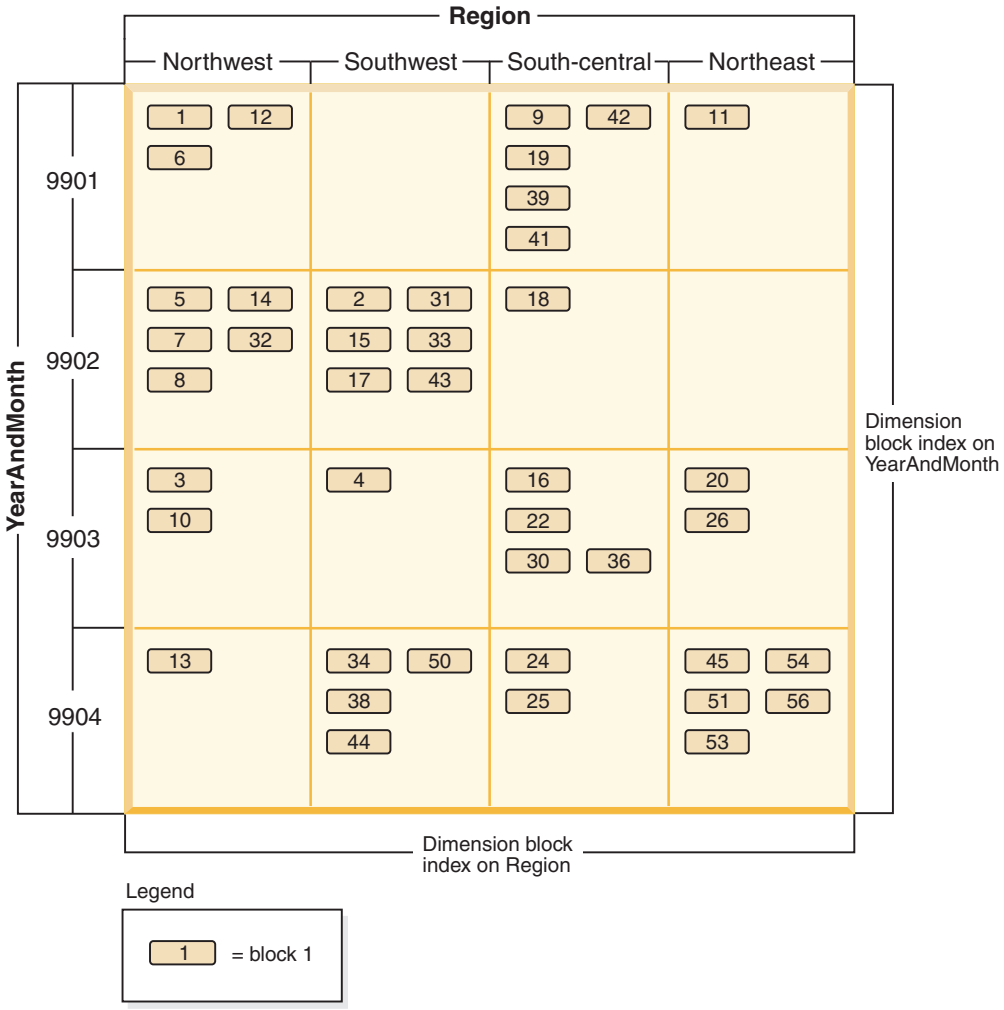


Figure 15. Sales table with dimensions of 'Region' and 'YearAndMonth' showing dimension block indexes

Figure 16 on page 60 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 16 on page 60, 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 14 on page 58).

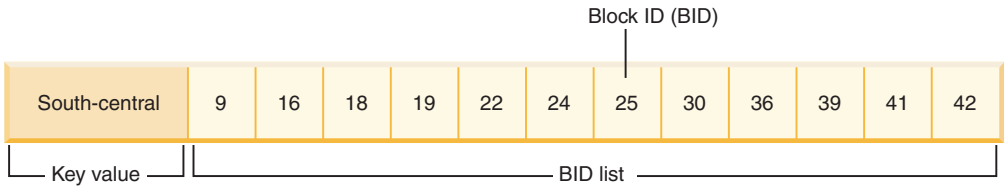


Figure 16. Key from the dimension block index on 'Region'

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 17.

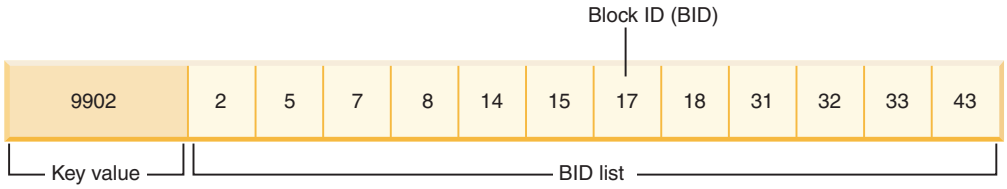


Figure 17. Key from the dimension block index on 'YearAndMonth'

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. First, the block index is so much smaller than a regular index, the block index scan is very efficient. Second, prefetching of the data pages does not rely on sequential detection when block indexes are used. DB2 looks ahead in the index, prefetching the data pages of the blocks into memory using big-block I/O, and ensuring that the scan does not incur the I/O when the data pages are accessed in the table. Third, 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. Fourth, if a block-based buffer pool is used with its block size being the extent size, then MDC blocks will be prefetched from sequential pages on disk into sequential pages in memory, further increasing the effect of clustering on performance. Finally, the records from each block are retrieved using a mini-relational scan of its data pages, which is often a faster method of scanning data than 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 18.

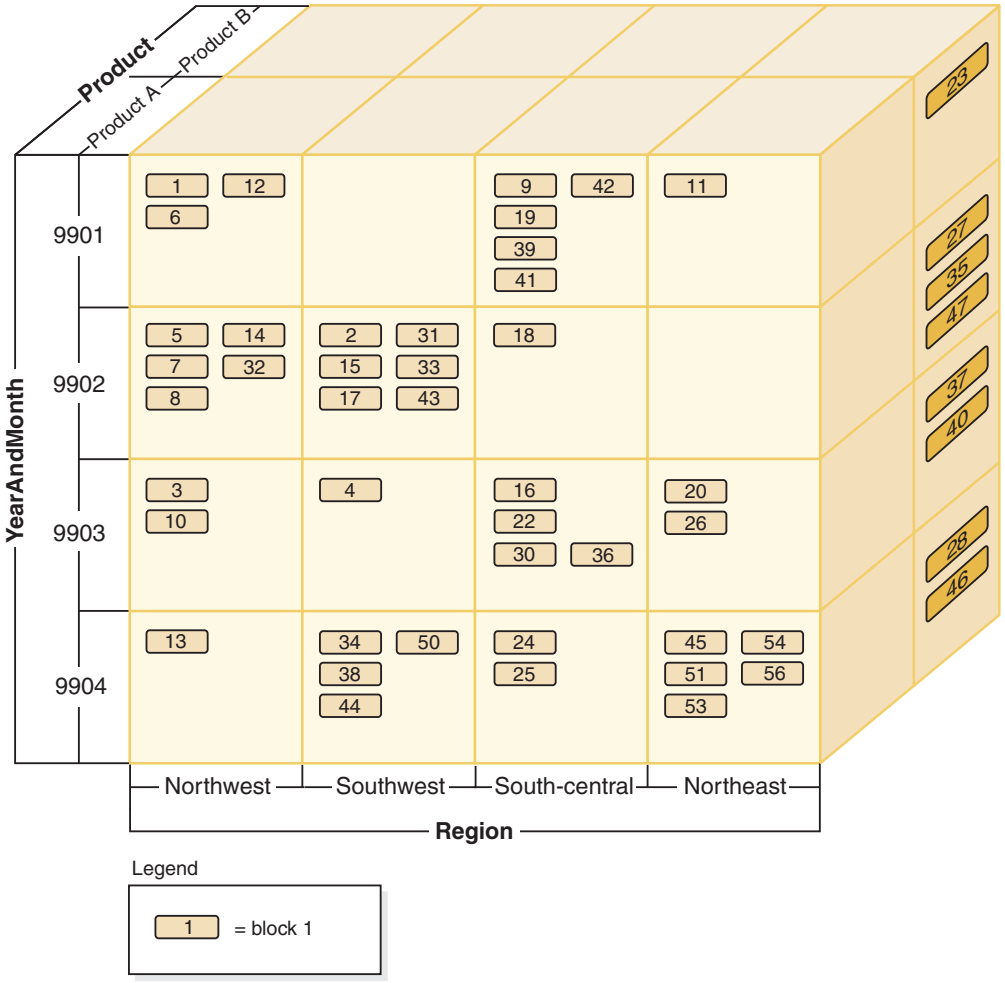


Figure 18. Multidimensional table with dimensions of 'Region', 'YearAndMonth', and 'Product'

Four block indexes will be created for the MDC table shown in Figure 18: 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 19.) 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 20 on page 62.)

Product A	1	2	3	...	11	...	20	22	24	25	26	30	...	56

Figure 19. 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 20. 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 example above, 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 21 on page 63 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".

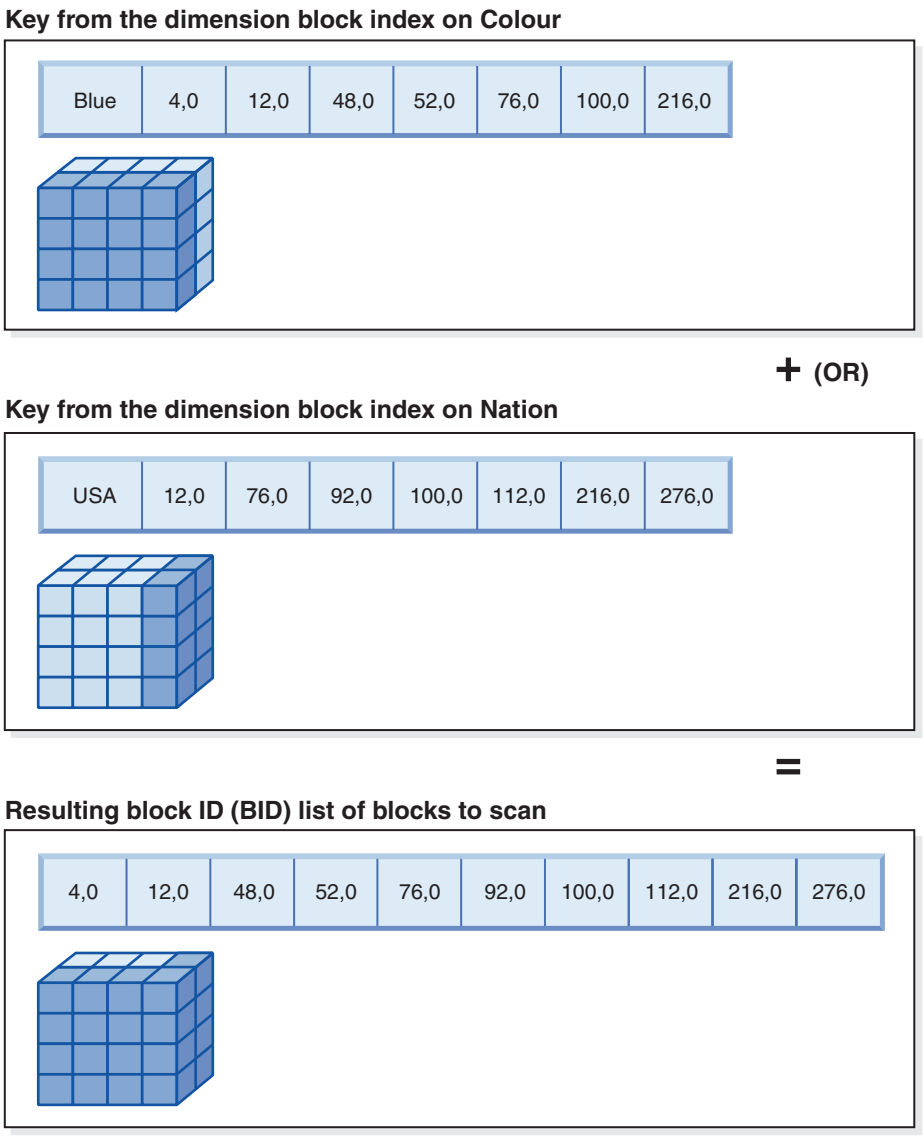


Figure 21. How the logical OR operation can be used with block indexes

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 22.) 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).

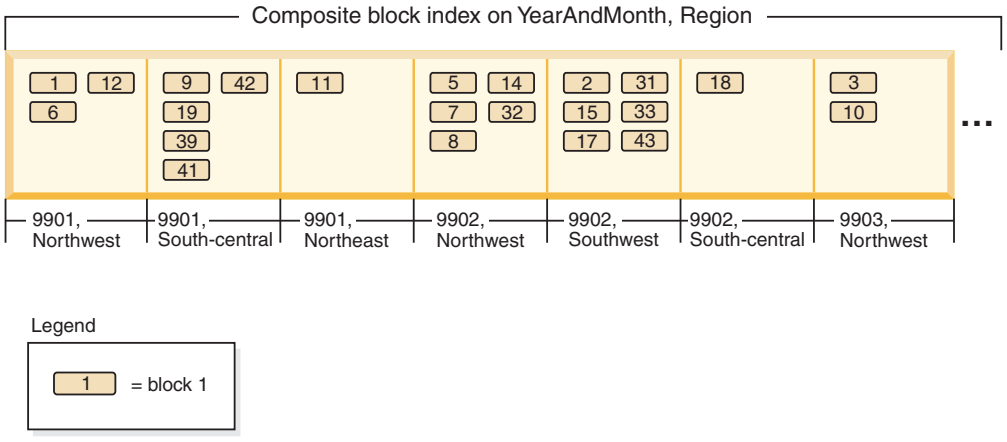


Figure 22. Composite block index on 'YearAndMonth', 'Region'

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 22 on page 64 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 23. 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.

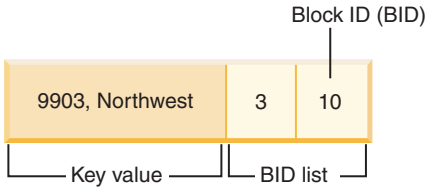


Figure 23. Key from composite block index on 'YearAndMonth', 'Region'

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 24 on page 66 for an illustration of the keys of the dimension block indexes after the addition of Block 48.

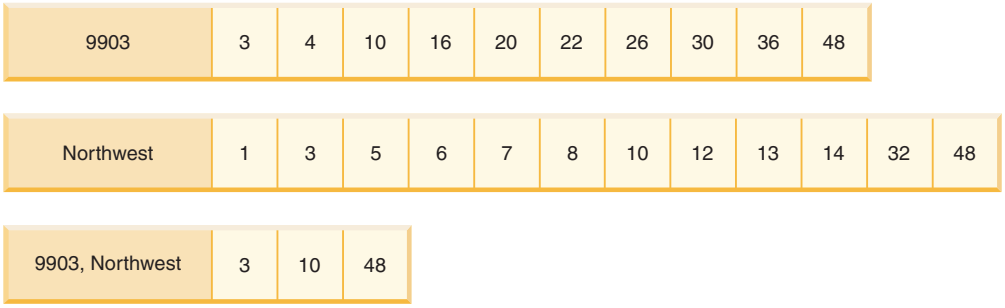


Figure 24. Keys from the dimension block indexes after addition of Block 48

Block maps 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. This 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 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.

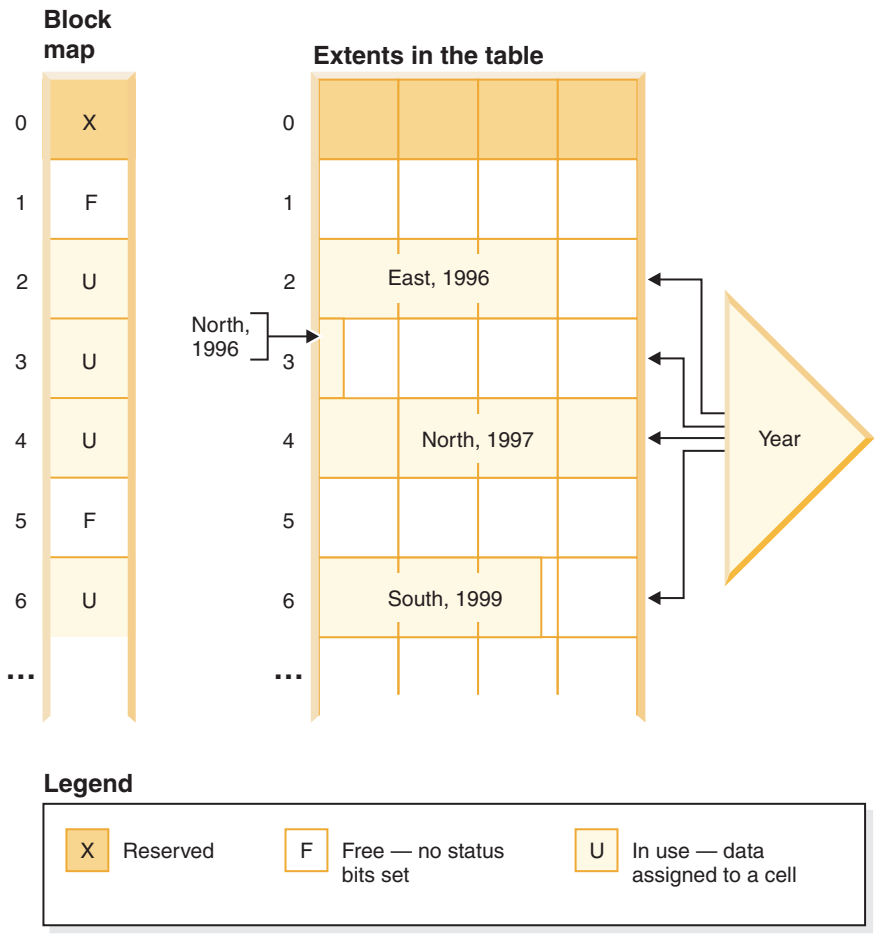


Figure 25. How a block map works

In Figure 25, 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, those 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 25) would start from the third extent (extent 2) in the table, skipping the first reserved extent and the following empty extent, scan blocks 2, 3 and 4 in the table, skip the next extent (not touching any of that extent's data pages), and then continue scanning from there.

Deleting from MDC tables

When a record is deleted in an MDC 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, however, the database manager frees the block by changing its IN_USE status bit and removing the block's BID from all block indexes. Again, if there are record-based indexes as well, the RID is removed from them.

Note: Therefore, block index entries need only be removed once per entire block and only if the block is completely emptied, instead of once per deleted row in a record-based index.

Updates to MDC 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 affects a variable length column and the record 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.

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.

Block indexes only need to be updated when inserting the first record into a block or when deleting the last record from a block. Index overhead associated with block indexes for maintenance and logging is therefore much less than the index overhead associated with regular indexes. For every block index that would have otherwise been a regular index, the maintenance and logging overhead is greatly reduced.

MDC tables are treated like any existing table; that is, triggers, referential integrity, views, and materialized query tables can all be defined upon them.

Table partitioning and multidimensional clustering tables

A table can be both multi-dimensional clustered and partitioned. In a table that is both multi-dimensional clustered and partitioned, columns can be used both in the table partitioning range-partition-spec and in the MDC key. This is useful for achieving 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 multi-column, while MDC is multi-dimension.

Characteristics of a mainstream DB2 V9.1 data warehouse

The following recommendations were focused on typical, mainstream warehouses that are new for DB2 V9.1. The following characteristics are assumed:

- The database runs on multiple machines or multiple AIX logical partitions.
- Database partitioning feature (DPF) is 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 million to 100 billion 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 7. Using table partitioning with MDC tables

Issue	Recommended scheme	Recommendation
Data availability during roll-out	Table partitioning	Use the DETACH PARTITION clause to roll out large amounts of data with minimal disruption.
Query performance	Table partitioning and MDC	MDC is best for querying multiple dimensions. Table partitioning helps through data partition elimination.

Table 7. Using table partitioning with MDC tables (continued)

Issue	Recommended scheme	Recommendation
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 (either ALLOW READ ACCESS or ALLOW NO ACCESS)	Table partitioning and MDC	MDC provides most of the benefit here. Table partitioning provides incremental benefits.
Load data ″continually″ (ALLOW READ ACCESS)	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, range partitioning helps reduce the need for reorg by maintaining some course 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 6 on page 32.

```
CREATE TABLE orders (YearAndMonth INT, Province CHAR(2))
PARTITION BY RANGE (YearAndMonth)
(STARTING 9901 ENDING 9904 EVERY 2)
ORGANIZE BY (Province);
```

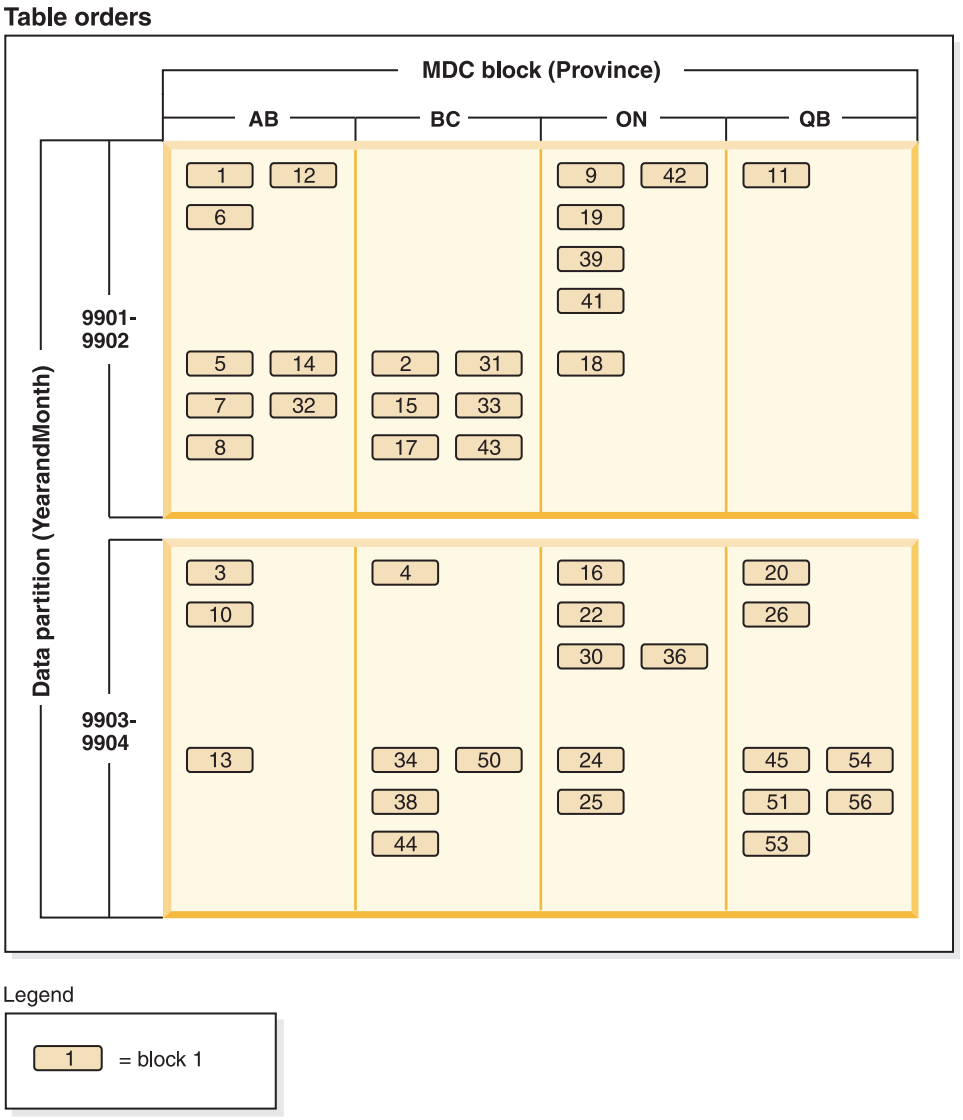


Figure 26. A table partitioned by YearAndMonth and organized by Province

Example 2:

Finer granularity can be achieved by adding YearAndMonth to the ORGANIZE BY clause, as shown in Figure 7 on page 33.

```
CREATE TABLE orders (YearAndMonth INT, Province CHAR(2))
PARTITION BY RANGE (YearAndMonth)
(STARTING 9901 ENDING 9904 EVERY 2)
ORGANIZE BY (YearAndMonth, Province);
```

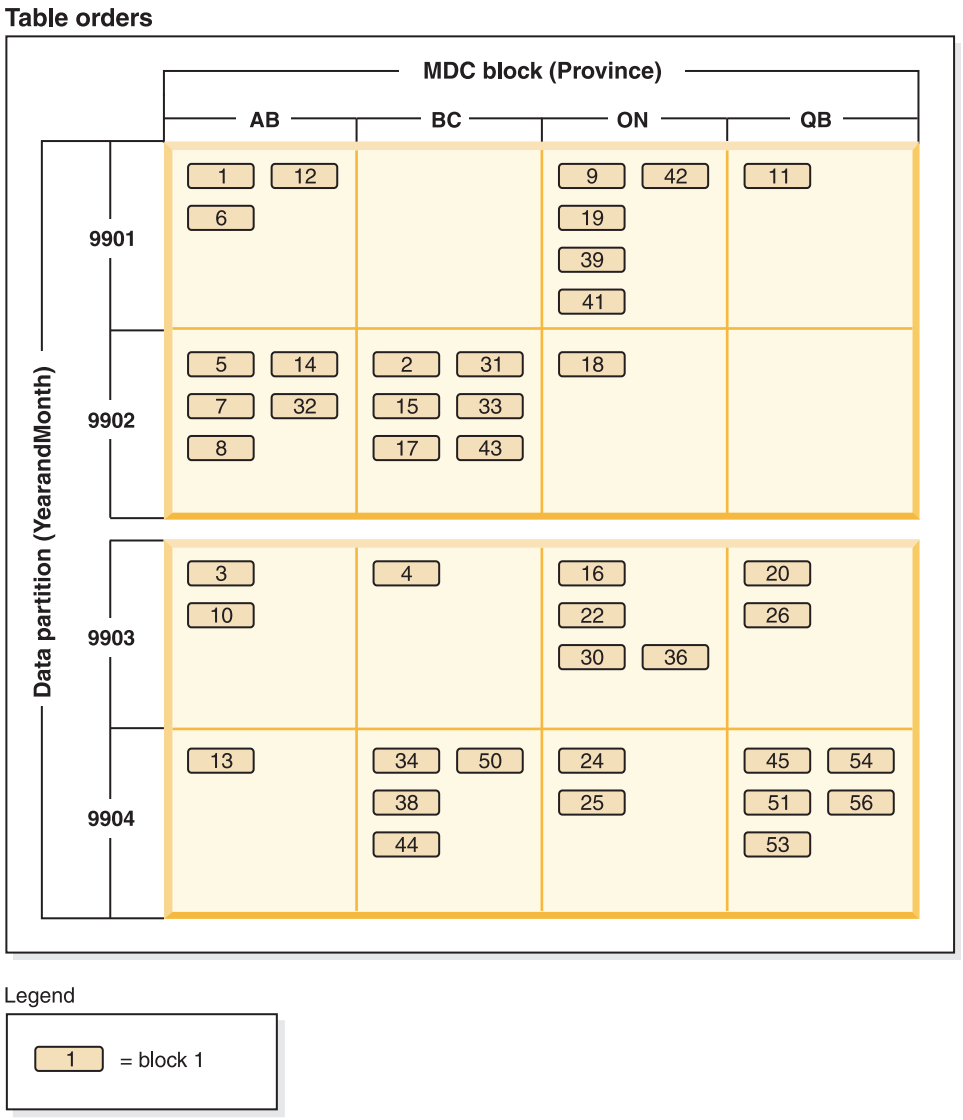


Figure 27. A table partitioned by YearAndMonth and organized by Province and YearAndMonth

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.

Chapter 4. Parallel database systems

Parallelism

Components of a task, such as a database query, can be run in parallel to dramatically enhance performance. The nature of the task, the database configuration, and the hardware environment, all determine how the DB2 database product will perform a task in parallel.

These considerations are interrelated, and should be considered together when you work on the physical and logical design of a database. The following types of parallelism are supported by the DB2 database system:

- I/O
- Query
- Utility

Input/output parallelism

When there are multiple containers for a table space, the database manager can exploit *parallel I/O*. Parallel I/O refers to the process of writing to, or reading from, two or more I/O devices simultaneously; it can result in significant improvements in throughput.

Query parallelism

There are two types of query parallelism: interquery parallelism and intraquery parallelism.

Interquery parallelism refers to the ability of the database to accept queries from multiple applications at the same time. Each query runs independently of the others, but the database manager runs all of them at the same time. DB2 database products have always supported this type of parallelism.

Intraquery parallelism refers to the simultaneous processing of parts of a single query, using either *intrapartition parallelism*, *interpartition parallelism*, or both.

Intrapartition parallelism

Intrapartition parallelism refers to the ability to break up a query into multiple parts. Some DB2 utilities also perform this type of parallelism.

Intrapartition parallelism subdivides what is usually considered a single database operation such as index creation, database loading, or SQL queries into multiple parts, many or all of which can be run in parallel *within a single database partition*.

Figure 28 on page 74 shows a query that is broken into four pieces that can be run in parallel, with the results returned more quickly than if the query were run in serial fashion. The pieces are copies of each other. To utilize intrapartition parallelism, you must configure the database appropriately. You can choose the degree of parallelism or let the system do it for you. The degree of parallelism represents the number of pieces of a query running in parallel.

© Copyright IBM Corp. 1993, 2009

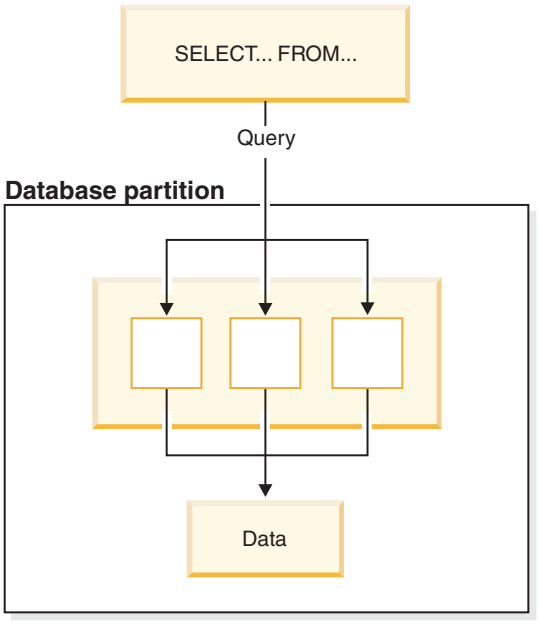


Figure 28. Intrapartition parallelism

Interpartition parallelism

Interpartition parallelism refers to the ability to break up a query into multiple parts across multiple partitions of a partitioned database, on one machine or multiple machines. The query is run in parallel. Some DB2 utilities also perform this type of parallelism.

Interpartition parallelism subdivides what is usually considered a single database operation such as index creation, database loading, or SQL queries into multiple parts, many or all of which can be run in parallel *across multiple partitions of a partitioned database on one machine or on multiple machines*.

Figure 29 on page 75 shows a query that is broken into four pieces that can be run in parallel, with the results returned more quickly than if the query were run in serial fashion on a single database partition.

The degree of parallelism is largely determined by the number of database partitions you create and how you define your database partition groups.

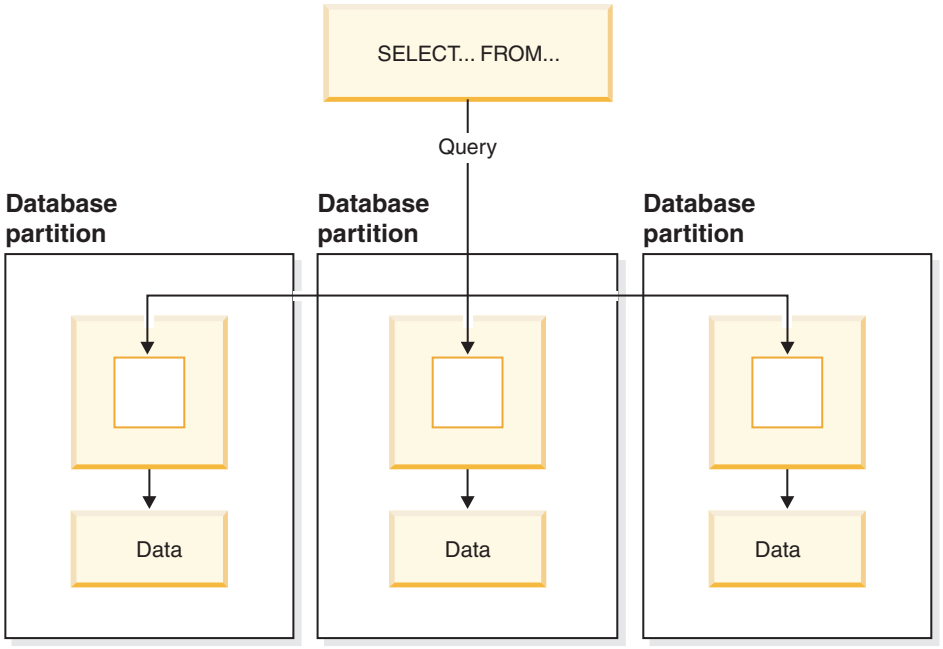


Figure 29. Interpartition parallelism

Simultaneous intrapartition and interpartition parallelism

You can use intrapartition parallelism and interpartition parallelism at the same time. This combination provides two dimensions of parallelism, resulting in an even more dramatic increase in the speed at which queries are processed.

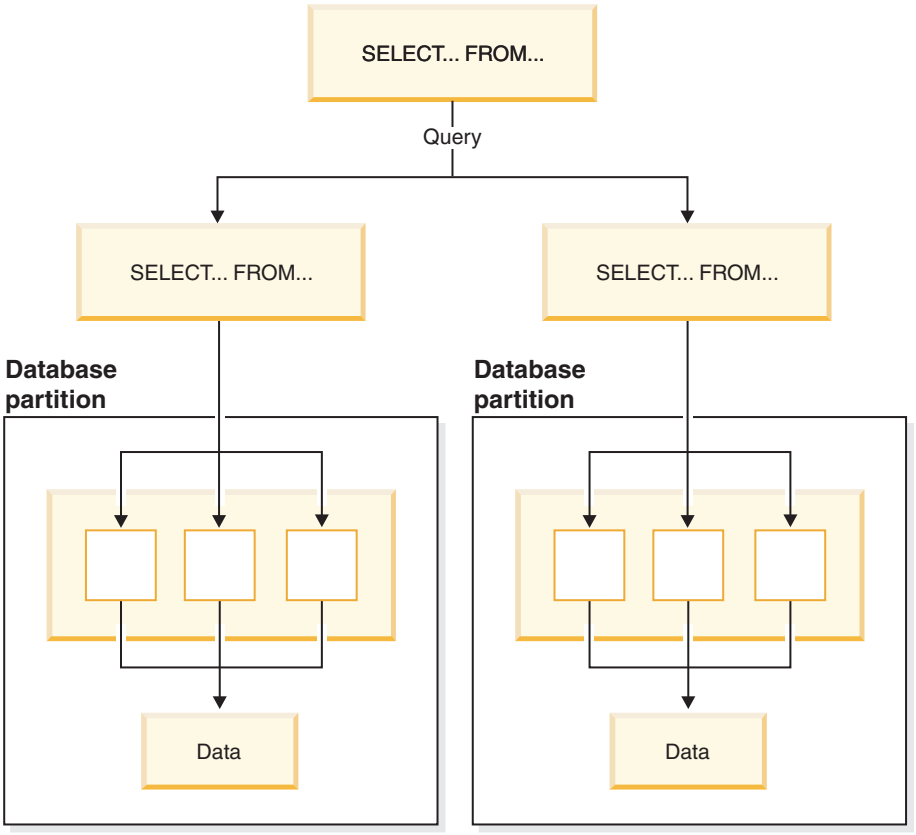


Figure 30. Simultaneous interpartition and intrapartition parallelism

Utility parallelism

DB2 utilities can take advantage of intrapartition parallelism. They can also take advantage of interpartition parallelism; where multiple database partitions exist, the utilities execute in each of the database partitions in parallel.

The load utility can take advantage of intrapartition parallelism and I/O parallelism. Loading data is a CPU-intensive task. The load utility takes advantage of multiple processors for tasks such as parsing and formatting data. It can also use parallel I/O servers to write the data to containers in parallel.

In a partitioned database environment, the LOAD command takes advantage of intrapartition, interpartition, and I/O parallelism by parallel invocations at each database partition where the table resides.

During index creation, the scanning and subsequent sorting of the data occurs in parallel. The DB2 system exploits both I/O parallelism and intrapartition parallelism when creating an index. This helps to speed up index creation when a CREATE INDEX statement is issued, during restart (if an index is marked invalid), and during the reorganization of data.

Backing up and restoring data are heavily I/O-bound tasks. The DB2 system exploits both I/O parallelism and intrapartition parallelism when performing backup and restore operations. Backup exploits I/O parallelism by reading from multiple table space containers in parallel, and asynchronously writing to multiple backup media in parallel.

Partitioned database environments

The Database Partitioning Feature (DPF) extends the database manager to the parallel, multi-partition environment.

- A *database partition* is a part of a database that consists of its own data, indexes, configuration files, and transaction logs. A database partition is sometimes called a node or a database node. 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, while 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 do not need to be aware of the physical location of the data.

The data, while physically split, 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 should 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, while allowing smaller tables to be stored 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. As a result, an add node operation cannot be run. You should not manually update the `db2nodes.cfg` file. A manual update will result in the display of a SQL6031N error.

Database partition and processor environments

Capacity refers to the number of users and applications able to access the database. This is in large part determined by memory, agents, locks, I/O, and storage management. *Scalability* refers to the ability of a database to grow and continue to exhibit the same operating characteristics and response times.

This section provides an overview of the following hardware environments:

- Single database partition on a single processor (uniprocessor)
- Single database partition with multiple processors (SMP)
- Multiple database partition configurations
  - Database partitions with one processor (MPP)
  - Database partitions with multiple processors (cluster of SMPs)
  - Logical database partitions

Capacity and scalability are discussed for each environment.

Single database partition on a single processor

This environment is made up of memory and disk, but contains only a single CPU (see Figure 31 on page 79). It is referred to by many different names, including stand-alone database, client/server database, serial database, uniprocessor system, and single node or non-parallel environment.

The database in this environment serves the needs of a department or small office, where the data and system resources (including a single processor or CPU) are managed by a single database manager.

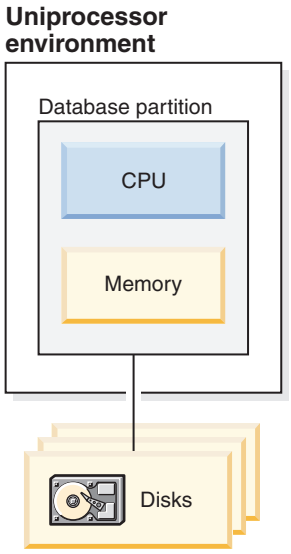


Figure 31. Single database partition on a single processor

Capacity and scalability

In this environment you can add more disks. Having one or more I/O servers for each disk allows for more than one I/O operation to take place at the same time.

A single-processor system is restricted by the amount of disk space the processor can handle. As workload increases, a single CPU may not be able to process user requests any faster, regardless of other components, such as memory or disk, that you may add. If you have reached maximum capacity or scalability, you can consider moving to a single database partition system with multiple processors.

Single database partition with multiple processors

This environment is typically made up of several equally powerful processors within the same machine (see Figure 32 on page 80), and is called a *symmetric multiprocessor (SMP)* system. Resources, such as disk space and memory, are *shared*.

With multiple processors available, different database operations can be completed more quickly. DB2 database systems can also divide the work of a single query among available processors to improve processing speed. Other database operations, such as loading data, backing up and restoring table spaces, and creating indexes on existing data, can take advantage of multiple processors.

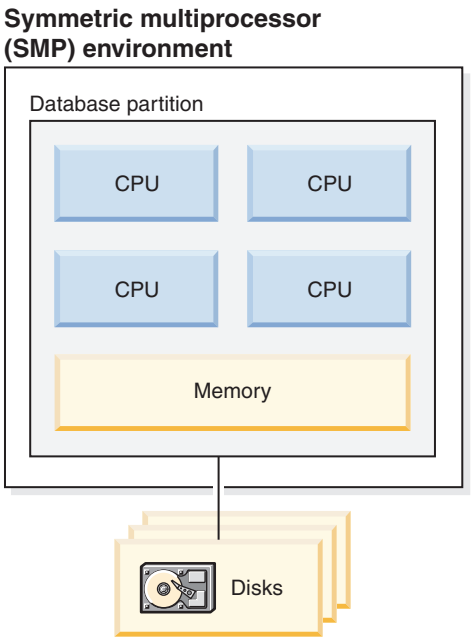


Figure 32. Single partition database symmetric multiprocessor environment

Capacity and scalability

In this environment you can add more processors. However, since the different processors may attempt to access the same data, limitations with this environment can appear as your business operations grow. With shared memory and shared disks, you are effectively sharing all of the database data.

You can increase the I/O capacity of the database partition associated with your processor by increasing the number of disks. You can establish I/O servers to specifically deal with I/O requests. Having one or more I/O servers for each disk allows for more than one I/O operation to take place at the same time.

If you have reached maximum capacity or scalability, you can consider moving to a system with multiple database partitions.

Multiple database partition configurations

You can divide a database into multiple database partitions, each on its own machine. Multiple machines with multiple database partitions can be grouped together. This section describes the following database partition configurations:

- Database partitions on systems with one processor
- Database partitions on systems with multiple processors
- Logical database partitions

Database partitions with one processor

In this environment, there are many database partitions. Each database partition resides on its own machine, and has its own processor, memory, and disks (Figure 33 on page 81). All the machines are connected by a communications facility. This environment is referred to by many different names, including: cluster, cluster of uniprocessors, massively parallel processing (MPP) environment, and shared-nothing configuration. The latter name accurately reflects the arrangement

of resources in this environment. Unlike an SMP environment, an MPP environment has no shared memory or disks. The MPP environment removes the limitations introduced through the sharing of memory and disks.

A partitioned database environment allows a database to remain a logical whole, despite being physically divided across more than one database partition. The fact that data is distributed remains transparent to most users. Work can be divided among the database managers; each database manager in each database partition works against its own part of the database.

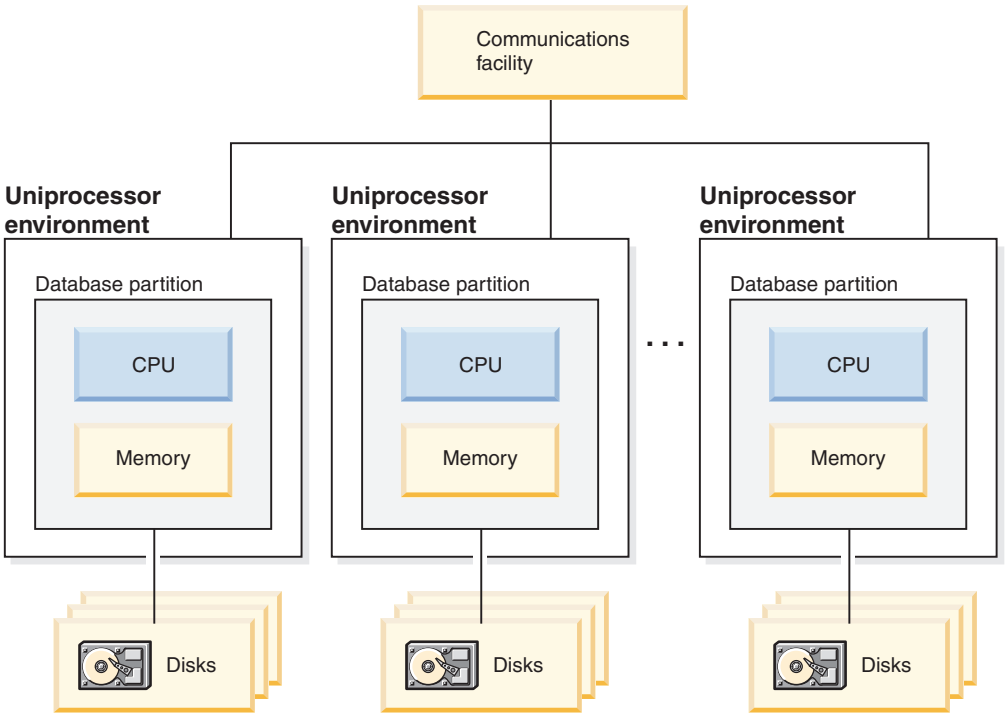


Figure 33. Massively parallel processing (MPP) environment

Capacity and scalability

In this environment you can add more database partitions (nodes) to your configuration. On some platforms, for example the RS/6000® SP, the maximum number is 512 nodes. However, there may be practical limits on managing a high number of machines and instances.

If you have reached maximum capacity or scalability, you can consider moving to a system where each database partition has multiple processors.

Database partitions with multiple processors

An alternative to a configuration in which each database partition has a single processor, is a configuration in which each database partition has multiple processors. This is known as an *SMP cluster* (Figure 34 on page 82).

This configuration combines the advantages of SMP and MPP parallelism. This means that a query can be performed in a single database partition across multiple processors. It also means that a query can be performed in parallel across multiple database partitions.

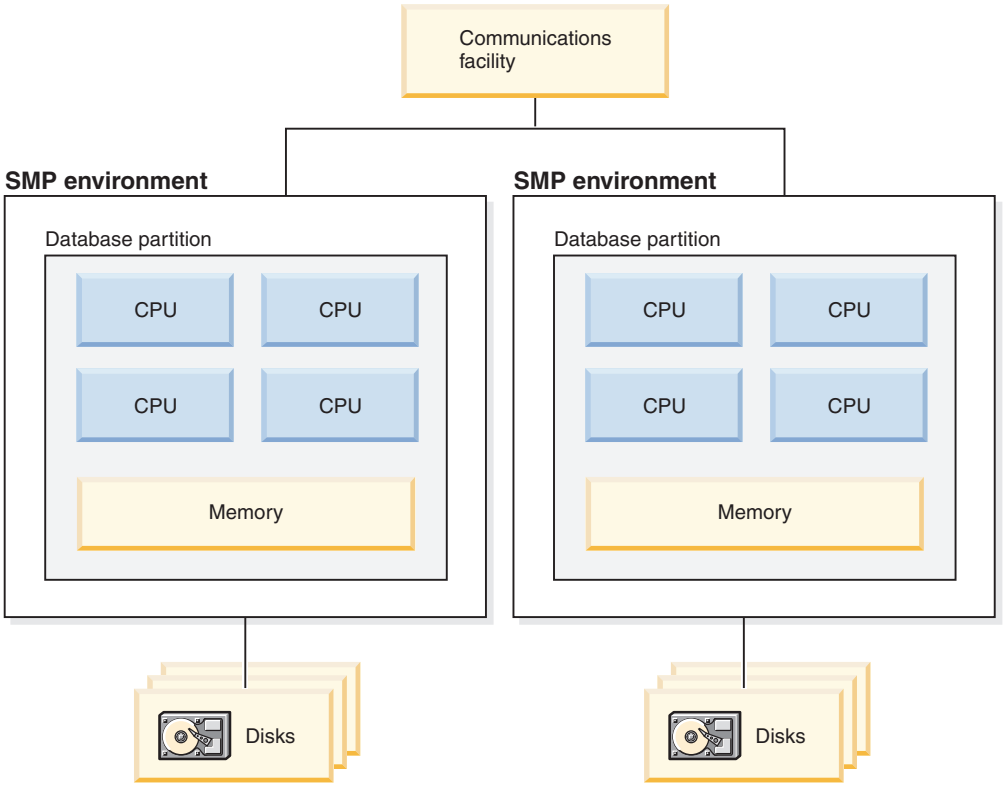


Figure 34. Several symmetric multiprocessor (SMP) environments in a cluster

Capacity and scalability

In this environment you can add more database partitions, and you can add more processors to existing database partitions.

Logical database partitions

A logical database partition differs from a physical partition in that it is not given control of an entire machine. Although the machine has shared resources, database partitions do not share the resources. Processors are shared but disks and memory are not.

Logical database partitions provide scalability. Multiple database managers running on multiple logical partitions may make fuller use of available resources than a single database manager could. Figure 35 on page 83 illustrates the fact that you may gain more scalability on an SMP machine by adding more database partitions; this is particularly true for machines with many processors. By distributing the database, you can administer and recover each database partition separately.

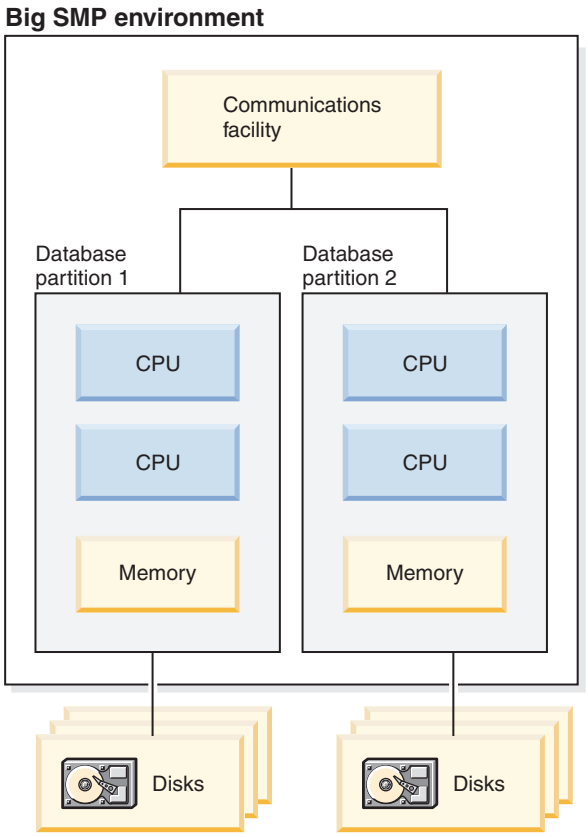


Figure 35. Partitioned database with symmetric multiprocessor environment

Figure 36 on page 84 illustrates that you can multiply the configuration shown in Figure 35 to increase processing power.

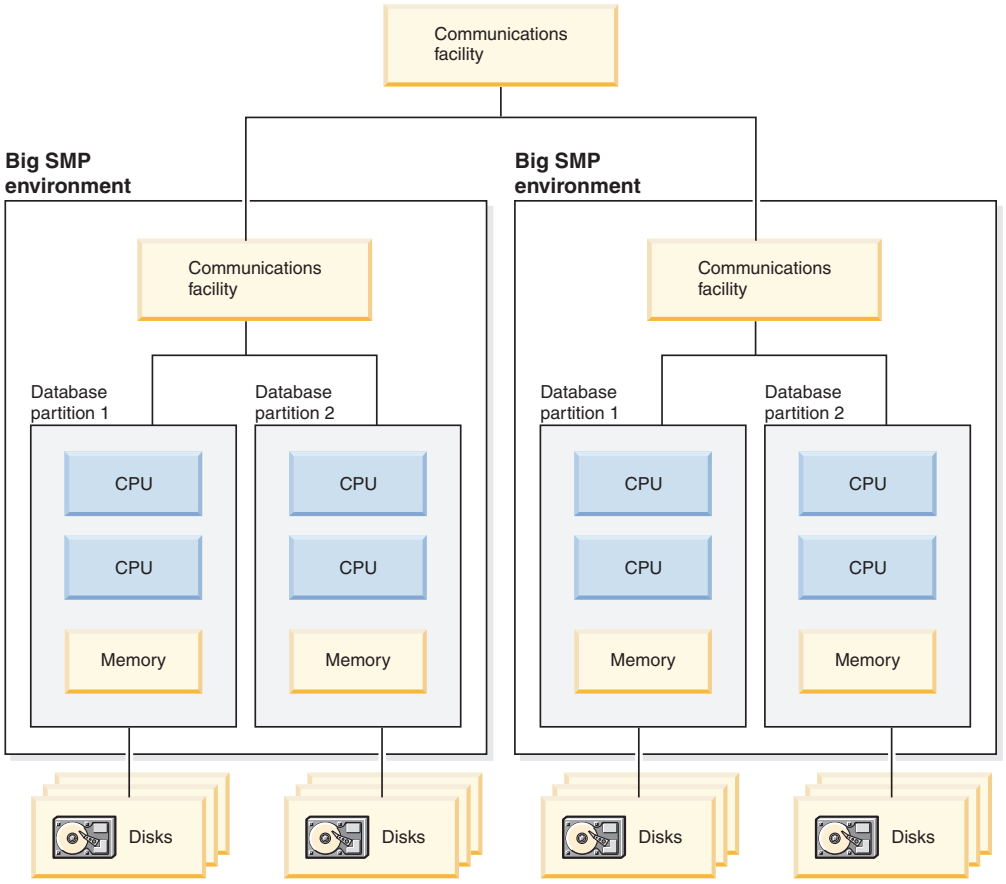


Figure 36. Partitioned database with symmetric multiprocessor environments clustered together

Note: The ability to have two or more database partitions coexist on the same machine (regardless of the number of processors) allows greater flexibility in designing high availability configurations and failover strategies. Upon machine failure, a database partition can be automatically moved and restarted on a second machine that already contains another database partition of the same database.

Summary of parallelism best suited to each hardware environment

The following table summarizes the types of parallelism best suited to take advantage of the various hardware environments.

Table 8. Types of Parallelism Possible in Each Hardware Environment

Hardware Environment	I/O Parallelism	Intra-Query Parallelism	
		Intra-Partition Parallelism	Inter-Partition Parallelism
Single Database Partition, Single Processor	Yes	No(1)	No
Single Database Partition, Multiple Processors (SMP)	Yes	Yes	No
Multiple Database Partitions, One Processor (MPP)	Yes	No(1)	Yes

Table 8. Types of Parallelism Possible in Each Hardware Environment (continued)

Hardware Environment	I/O Parallelism	Intra-Query Parallelism	
		Intra-Partition Parallelism	**Inter-Partition Parallelism**
Multiple Database Partitions, Multiple Processors (cluster of SMPs)	Yes	Yes	Yes
Logical Database Partitions	Yes	Yes	Yes
Note: (1) There may be an advantage to setting the degree of parallelism (using one of the configuration parameters) to some value greater than one, even on a single processor system, especially if the queries you execute are not fully utilizing the CPU (for example, if they are I/O bound).			

Part 2. Installation considerations

© Copyright IBM Corp. 1993, 2009

Chapter 5. Installation prerequisites

Installing DB2 servers (Windows)

This task describes how to start the DB2 Setup wizard on Windows. You will use the DB2 Setup wizard to define your installation and install your DB2 database product on your system.

Before you begin

Before you start the DB2 Setup wizard:

- If you are planning on setting up a partitioned database environment, refer to ″Setting up a partitioned database environment″.
- Ensure that your system meets installation, memory, and disk requirements.
- If you are planning to use LDAP on Windows to register the DB2 server in Active Directory, you should extend the directory schema before you install otherwise you will need to manually register the node and catalog the databases. For more information, see the “Extending the Active Directory Schema for LDAP directory services (Windows)” topic.
- You must have a local Administrator user account with the recommended user rights to perform the installation. In DB2 database servers where LocalSystem can be used as the DAS and DB2 instance user and you are not using the database partitioning feature, a non-administrator user with elevated privileges can perform the installation.

  Note: If a non-Administrator user account is going to do the product installation, then the VS2005 runtime library must be installed before attempting to install a DB2 database product. The VS2005 runtime library is needed on the operating system before the DB2 database product can be installed. The VS2005 runtime library is available from the Microsoft® runtime library download web site. There are two choices: choose vcredist_x86.exe for 32-bit systems or vcredist_x64.exe for 64-bit systems.
- Although not mandatory, it is recommended that you close all programs so that the installation program can update any files on the computer without requiring a reboot.

Restrictions

- The DB2 copy name and the instance name cannot start with a numeric value.
- The DB2 copy name and the instance name must be unique among all DB2 copies.
- The use of XML features is restricted to a database that has only one database partition.
- No other DB2 database product can be installed in the same path if one of the following is already installed:
  - IBM® Data Server Runtime Client
  - IBM Data Server Driver Package
  - *DB2 Information Center*
- The DB2 Setup wizard fields do not accept non-English characters.
- If you enable extended security on Windows Vista, users must belong to the DB2ADMNS or DB2USERS group to run local DB2 commands and applications

© Copyright IBM Corp. 1993, 2009

because of an extra security feature (User Access Control) that limits the privileges that local administrators have by default. If users do not belong to one of these groups, they will not have read access to local DB2 configuration or application data.

Procedure

To start the DB2 Setup wizard:

1. Log on to the system with the local Administrator account that you have defined for the DB2 installation.
2. If you the DB2 database product DVD, insert it into the drive. If enabled, the auto-run feature automatically starts the DB2 Setup Launchpad. If the auto-run does not work, use Windows Explorer to browse the DB2 database product DVD and double-click on the setup icon to start the DB2 Setup Launchpad.
3. If you downloaded the DB2 database product from passport advantage, run the executable file to extract the DB2 database product installation files. Use Windows Explorer to browse the DB2 installation files and double-click on the setup icon to start the DB2 Setup Launchpad.
4. From the DB2 Setup launchpad, you can view installation prerequisites and the release notes, or you can proceed directly to the installation. You may want to review the installation prerequisites and release notes for late-breaking information.
5. Click **Install a Product** and the Install a Product window will display the products available for installation.

   If you have no existing DB2 database products installed on your computer, launch the installation by clicking **Install New**. Proceed through the installation following the DB2 Setup wizard's prompts.

   If you have at least one existing DB2 database product installed on your computer, you can:

   - Click **Install New** to create a new DB2 copy.
   - Click **Work with Existing** to upgrade an existing DB2 copy, to add functionality to an existing DB2 copy, migrate an existing DB2 Version 8 or Version 9.1 copy, or to install an add-on product.
6. The DB2 Setup wizard will determine the system language, and launch the setup program for that language. Online help is available to guide you through the remaining steps. To invoke the online help, click **Help** or press **F1**. You can click **Cancel** at any time to end the installation.

Results

Your DB2 database product will be installed, by default, in the *Program_Files*\IBM\sqllib directory, where *Program_Files* represents the location of the Program Files directory.

If you are installing on a system where this directory is already being used, the DB2 database product installation path will have *_xx* added to it, where *xx* are digits, starting at 01 and increasing depending on how many DB2 copies you have installed.

You can also specify your own DB2 database product installation path.

What to do next

- Verify your installation.

- Perform the necessary post-installation tasks.

For information on errors encountered during installation, review the installation log file located in the My Documents\DB2LOG\ directory. The log file uses the following format: DB2-*ProductAbrrev-DateTime*.log, for example, DB2-ESE-Tue Apr 04 17_04_45 2006.log.

If you want your DB2 database product to have access to DB2 documentation either on your local computer or on another computer on your network, then you must install the *DB2 Information Center*. The *DB2 Information Center* contains documentation for the DB2 database system and DB2 related products. By default, DB2 information will be accessed from the web if the *DB2 Information Center* is not locally installed.

DB2 Express Edition and DB2 Workgroup Server Edition memory limits

If you are installing DB2 Express Edition, the maximum allowed memory for the instance is 4 GB.

If you are installing DB2 Workgroup Server Edition, the maximum allowed memory for the instance is 16 GB.

The amount of memory allocated to the instance is determined by the **INSTANCE_MEMORY** database manager configuration parameter.

Important notes when migrating from Version 9.1:

- If the memory configuration for your Version 9.1 DB2 database product exceeds the allowed limit, the DB2 database product might not start after migrating to the current version.
- The self tuning memory manager will not increase your overall instance memory limit beyond the license limits.

Preparing the environment for a partitioned DB2 server (Windows)

This topic describes the steps required to prepare your Windows environment for a partitioned installation of the DB2 product.

Each participating computer must have the same operating system.

To prepare your Windows environment for installation:

1. Ensure that the primary computer and participating computers belong to the same Windows domain. Check the domain to which the computer belongs by using the System Properties dialog, accessible through the Control Panel.
2. Ensure that time and date settings on the primary computer and participating computers are consistent. To be considered consistent, the difference in GMT time between all computers must be no greater than one hour.

   System date and time can be modified using the Date/Time Properties dialog, accessible through the Control Panel. You can use the `max_time_diff` configuration parameter to change this restriction. The default is `max_time_diff = 60`, which allows a difference of less than 60 minutes.
3. Ensure that each computer object that participates in the partitioned database environment has the ″Trust computer for delegation″ privilege flagged. You can verify that the ″Trust computer for delegation″ checkbox on the General tab of each computer's account Properties dialog box in the Active Directory Users and Computers console is checked.

4. Ensure that all participating computers can communicate with each other using TCP/IP:
   a. On one participating computer, enter the hostname command, which will return the hostname of the computer.
   b. On another participating computer, enter the following command:

      ```
      ping hostname
      ```

      where *hostname* represents the hostname of the primary computer. If the test is successful, you will receive output similar to the following:

      ```
      Pinging ServerA.ibm.com [9.21.27.230] with 32 bytes of data:

      Reply from 9.21.27.230: bytes=32 time<10ms TTL=128
      Reply from 9.21.27.230: bytes=32 time<10ms TTL=128
      Reply from 9.21.27.230: bytes=32 time<10ms TTL=128
      ```

      Repeat these steps until you are sure that all participating computers can communicate with each other using TCP/IP. Each computer must have a static IP address.

      If you are planning to use multiple network adapters, you can specify which adapter to use to communicate between database partition servers. Use thedb2nchg command to specify the netname field in the db2nodes.cfg file after the installation is complete.
5. During the installation you will be asked to provide a DB2 Administration Server user account. This is a local or domain user account that will be used by the DB2 Administration Server (DAS). The DAS is an administration service used to support the GUI tools and assist with administration tasks. You can define a user now or have the DB2 Setup wizard create one for you. If you want to create a new domain user using the DB2 Setup wizard, the account used to perform the installation must have authority to create domain users.
6. On the primary computer, where you will install the instance-owning partition, you must have a domain user account that belongs to the local *Administrators* group. You will log on as this user when you install DB2. You must add the same user account to the local *Administrators* group on each participating computer. This user must have the *Act as part of the operating system* user right.
7. Ensure that all computers in the instance have the database directory on the same local drive letter. You can check this condition by running the GET DATABASE CONFIGURATION command and verifying the value of the DFTDBPATH DBM configuration parameter.
8. During the installation you will be asked to provide a domain user account to be associated with the DB2 instance. Every DB2 instance has one user assigned. DB2 logs on with this user name when the instance is started. You can define a user now, or you can have the DB2 Setup wizard create a new domain user for you.

   When adding a new node to a partitioned environment the DB2 copy name must be the same on all computers.

   If you want to create a new domain user using the DB2 Setup wizard, the account used to perform the installation must have authority to create domain users. The instance user domain account must belong to the local *Administrators* group on all the participating computers and will be granted the following user rights:

   - Act as part of the operating system
   - Create token object
   - Lock pages in memory

- Log on as a service
- Increase quotas
- Replace a process level token

If extended security was selected, then the account must also be a member of the DB2ADMNS group. The DB2ADMNS group already has these privileges so the privileges would not be required to be added explicitly to the account.

Fast communications manager (Windows)

The fast communications manager (FCM) provides communications support for DB2 server products belonging to the same instance. Each database partition server has one FCM sender, and one FCM receiver daemon to provide communications between database partition servers to handle agent requests and to deliver message buffers. The FCM daemon is started when you start the instance.

If communications fail between database partition servers or if they re-establish communications, the FCM thread updates information (that you can query with the database system monitor) and causes the appropriate action (such as the rollback of an affected transaction) to be performed. You can use the database system monitor to help you set the FCM configuration parameters.

You can specify the number of FCM message buffers with the *fcm_num_buffers* database manager configuration parameter and the number of FCM channels with the *fcm_num_channels* database manager configuration parameter. The *fcm_num_buffers* and *fcm_num_channels* database manager configuration parameter are set to AUTOMATIC as the default value. FCM monitors resource usage when any of these parameter are set to automatic, and incrementally releases resources. It is recommended to leave these parameters set to AUTOMATIC.

An overview of installing your DB2 server product (Linux and UNIX)

This topic outlines the steps for installing your DB2 server product on AIX®, HP-UX, Linux, and Solaris.

To install your DB2 server product:

1. Review your DB2 product prerequisites.
2. Review DB2 migration information if applicable.
3. Modify kernel parameters on HP-UX, Linux, and Solaris. On all platforms, except for Linux on x86_32, you must install a 64-bit kernel before proceeding with the installation, otherwise the installation will fail.
4. Prepare the installation media:

   Product DVD
   If the DB2 product DVD does not automount, mount your DB2 product DVD.

   Installation image
   If you downloaded an installation image, untar the file.
5. Install your DB2 product using one of the available methods:
   - The DB2 Setup wizard
   - The db2_install command
   - A silent installation using a response file
   - Payload file deployment

For DB2 servers, you can use the DB2 Setup wizard to perform installation and configuration tasks, such as:

- Selecting DB2 installation type (typical, compact, or custom).
- Selecting DB2 product installation location.
- Install the languages that you can specify later as the default language for the product interface and messages.
- Install or upgrade the IBM Tivoli® System Automation for Multiplatforms Base Component (Linux and AIX).
- Setting up a DB2 instance.
- Setting up the DB2 Administration Server (including DAS user setup).
- Setting up the DB2 Text Search server.
- Setting up Administration contact and health monitor notification.
- Setting up and configuring your instance setup and configuration (including instance user setup).
- Setting up Informix data source support.
- Preparing the DB2 tools catalog.
- Specify the DB2 Information Center port.
- Creating response files.

6. If you installed a DB2 server using a method other than the DB2 Setup wizard, post-installation configuration steps are required.

DB2 installation methods

This topic provides information about DB2 installation methods. The following table shows the installation methods that are available by operating system.

Table 9. Installation method by operating system.

Installation method	Windows	Linux or UNIX
DB2 Setup wizard	Yes	Yes
Response file installation	Yes	Yes
db2_install command	No	Yes
Payload file deployment	No	Yes

The following list describes DB2 installation methods.

DB2 Setup wizard
The DB2 Setup wizard is a GUI installer available on Linux, UNIX, and Windows operating systems. The DB2 Setup wizard provides an easy-to-use interface for installing DB2 products and for performing initial setup and configuration tasks.

The DB2 Setup wizard can also create DB2 instances and response files that can be used to duplicate this installation on other machines.

Note: For non-root installations on Linux and UNIX platforms, only one DB2 instance can exist. The DB2 Setup wizard automatically creates the non-root instance.

On Linux and UNIX platforms, an X server is required to display the DB2 Setup wizard.

Response file installation
A response file is a text file that contains setup and configuration values.

The file is read by the DB2 setup program and the installation is performed according to the values that have been specified.

A response file installation is also referred to as a silent installation.

Another advantage to response files is that they provide access to parameters that cannot be set using the DB2 Setup wizard.

On Linux and UNIX operating systems, if you embed the DB2 installation image in your own application, it is possible for your application to receive installation progress information and prompts from the installer in computer-readable form. This behavior is controlled by the INTERACTIVE response file keyword.

There are a number of ways to create a response file:

Using the response file generator (Windows platforms)
On Windows, you can use the response file generator to create a response file that replicates an existing installation. For example, you might install an IBM data server client, fully configure the client, then generate a response file to replicate the installation and configuration of the client to other computers.

Using the DB2 Setup wizard
The DB2 Setup wizard can create a response file based on the selections you make as you proceed through the DB2 Setup wizard. Your selections are recorded in a response file that you can save to a location on your system. If you select a partitioned database installation, two response files will be generated, one for the instance-owning computer and one for participating computers.

One benefit of this installation method is that you can create a response file without performing an installation. This feature can be useful to capture the options required to install the DB2 product. The response file can be used at a later time to install the DB2 product according to the exact options you specified.

You can export a client or server profile with the db2cfexp command to save your client or server configuration, and then easily import the profile using the db2cfimp command. A client or server profile exported with the db2cfexp command can also be imported during a response file installation using the CLIENT_IMPORT_PROFILE keyword.

You should export the client or server profile after performing the installation and cataloging any data sources.

Customizing the sample response files that are provided for each DB2 product
An alternative to using the response file generator or the DB2 Setup wizard to create a response file is to manually modify a sample response file. Sample response files are provided on the DB2 product DVD. The sample response files provide details about all the valid keywords for each product.

db2_install command (Linux and UNIX platforms only)
The db2_install command installs *all* components for the DB2 product you specify with the English interface support. You can select additional languages to support with the `-L` parameter. You cannot select or deselect components.

Although the db2_install command installs all components for the DB2 product you specify, it does not perform user and group creation, instance creation, or configuration. This method of installation might be preferred in cases where configuration is to be done after installation. If you would rather configure your DB2 product while installing it, consider using the DB2 Setup wizard.

On Linux and UNIX operating systems, if you embed the DB2 installation image in your own application, it is possible for your application to receive installation progress information and prompts from the installer in computer-readable form.

This installation methods requires manual configuration after the product files are deployed.

Payload file deployment (Linux and UNIX only)
This method is an advanced installation method that is not recommended for most users. It requires the user to physically install payload files. A payload file is a compressed tarball that contains all of the files and metadata for an installable component.

This installation methods requires manual configuration after the product files are deployed.

Note: DB2 product installations are no longer operating system packages on Linux and UNIX platforms. As a result, you can no longer use operating system commands for installation. Any existing scripts that you use to interface and query with DB2 installations will need to change.

Installing DB2 servers using the DB2 Setup wizard (Linux and UNIX)

This task describes how to start the DB2 Setup wizard on Linux and UNIX operating systems. The DB2 Setup wizard is used to define your installation preferences and to install your DB2 database product on your system.

Before you begin

Before you start the DB2 Setup wizard:

- If you are planning on setting up a partitioned database environment, refer to "Setting up a partitioned database environment" in *Quick Beginnings for DB2 Servers*
- Ensure that your system meets installation, memory, and disk requirements.
- You can install a DB2 database server using either root or non-root authority. For more information on non-root installation, see "Non-root installation overview (Linux and UNIX)" in *Quick Beginnings for DB2 Servers*.
- The DB2 database product image must be available. You can obtain a DB2 installation image either by purchasing a physical DB2 database product DVD, or by downloading an installation image from Passport Advantage®.
- If you are installing a non-English version of a DB2 database product, you must have the appropriate National Language Packages.
- The DB2 Setup wizard is a graphical installer. You must have X windows software capable of rendering a graphical user interface for the DB2 Setup wizard to run on your machine. Ensure that the X windows server is running. Ensure that you have properly exported your display. For example, `export DISPLAY=9.26.163.144:0.`

- If you are using security software in your environment, you must manually create required DB2 users before you start the DB2 Setup wizard.

Restrictions

- The use of XML features is restricted to a database that is defined with the code set UTF-8 and has only one database partition.
- The DB2 Setup wizard fields do not accept non-English characters.

Procedure

To start the DB2 Setup wizard:

1. If you have a physical DB2 database product DVD, change to the directory where the DB2 database product DVD is mounted by entering the following command:

   ```
   cd /dvdrom
   ```

   where */dvdrom* represents the mount point of the DB2 database product DVD.
2. If you downloaded the DB2 database product image, you must decompress and untar the product file.
   a. Decompress the product file:

      ```
      gzip -d product.tar.gz
      ```

      where *product* is the name of the product that you downloaded.
   b. Untar the product file:

      On Linux operating systems

      ```
      tar -xvf product.tar
      ```

      On AIX, HP-UX, and Solaris operating systems

      ```
      gnutar -xvf product.tar
      ```

      where *product* is the name of the product that you downloaded.
   c. Change directory:

      ```
      cd ./product
      ```

      where *product* is the name of the product that you downloaded.

   Note: If you downloaded a National Language Package, untar it into the same directory. This will create the subdirectories (for example ./nlpack/disk1) in the same directory, and allows the installer to automatically find the installation images without prompting.
3. Enter the ./db2setup command from the directory where the database product image resides to start the DB2 Setup wizard.
4. The IBM DB2 Setup Launchpad opens. From this window, you can view installation prerequisites and the release notes, or you can proceed directly to the installation. You may want to review the installation prerequisites and release notes for late-breaking information.
5. Click **Install a Product** and the **Install a Product** window will display the products available for installation.

   Launch the installation by clicking **Install New**. Proceed through the installation following the DB2 Setup wizard's prompts.

   Once you have initiated the installation, proceed through the DB2 Setup wizard installation panels and make your selections. Installation help is available to

guide you through the remaining steps. To invoke the installation help, click **Help** or press F1. You can click **Cancel** at any time to end the installation.

Results

For non-root installations, DB2 database products are always installed in the $*HOME*/sqllib directory, where $*HOME* represents the non-root user's home directory.

For root installations, DB2 database products are installed, by default, in one of the following directories:

AIX, HP-UX, and Solaris
/opt/IBM/db2/V9.5

Linux /opt/ibm/db2/V9.5

If you are installing on a system where this directory is already being used, the DB2 database product installation path will have *_xx* added to it, where *_xx* are digits, starting at 01 and increasing depending on how many DB2 copies you have installed.

You can also specify your own DB2 database product installation path.

DB2 installation paths have the following rules:

- Can include lowercase letters (a–z), uppercase letters (A–Z), and the underscore character (_)
- Cannot exceed 128 characters
- Cannot contain spaces
- Cannot contain non-English characters

The installation log files are:

- The DB2 setup log file. This file captures all DB2 installation information including errors.
  - For root installations, the DB2 setup log file name is db2setup.log.
  - For non-root installations, the DB2 setup log file name is db2setup_*username*.log, where *username* is the non-root user ID under which the installation was performed.
- The DB2 error log file. This file captures any error output that is returned by Java™ (for example, exceptions and trap information).
  - For root installations, the DB2 error log file name is db2setup.err.
  - For non-root installations, the DB2 error log file name is db2setup_*username*.err, where *username* is the non-root user ID under which the installation was performed.

By default, these log files are located in the /tmp directory. You can specify the location of the log files.

There is no longer a db2setup.his file. Instead, the DB2 installer saves a copy of the DB2 setup log file in the DB2_DIR/install/logs/ directory, and renames it db2install.history. If the name already exists, then the DB2 installer renames it db2install.history.*xxxx*, where *xxxx* is 0000-9999, depending on the number of installations you have on that machine.

Each installation copy has a separate list of history files. If an installation copy is removed, the history files under this install path will be removed as well. This copying action is done near the end of the installation and if the program is stopped or aborted before completion, then the history file will not be created.

What to do next

- Verify your installation.
- Perform the necessary post-installation tasks.

National Language Packs can also be installed by running the ./db2setup command from the directory where the National Language Pack resides, after a DB2 database product has been installed.

On Linux x86, if you want your DB2 database product to have access to DB2 documentation either on your local computer or on another computer on your network, then you must install the *DB2 Information Center*. The *DB2 Information Center* contains documentation for the DB2 database system and related products.

DB2 Express Edition and DB2 Workgroup Server Edition memory limits

If you are installing DB2 Express Edition, the maximum allowed memory for the instance is 4 GB.

If you are installing DB2 Workgroup Server Edition, the maximum allowed memory for the instance is 16 GB.

The amount of memory allocated to the instance is determined by the **INSTANCE_MEMORY** database manager configuration parameter.

Important notes when migrating from Version 9.1:

- If the memory configuration for your Version 9.1 DB2 database product exceeds the allowed limit, the DB2 database product might not start after migrating to the current version.
- The self tuning memory manager will not increase your overall instance memory limit beyond the license limits.

Fast communications manager (Linux and UNIX)

The fast communications manager (FCM) provides communications support for DB2 server products that use the Database Partitioning Feature (DPF).

For multiple partition instances, each database partition server has one FCM sender daemon and one FCM receiver daemon to provide communications between database partition servers to handle agent requests and to deliver message buffers. The FCM daemon is started when you start the multiple partition instance.

If communications fail between database partition servers or if they re-establish communications, the FCM daemons update information. You can query this information with the database system monitor. The FCM daemons also trigger the appropriate action. An example of an appropriate action is the rollback of an affected transaction. You can use the database system monitor to help you set the FCM configuration parameters.

You can specify the number of FCM message buffers with the *fcm_num_buffers* database manager configuration parameter. You can also specify the number of FCM channels with the *fcm_num_channels* database manager configuration parameter. The *fcm_num_buffers* and *fcm_num_channels* database manager configuration parameters are set to AUTOMATIC as the default value. The FCM

monitors resource usage when any of these parameter are set to automatic, and incrementally releases resources. It is recommended to leave these parameters set to AUTOMATIC.

Chapter 6. Before you install

Additional partitioned database environment preinstallation tasks (Linux and UNIX)

Updating environment settings for a partitioned DB2 installation (AIX)

This task describes the environment settings that you need to update on each computer that will participate in your partitioned database system.

To update AIX environment settings:

1. Log on to the computer as a user with `root` authority.
2. Set the AIX `maxuproc` (maximum number of processes per user) device attribute to *4096* by entering the following command:

   ```
   chdev -l sys0 -a maxuproc='4096'
   ```

   Note: A `bosboot/reboot` may be required to switch to the 64-bit kernel if a different image is being run.
3. Set the TCP/IP network parameters on all the workstations that are participating in your partitioned database system to the following values. These values are the minimum values for these parameters. If any of the network-related parameters are already set to a higher value, do not change it.

   ```
   thewall       = 65536
   sb_max        = 1310720
   rfc1323       = 1
   tcp_sendspace = 221184
   tcp_recvspace = 221184
   udp_sendspace = 65536
   udp_recvspace = 65536
   ipqmaxlen     = 250
   somaxconn     = 1024
   ```

   To list the current settings of all network-related parameters, enter the following command:

   ```
   no -a | more
   ```

   To set a parameter, enter the follow command:

   ```
   no -o parameter_name=value
   ```

   where:

   - *parameter_name* represents the parameter you want to set.
   - *value* represents the value that you want to set for this parameter.

   For example, to set the `tcp_sendspace` parameter to `221184`, enter the following command:

   ```
   no -o tcp_sendspace=221184
   ```
4. If you are using a high speed interconnect, you must set the *spoolsize* and *rpoolsize* for *css0* to the following values:

   ```
   spoolsize     16777216
   rpoolsize     16777216
   ```

© Copyright IBM Corp. 1993, 2009

To list the current settings of these parameters, enter the following command:

```
lsattr -l css0 -E
```

To set these parameters, enter the following commands:

```
/usr/lpp/ssp/css/chgcss -l css0 -a spoolsize=16777216
/usr/lpp/ssp/css/chgcss -l css0 -a rpoolsize=16777216
```

If you are not using the /tftpboot/tuning.cst file to tune your system, you can use the DB2DIR/misc/rc.local.sample sample script file, where DB2DIR is path where the DB2 product has been installed to, to update the network-related parameters after installation. To update the network-related parameters using the sample script file after installation, perform the following steps:

a. Copy this script file to the /etc directory and make it executable by root by entering the following commands:

```
cp /usr/opt/db2_09_01/misc/rc.local.sample /etc/rc.local
chown root:sys /etc/rc.local
chmod 744 /etc/rc.local
```

b. Review the /etc/rc.local file and update it if necessary.

c. Add an entry to the /etc/inittab file so that the /etc/rc.local script is executed whenever the machine is rebooted. You can use the mkitab command to add an entry to the /etc/inittab file. To add this entry, enter the following command:

```
mkitab "rclocal:2:wait:/etc/rc.local > /dev/console 2>&1"
```

d. Ensure that /etc/rc.nfs entry is included in the /etc/inittab file by entering the following command:

```
lsitab rcnfs
```

e. Update the network parameters without rebooting your system by entering the following command:

```
/etc/rc.local
```

5. Ensure that you have enough paging space for a partitioned installation of DB2 ESE to run. If you do not have sufficient paging space, the operating system will kill the process that is using the most virtual memory (this is likely to be one of the DB2 processes). To check for available paging space, enter the following command:

```
lsps -a
```

This command will return output similar to the following:

```
Page Space  Physical Volume  Volume Group  Size  %Used  Active  Auto  Type
paging00    hdisk1          rootvg        60MB     19     yes   yes    lv
hd6         hdisk0          rootvg        60MB     21     yes   yes    lv
hd6         hdisk2          rootvg        64MB     21     yes   yes    lv
```

The paging space available should be equal to twice the amount of physical memory installed on your computer.

6. If you are creating a small to intermediate size partitioned database system, the number of network file system daemons (NFSDs) on the instance-owning computer should be close to:

```
# of biod on a computer × # of computers in the instance
```

Ideally, you should run 10 biod processes on every computer. According to the above formula, on a four computer system with 10 biod processes, you would use 40 NFSDs.

If you are installing a larger system, you can have up to 120 NFSDs on the computer.

For additional information about NFS, refer to your NFS documentation.

Setting up a working collective to distribute commands to ESE workstations (AIX)

In a partitioned database environment on AIX, you can set up a working collective to distribute commands to the set of RS/6000 SP™ workstations that participate in your partitioned database system. Commands can be distributed to the workstations by the dsh command.

This can be useful when installing or administrating a partitioned database system on AIX, to enable you to quickly execute the same commands on all the computers in your environment with less opportunity for error.

You must know the hostname of each computer that you want to include in the working collective.

You must be logged on to the Control workstation as a user with `root` authority.

Have a file that lists the hostnames for all of the RS/6000 SP workstations that will participate in your partitioned database system. To set up the working collective to distribute commands to this list of workstations:

1. Create a file called eeelist.txt that will list the *hostnames* for all of the workstations that will participate in the working collective.

   For example, assume that you wanted to create a working collective with two SP nodes called `workstation1` and `workstation2`. The contents of eeelist.txt would be:

   ```
   workstation1
   workstation2
   ```

2. Update the working collective environment variable. To update this list, enter the following command:

   ```
   export WCOLL=path/eeelist.txt
   ```

   where *path* is the location where *eeelist.txt* was created, and *eeelist.txt* is the name of the file that you created that lists the RS/6000SP workstations in the working collective.

3. Verify that the names in the working collective are indeed the workstations that you want, by entering the following command:

   ```
   dsh -q
   ```

   You will receive output similar to the following:

   ```
   Working collective file /eeelist.txt:
   workstation1
   workstation2
   Fanout: 64
   ```

Verifying that NFS is running (Linux and UNIX)

Before setting up a database partitioned environment, you should verify that Network File System (NFS) is running on each computer that will participate in your partitioned database system.

NFS must be running on each computer.

To verify that NFS is running on each computer:

AIX operating systems

Type the following command on each computer:

```
lssrc -g nfs
```

The `Status` field for NFS processes should indicate `active`.

After you have verified that NFS is running on each system, you should check for the specific NFS processes required by DB2 products. The required processes are:

```
rpc.lockd
rpc.statd
```

HP-UX and Solaris operating systems

Type the following command on each computer:

```
showmount -e hostname
```

Enter the showmount command without the *hostname* parameter to check the local system.

If NFS is not active you will receive a message similar to the following:

```
showmount: ServerA: RPC: Program not registered
```

After you have verified that NFS is running on each system, you should check for the specific NFS processes required by DB2 products:

```
rpc.lockd
rpc.statd
```

You can use the following commands to check for these processes:

```
ps -ef | grep rpc.lockd
ps -ef | grep rpc.statd
```

Linux operating systems

Type the following command on each computer:

```
showmount -e hostname
```

Enter the showmount command without the *hostname* parameter to check the local system.

If NFS is not active you will receive a message similar to the following:

```
showmount: ServerA: RPC: Program not registered
```

After you have verified that NFS is running on each system, you should check for the specific NFS processes required by DB2 products. The required process is `rpc.statd`.

You can use the ps -ef | grep rpc.statd commands to check for this process.

If these processes are not running, consult your operating system documentation.

Verifying port range availability on participating computers (Linux and UNIX)

This task describes the steps required to verify port range availability on participating computers. The port range is used by the Fast Communications Manager (FCM). FCM is a feature of DB2 that handles communications between database partition servers.

Verifying the port range availability on participating computers should be done after you install the instance-owning database partition server and before you install any participating database partition servers.

When you install the instance-owning database partition server on the primary computer, DB2 reserves a port range according to the specified number of logical database partition servers participating in partitioned database environment. The default range is four ports. For each server that participates in the partitioned database environment, you must manually configure the /etc/services file for the FCM ports. The range of the FCM ports depends on how many logical partitions you want to use on the participating computer. A minimum of two entries are required, **DB2_<instance>** and **DB2_<instance>_END**. Other requirements for the FCM ports specified on participating computers are:

- The starting port number must match the starting port number of the primary computer
- Subsequent ports must be sequentially numbered
- Specified port numbers must be free
-

To make changes to the services file, you require root authority.

To verify the port range availability on participating computers:

1. Open the services file located in the /etc/services directory.
2. Locate the ports reserved for the DB2 Fast Communications Manager (FCM). The entries should appear similar to the following:

   ```
   DB2_db2inst1         60000/tcp
   DB2_db2inst1_1       60001/tcp
   DB2_db2inst1_2       60002/tcp
   DB2_db2inst1_END     60003/tcp
   ```

   DB2 reserves the first four available ports after 60000.
3. On each participating computer, open the services file and verify that the ports reserved for DB2 FCM in the services file of the primary computer are not being used.
4. In the event that the required ports are in use on a participating computer, identify an available port range for all computers and update each service file, including the services file on the primary computer.

After you install the instance-owning database partition server on the primary computer, you must install your DB2 product on the participating database partition servers. You can use the response file generated for the partitioning servers (default name is db2ese_addpart.rsp), you need to manually configure the /etc/services files for the FCM ports. The range of the FCM ports depend on how many logical partitions you want to use on the current machine. The minimum entries are for DB2_ and DB2__END two entries with consecutive free port numbers. The FCM port numbers used on each participating machines must have the same starting port number, and subsequent ports must be sequentially numbered.

Creating a file system for a partitioned DB2 server (Linux)

This task is part of setting up your partitioned database system. This task describes how to:

- create a DB2 home file system

- NFS export the home file system
- NFS mount the home file system from each participating computer

You must have a file system that is available to all machines that will participate in your partitioned database system. This file system will be used as the instance home directory.

For configurations that use more than one machine for a single database instance, NFS (Network File System) is used to share this file system. Typically, one machine in a cluster is used to export the file system using NFS, and the remaining machines in the cluster mount the NFS file system from this machine. The machine that exports the file system has the file system mounted locally.

For more command information, see your Linux distribution documentation.

To create this file system:

1. On one machine, select a disk partition or create one using fdisk.
2. Using a utility like mkfs, create a file system on this partition. The file system should be large enough to contain the necessary DB2 program files as well as enough space for your database needs.
3. Locally mount the file system you have just created and add an entry to the /etc/fstab file so that this file system is mounted each time the system is rebooted. For example:

   ```
   /dev/hda1    /db2home    ext3   defaults   1 2
   ```

4. To automatically export an NFS file system on Linux at boot time, add an entry to the /etc/exports file. Be sure to include all of the host names participating in the cluster as well as all of the names that a machine might be known as. Also, ensure that each machine in the cluster has root authority on the exported file system by using the ″root″ option.

   The /etc/exports file is an ASCII file which contains the following type of information:

   ```
   /db2home machine1_name(rw) machine2_name(rw)
   ```

   To export the NFS directory, run

   ```
   /usr/sbin/exports -r
   ```

5. On each of the remaining machines in the cluster, add an entry to the /etc/fstab file to NFS mount the file system automatically at boot time. As in the following example, when you specify the mount point options, ensure that the file system is mounted at boot time, is read-write, is mounted hard, includes the bg (background) option, and that setuid programs can be run properly.

   ```
   fusion-en:/db2home /db2home nfs rw,timeo=7,
    hard,intr,bg,suid,lock
   ```

   where *fusion-en* represents the machine name.

6. NFS mount the exported file system on each of the remaining machines in the cluster by entering the following command:

   ```
   mount /db2home
   ```

   If the mount command fails, use the showmount command to check the status of the NFS server. For example:

   ```
   showmount -e fusion-en
   ```

   where *fusion-en* represents the machine name.

This showmount command should list the file systems which are exported from the machine named `fusion-en`. If this command fails, the NFS server may not have been started. Run the following command as root on the NFS server to start the server manually:

```
/etc/rc.d/init.d/nfs restart
```

Assuming the present run level is 3, you can have this command run automatically at boot time by renaming K20nfs to S20nfs under the following directory: /etc/rc.d/rc3.d.

7. Ensure that the following steps were successful:
   a. On a single machine in the cluster, you have created a file system to be used as the instance and home directory.
   b. If you have a configuration that uses more than one machine for a single database instance, you have exported this file system using NFS.
   c. You have mounted the exported file system on each of the remaining machines in the cluster.

Creating a DB2 home file system for a partitioned database system (AIX)

This task is part of setting up your partitioned database system. This task describes how to:

- create a DB2 home file system
- NFS export the home file system
- NFS mount the home file system from each participating computer

It is recommended that you create a home file system that is as large as the content on the DB2 product DVD. You can use the following command to check the size, shown in KB:

```
du -sk <DVD mounting point>
```

A DB2 instance will require at least 50 MB of space. If you do not have enough free space, you can mount the DB2 product DVD from each participating computer as an alternative to copying the contents to disk.

You must have:

- `root` authority to create a file system
- Created a volume group where your file system is to physically reside.

To create, NFS export, and NFS mount the DB2 home file system, perform the following steps:

Creating the DB2 home file system

Log on to the primary computer (ServerA) in your partitioned database system as a user with `root` authority and create a home file system for your partitioned database system called /db2home.

1. Enter the **smit jfs** command.
2. Click on the **Add a Journaled File System** icon.
3. Click on the **Add a Standard Journaled File System** icon.
4. Select an existing volume group from the **Volume Group Name** list where you want this file system to physically reside.

5. Set the SIZE of file system (**SIZE of file system (in 512–byte blocks) (Num.)** field). This sizing is enumerated in 512-byte blocks, so if you only need to create a file system for the instance home directory, you can use 180 000, which is about 90 MB. If you need to copy the product DVD image over to run the installation, you can create it with a value of 2 000 000, which is about 1 GB.
6. Enter the mount point for this file system in the **MOUNT POINT** field. In this example, the mount point is `/db2home`.
7. Set the **Mount AUTOMATICALLY at system restart** field to `yes`.

   The remaining fields can be left to the default settings.
8. Click **OK**.

Exporting the DB2 home file system

1. NFS export the /db2home file system so that it is available to all of the computers that will participate in your partitioned database system:
   a. Enter the **smit nfs** command.
   b. Click on the **Network File System (NFS)** icon.
   c. Click on the **Add a Directory to Exports List** icon.
   d. Enter the pathname and directory to export (for example, `/db2home`) in the **PATHNAME of directory to export** field.
   e. Enter the name of each workstation that will participate in your partitioned database system in the **HOSTS allowed root access** field. Use a comma (,) as the delimiter between each name. For example, `ServerA, ServerB, ServerC`. If you are using a high speed interconnect, it is recommended that you specify the high speed interconnect names for each workstation in this field as well. The remaining fields can be left to the default settings.
   f. Click **OK**.
2. Log out.

Mounting the DB2 home file system from each participating computer

Log on to *each* participating computer (ServerB, ServerC, ServerD) and NFS mount the file system that you exported by performing the following steps:

1. Enter the **smit nfs** command.
2. Click on the **Network File System (NFS)** icon.
3. Click on the **Add a File System for Mounting** icon.
4. Enter the pathname of the mount point in the **PATHNAME of the mount point (Path)** field.

   The path name of the mount point is where you should create the DB2 home directory. For this example, use /db2home.
5. Enter the pathname of the remote directory in the **PATHNAME of the remote directory** field.

   For our example, you should enter the same value that you entered in the **PATHNAME of the mount point (Path)** field.
6. Enter the *hostname* of the machine where you exported the file system in the **HOST where the remote directory resides** field.

   This value is the hostname of the machine where the file system that you are mounting was created.

   To improve performance, you may want to NFS mount the file system that you created over a high speed interconnect. If you want to mount

this file system using a high speed interconnect, you must enter its name in the **HOST where remote directory resides** field.

You should be aware that if the high speed interconnect ever becomes unavailable for some reason, every workstation that participates in your partitioned database system will lose access to the DB2 home directory.

7. Set the **MOUNT now, add entry to /etc/filesystems or both?** field to `both`.
8. Set the **/etc/filesystems entry will mount the directory on system RESTART** field to `yes`.
9. Set the **MODE for this NFS file system** field to `read-write`.
10. Set the **Mount file system soft or hard** field to `soft`.

    A soft mount means that the computer *will not* try for an infinite period of time to remotely mount the directory. A hard mount means that your machine will infinitely try to mount the directory. This could cause problems in the event of a system crash. It is recommended that you set this field to `soft`.

    The remaining fields can be left to the default settings.
11. Ensure that this file system is mounted with the **Allow execution of SUID and sgid programs in this file system?** field set to `Yes`. This is the default setting.
12. Click **OK**.
13. Log out.

Creating required users for a DB2 server installation in a partitioned database environment (Linux)

Three users and groups are required to operate a DB2 database. The user and group names used in the following instructions are documented in the following table. You may specify your own user and group names as long as they adhere to your system naming rules and DB2 naming rules.

If you are planning to use the DB2 Setup wizard to install your DB2 product, the DB2 Setup wizard will create these users for you.

Table 10. Required users and groups

Required user	User name	Group name
Instance owner	db2inst1	db2iadm1
Fenced user	db2fenc1	db2fadm1
DB2 administration server user	dasusr1	dasadm1

If the DB2 administration server user is an existing user, this user must exist on all the participating computers before the installation. If you use the DB2 Setup wizard to create a new user for the DB2 administration server on the instance-owning computer, then the new user is also created (if necessary) during the response file installations on the participating computers. If the user already exists on the participating computers, the user must have the same primary group.

Prerequisites

- You must have `root` authority to create users and groups.

- If you manage users and groups with security software, additional steps might be required when defining DB2 users and groups.

Restriction

The user names you create must conform to both your operating system's naming rules, and those of DB2.

To create all three of these users, perform the following steps:

1. Log on to the primary computer.
2. Create a group for the instance owner (for example, `db2iadm1`), the group that will execute UDFs or stored procedures (for example, `db2fadm1`), and the group that will own the DB2 administration server (for example, `dasadm1`) by entering the following commands:

   ```
   groupadd -g 999 db2iadm1
   groupadd -g 998 db2fadm1
   groupadd -g 997 dasadm1
   ```

   Ensure that the specific numbers you are using do not currently exist on any of the machines.
3. Create a user that belongs to each group that you created in the previous step using the following commands. The home directory for each user will be the DB2 home directory that you previously created and shared (db2home).

   ```
   useradd -u 1004 -g db2iadm1 -m -d /db2home/db2inst1 db2inst1
   useradd -u 1003 -g db2fadm1 -m -d /db2home/db2fenc1 db2fenc1
   useradd -u 1002 -g dasadm1 -m -d /home/dasusr1 dasusr1
   ```
4. Set an initial password for each user that you created by entering the following commands:

   ```
   passwd db2inst1    passwd db2fenc1    passwd dasusr1
   ```
5. Log out.
6. Log on to the primary computer as each user that you created (`db2inst1`, `db2fenc1`, and `dasusr1`). You may be prompted to change each user's password since this is the first time that these users have logged onto the system.
7. Log out.
8. Create the exact same user and group accounts on each computer that will participate in your partitioned database environment.

Creating required users for a DB2 server installation in a partitioned database environment (AIX)

Three users and groups are required to operate a DB2 database. The user and group names used in the following instructions are documented in the following table. You may specify your own user and group names as long as they adhere to your system naming rules and DB2 naming rules.

If you are planning to use the DB2 Setup wizard to install your DB2 product, the DB2 Setup wizard will create these users for you.

Table 11. Required users and groups

Required user	User name	Group name
Instance owner	db2inst1	db2iadm1
Fenced user	db2fenc1	db2fadm1
DB2 administration server user	dasusr1	dasadm1

If the DB2 administration server user is an existing user, this user must exist on all the participating computers before the installation. If you use the DB2 Setup wizard to create a new user for the DB2 administration server on the instance-owning computer, then the new user is also created (if necessary) during the response file installations on the participating computers. If the user already exists on the participating computers, the user must have the same primary group.

Prerequisites

- You must have `root` authority to create users and groups.
- If you manage users and groups with security software, additional steps might be required when defining DB2 users and groups.

Restriction

The user names you create must conform to both your operating system's naming rules, and those of DB2.

To create all three of these users, perform the following steps:

1. Log on to the primary computer.
2. Create a group for the instance owner (for example, `db2iadm1`), the group that will execute UDFs or stored procedures (for example, `db2fadm1`), and the group that will own the DB2 administration server (for example, `dasadm1`) by entering the following commands:

   ```
   mkgroup id=999 db2iadm1
   mkgroup id=998 db2fadm1
   mkgroup id=997 dasadm1
   ```

3. Create a user that belongs to each group that you created in the previous step using the following commands. The home directory for each user will be the DB2 home directory that you previously created and shared (db2home).

   ```
   mkuser id=1004 pgrp=db2iadm1 groups=db2iadm1 home=/db2home/db2inst1
      core=-1 data=491519 stack=32767 rss=-1 fsize=-1 db2inst1
   mkuser id=1003 pgrp=db2fadm1 groups=db2fadm1 home=/db2home/db2fenc1
      db2fenc1
   mkuser id=1002 pgrp=dasadm1 groups=dasadm1 home=/home/dasusr1
      dasusr1
   ```

4. Set an initial password for each user that you created by entering the following commands:

   ```
   passwd db2inst1
   passwd db2fenc1
   passwd dasusr1
   ```

5. Log out.
6. Log on to the primary computer as each user that you created (`db2inst1`, `db2fenc1`, and `dasusr1`). You may be prompted to change each user's password since this is the first time that these users have logged onto the system.
7. Log out.
8. Create the exact same user and group accounts on each computer that will participate in your partitioned database environment.

Chapter 7. Installing your DB2 server product

Setting up a partitioned database environment

This topic describes how to set up a partitioned database environment. You will use the DB2 Setup wizard to install your instance-owning database server and to create the response files that will in turn be used to create your participating database servers.

Before you begin

Note: A partitioned database environment is not supported in non-root installations.

- Ensure that you have the DB2 Warehouse Activation CD license key that will need to be copied over to all participating computers.
- The same number of consecutive ports must be free on each computer that is to participate in the partitioned database environment. For example, if the partitioned database environment will be comprised of four computers, then each of the four computers must have the same four consecutive ports free. During instance creation, a number of ports equal to the number of logical partitions on the current server will be reserved in the `/etc/services` on Linux and UNIX and in the `%SystemRoot%\system32\drivers\etc\services` on Windows. These ports will be used by the Fast Communication Manager. The reserved ports will be in the following format:

  ```
  DB2_InstanceName
  DB2_InstanceName_1
  DB2_InstanceName_2
  DB2_InstanceName_END
  ```

  The only mandatory entries are the beginning (DB2_InstanceName) and ending (DB2_InstanceName_END) ports. The other entries are reserved in the services file so that other applications do not use these ports
- To support multiple participating DB2 database servers, the computer on which you want to install DB2 must belong to an accessible domain. However, you can add local partitions to this computer even though the computer doesn't belong to a domain.
- On Linux and UNIX systems, a remote shell utility is required for partitioned database systems. DB2 database systems support the following remote shell utilities:
  - rsh
  - ssh

  By default, DB2 database systems use rsh when executing commands on remote DB2 nodes, for example, when starting a remote DB2 database partition. To use the DB2 default, the rsh-server package must be installed. For more information, see "Security considerations when installing and using the DB2 database manager" in *Database Security Guide*.

  If you choose to use the rsh remote shell utility, inetd (or xinetd) must be installed and running as well. If you choose to use the ssh remote shell utility, you need to set the **DB2RSHCMD** registry variable immediately after the DB2 installation is complete. If this registry variable is not set, rsh is used.

© Copyright IBM Corp. 1993, 2009

- On Linux and UNIX operating systems, ensure the hosts file under the etc directory does not contain an entry for "127.0.0.2" if that IP address maps to the fully qualified hostname of the machine.

Note: The use of XML features prohibits later use of a partitioned database environment.

About this task

A database partition is part of a database that consists of its own data, indexes, configuration files, and transaction logs. A partitioned database is a database with two or more partitions.

Procedure

To set up a partitioned database environment:

1. Install your instance-owning database server using the DB2 Setup wizard. For detailed instructions, see the appropriate "Installing DB2 servers" topic for your platform.
   - On the **Select installation, response files creation, or both** window, ensure that you select the **Save my installation settings in a response files** option. After the installation has completed, two files will be copied to the directory specified in the DB2 Setup wizard: PROD_ESE.rsp and PROD_ESE_addpart.rsp. The PROD_ESE.rsp file is the response file for instance-owning database servers. The PROD_ESE_addpart.rsp file is the response file for participating database servers.
   - On the **Set up partitioning options for the DB2 instance** window, ensure that you select **Multiple partition instance**, and enter the maximum number of logical partitions.
2. Make the DB2 install image available to all participating computers in the partitioned database environment.
3. Distribute the participating database servers response file (PROD_ESE_addpart.rsp).
4. Install a DB2 database server on each of the participating computers using the db2setup command on Linux and UNIX, or the setup command on Windows:

   Linux and UNIX

   Go to the directory where the DB2 database product code is available and run:

   ```
   ./db2setup -r /responsefile_directory/response_file_name
   ```

   Windows

   ```
   setup -u x:\responsefile_directory\response_file_name
   ```

   For example, here is the command using the PROD_ESE_addpart.rsp as the response file:

   Linux and UNIX

   Go to the directory where the DB2 database product code is available and run:

   ```
   ./db2setup -r /db2home/PROD_ESE_addpart.rsp
   ```

   where /db2home is the directory where you have copied the response file.

   Windows

```
setup -u c:\resp_files\PROD_ESE_addpart.rsp
```

where c:\resp_files\ is the directory where you have copied the response file.

5. (Linux and UNIX only) Configure the db2nodes.cfg file. The DB2 installation only reserves the maximum number of logical partitions you want to use for the current computer, but does not configure the db2nodes.cfg file. If you do not configure the db2nodes.cfg file, the instance is still a single partitioned instance.
6. Update the services file on the participating servers to define the corresponding FCM port for the DB2 instance. The services file is in the following location:
   - /etc/services on Linux and UNIX
   - %SystemRoot%\system32\drivers\etc\services on Windows

Installing database partition servers on participating computers using a response file (Windows)

In this task you will use the response file you created using the DB2 Setup wizard to install database partition servers on participating computers.

Prerequisites

- You have installed a DB2 copy on the primary computer using the DB2 Setup wizard.
- You have created a response file for installing on participating computers and copied it onto the participating computer.
- You must have administrative authority on participating computers.
- You must have copied the contents of the DB2 product DVD onto the participating computer.

To install additional database partition servers using a response file:

1. Log to the computer that will participate in the partitioned database environment with the local Administrator account that you have defined for the DB2 installation.
2. Change to the directory where you copied the contents of the DB2 product DVD. For example:

   ```
   cd c:\db2dvd
   ```

   where db2dvd represents the name of the directory where you copied the contents of the DB2 product DVD.
3. From a command prompt, enter the setup command as follows:

   ```
   setup -u responsefile_directory\response_file_name
   ```

   In the following example, the response file, Addpart.file can be found in the c:\responsefile directory. The command for this example, would be:

   ```
   setup -u c:\reponsefile\Addpart.file
   ```
4. Check the messages in the log file when the installation finishes. You can find the log file in the My Documents\DB2LOG\ directory. You should see output similar to the following at the end of the log file:

   ```
   === Logging stopped: 5/9/2007  10:41:32 ===
   MSI (c) (C0:A8) [10:41:32:984]: Product: DB2
   Enterprise Server Edition - DB2COPY1 -- Installation
   operation completed successfully.
   ```

5. When you install the instance-owning database partition server on the primary computer, the DB2 product reserves a port range according to the specified number of logical database partition servers participating in partitioned database environment. The default range is four ports. For each server that participates in the partitioned database environment, you must manually configure the /etc/services file for the FCM ports. The range of the FCM ports depends on how many logical partitions you want to use on the participating computer. A minimum of two entries are required, **DB2_<instance>** and **DB2_<instance>_END**. Other requirements for the FCM ports specified on participating computers are:
   - The starting port number must match the starting port number of the primary computer.
   - Subsequent ports must be sequentially numbered.
   - Specified port numbers must be free.

You must log onto each participating computer and repeat these steps.

If you want your DB2 product to have access to DB2 documentation either on your local computer or on another computer on your network, then you must install the DB2 Information Center. The DB2 Information Center contains documentation for the DB2 database system and DB2 related products.

Installing database partition servers on participating computers using a response file (Linux and UNIX)

In this task you will use the response file you created using the DB2 Setup wizard to install database partition servers on participating computers.

Prerequisites

- You have installed DB2 on the primary computer using the DB2 Setup wizard and have created a response file for installing on participating computers.
- You must have root authority on participating computers.

To install additional database partition servers using a response file:

1. As root, log on to a computer that will participate in the partitioned database environment.
2. Change to the directory where you copied the contents of the DB2 product DVD. For example:
   ```
   cd /db2home/db2dvd
   ```
3. Enter the db2setup command as follows:
   ```
   ./db2setup -r /responsefile_directory/response_file_name
   ```
   In our example, we saved the response file, AddPartitionResponse.file, to the /db2home directory. The command for our example, would be:
   ```
   ./db2setup -r /db2home/AddPartitionResponse.file
   ```
4. Check the messages in the log file when the installation finishes.

You must log onto each participating computer and perform a response file installation.

If you want your DB2 product to have access to DB2 documentation either on your local computer or on another computer on your network, then you must install the

DB2 Information Center. The DB2 Information Center contains documentation for the DB2 database system and DB2 related products.

Chapter 8. After you install

Verifying the installation

Verifying a partitioned database environment installation (Windows)

To verify that your DB2 server installation was successful, you will create a sample database and run SQL commands to retrieve sample data and to verify that the data has been distributed to all participating database partition servers.

You have completed all of the installation steps.

To create the SAMPLE database:

1. Log on to the primary computer (ServerA) as user with SYSADM authority.
2. Enter the db2sampl command to create the SAMPLE database.

   This command may take a few minutes to process. When the command prompt returns, the process is complete.

   The SAMPLE database is automatically cataloged with the database alias SAMPLE when it is created.
3. Start the database manager by entering the db2start command.
4. Enter the following DB2 commands from a DB2 command window to connect to the SAMPLE database, retrieve a list of all the employees that work in department 20:

   ```
   db2 connect to sample
   db2 "select * from staff where dept = 20"
   ```
5. To verify that data has been distributed across database partition servers, enter the following commands from a DB2 command window:

   ```
   db2 "select distinct dbpartitionnum(empno) from employee"
   ```

   The output will list the database partitions used by the employee table. The specific output will depend on the number of database partitions in the database and the number of database partitions in the database partition group that is used by the table space where the employee table was created.

After you have verified the installation, you can remove the SAMPLE database to free up disk space. However, it is useful to keep the sample database, if you plan to make use of the sample applications.

Enter the db2 drop database sample command to drop the SAMPLE database.

Verifying a partitioned database server installation (Linux and UNIX)

To verify that your DB2 server installation was successful, you will create a sample database and run SQL commands to retrieve sample data and to verify that the data has been distributed to all participating database partition servers.

Before following these steps, make sure you have completed all of the installation steps.

© Copyright IBM Corp. 1993, 2009

To create the SAMPLE database:

1. Log on to the primary computer (ServerA) as the instance-owning user. For this example, db2inst1 is the instance-owning user.
2. Enter the db2sampl command to create the SAMPLE database. By default, the sample database will be created in the instance-owner's home directory. In our example /db2home/db2inst1/ is the instance owner's home directory. The instance owner's home directory is the default database path.

   This command may take a few minutes to process. There is no completion message; when the command prompt returns, the process is complete.

   The SAMPLE database is automatically cataloged with the database alias SAMPLE when it is created.
3. Start the database manager by entering the db2start command.
4. Enter the following DB2 commands from a DB2 command window to connect to the SAMPLE database, retrieve a list of all the employees that work in department 20:

   ```
   db2 connect to sample
   db2 "select * from staff where dept = 20"
   ```
5. To verify that data has been distributed across database partition servers, enter the following commands from a DB2 command window:

   ```
   db2 "select distinct dbpartitionnum(empno) from employee"
   ```

   The output will list the database partitions used by the employee table. The specific output will depend on:

   - The number of database partitions in the database
   - The number of database partitions in the database partition group that is used by the table space where the employee table was created

After you have verified the installation, you can remove the SAMPLE database to free up disk space. Enter the db2 drop database sample command to drop the SAMPLE database.

Part 3. Implementation and maintenance

© Copyright IBM Corp. 1993, 2009

Chapter 9. Before creating a database

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.

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.

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 sqldbdir and is located in the sqllib 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 sqldbdir 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.

© Copyright IBM Corp. 1993, 2009

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 **stop_start_time**.

Creating node configuration files

If your database is to operate in a partitioned database environment, you must create a node configuration file called db2nodes.cfg.

This file must be located in the sqllib subdirectory of the home directory for the instance before you can start the database manager with parallel capabilities across multiple database partitions. The file contains configuration information for all database partitions in an instance, and is shared by all database partitions for that instance.

Windows considerations

If you are using DB2 Enterprise Server Edition on Windows, the node configuration file is created for you when you create the instance. You should not attempt to create or modify the node configuration file manually. You can use the db2ncrt command to add a database partition server to an instance. You can use the db2ndrop command to drop a database partition server from an instance. You can use the db2nchg command to modify a database partition server configuration including moving the database partition server from one computer to another; changing the TCP/IP host name; or, selecting a different logical port or network name.

Note: You should not create files or directories under the sqllib subdirectory other than those created by the database manager to prevent the loss of data if an instance is deleted. There are two exceptions. If your system supports stored procedures, put the stored procedure applications in the function subdirectory under the sqllib subdirectory. The other exception is when user-defined functions (UDFs) have been created. UDF executables are allowed in the same directory.

The file contains one line for each database partition that belongs to an instance. Each line has the following format:

```
dbpartitionnum hostname [logical-port [netname]]
```

Tokens are delimited by blanks. The variables are:

dbpartitionnum
The database partition number, which can be from 0 to 999, uniquely defines a database partition. Database partition numbers must be in ascending sequence. You can have gaps in the sequence.

Once a database partition number is assigned, it cannot be changed. (Otherwise the information in the distribution map, which specifies how data is distributed, would be compromised.)

If you drop a database partition, its database partition number can be used again for any new database partition that you add.

The database partition number is used to generate a database partition name in the database directory. It has the format:

```
NODE nnnn
```

The *nnnn* is the database partition number, which is left-padded with zeros. This database partition number is also used by the CREATE DATABASE and DROP DATABASE commands.

hostname
The hostname of the IP address for inter-partition communications. 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.

(There is an exception when netname is specified. In this situation, netname is used for most communications, with hostname only being used for db2start, db2stop, and db2_all.)

logical-port
This parameter is optional, and specifies the logical port number for the database partition. This number is used with the database manager instance name to identify a TCP/IP service name entry in the etc/services file.

The combination of the IP address and the logical port is used as a well-known address, and must be unique among all applications to support communications connections between database partitions.

For each hostname, one *logical-port* must be either 0 (zero) or blank (which defaults to 0). The database partition associated with this *logical-port* is the default node on the host to which clients connect. You can override this with the **DB2NODE** environment variable in db2profile script, or with the sqlesetc() API.

netname
This parameter is optional, and is used to support a host that has more than one active TCP/IP interface, each with its own hostname.

The following example shows a possible node configuration file for an RS/6000 SP system on which SP2EN1 has multiple TCP/IP interfaces, two logical partitions, and uses SP2SW1 as the DB2 database interface. It also shows the database partition numbers starting at 1 (rather than at 0), and a gap in the *dbpartitionnum* sequence:

Table 12. Database partition number example table.

dbpartitionnum	*hostname*	*logical-port*	*netname*
1	SP2EN1.mach1.xxx.com	0	SP2SW1
2	SP2EN1.mach1.xxx.com	1	SP2SW1
4	SP2EN2.mach1.xxx.com	0	
5	SP2EN3.mach1.xxx.com		

You can update the db2nodes.cfg file using an editor of your choice. (The exception is: an editor should not be used on Windows.) You must be careful,

however, to protect the integrity of the information in the file, as database partitioning requires that the node configuration file is locked when you issue db2start and unlocked after db2stop ends the database manager. The db2start command can update the file, if necessary, when the file is locked. For example, you can issue db2start with the **RESTART** option or the **ADD DBPARTITIONNUM** option.

Note: If the db2stop command is not successful and does not unlock the node configuration file, issue db2stop **FORCE** to unlock it.

Format of the DB2 node configuration file

The db2nodes.cfg file is used to define the database partition servers that participate in a DB2 instance. The db2nodes.cfg file is also used to specify the IP address or host name of a high-speed interconnect, if you want to use a high-speed interconnect for database partition server communication.

The format of the db2nodes.cfg file on Linux and UNIX operating systems is as follows:

```
nodenumber hostname logicalport netname resourcesetname
```

nodenumber, *hostname*, *logicalport*, *netname*, and *resourcesetname* are defined in the following section.

The format of the db2nodes.cfg file on Windows operating systems is as follows:

```
nodenumber hostname computername logicalport netname resourcesetname
```

On Windows operating systems, these entries to the db2nodes.cfg are added by the db2ncrt or db2 add db partition commands. You should not add these lines directly or edit this file.

nodenumber
: A unique number, between 0 and 999, that identifies a database partition server in a partitioned database system.

  To scale your partitioned database system, you add an entry for each database partition server to the db2nodes.cfg file. The *nodenumber* value that you select for additional database partition servers must be in ascending order, however, gaps can exist in this sequence. You can choose to put a gap between the *nodenumber* values if you plan to add logical partition servers and wish to keep the nodes logically grouped in this file.

  This entry is required.

hostname
: The TCP/IP host name of the database partition server for use by the FCM. This entry is required.

  If host names are supplied in the db2nodes.cfg file, instead of IP addresses, the database manager will dynamically try to resolve the host names. Resolution can be either local or through lookup at registered Domain Name Servers (DNS), as determined by the OS settings on the machine.

  Starting with DB2 Version 9.1, both TCP/IPv4 and TCP/IPv6 protocols are supported. The method to resolve host names has changed.

  While the method used in pre-Version 9.1 releases resolves the string as defined in the db2nodes.cfg file, the method in Version 9.1 or later tries to resolve the Fully Qualified Domain Names (FQDN) when short names are

defined in the db2nodes.cfg file. Specifying short configured for fully qualified host names, this may lead to unnecessary delays in processes that resolve host names.

To avoid any delays in DB2 commands that require host name resolution, use any of the following work arounds:

1. If short names are specified in the db2nodes.cfg file and the operating system host file, specify the short name and the fully qualified domain name for host name in the operating system host files.
2. To use only IPv4 addresses when you know that the DB2 server listens on an IPv4 port, issue the following command:

   ```
   db2 catalog tcpip4 node db2tcp2 remote 192.0.32.67 server db2inst1
      with "Look up IPv4 address from 192.0.32.67"
   ```

3. To use only IPv6 addresses when you know that the DB2 server listens on an IPv6 port, issue the following command:

   ```
   db2 catalog tcpip6 node db2tcp3 1080:0:0:0:8:800:200C:417A server 50000
      with "Look up IPv6 address from 1080:0:0:0:8:800:200C:417A"
   ```

logicalport
: Specifies the logical port number for the database partition server. This field is used to specify a particular database partition server on a workstation that is running logical database partition servers.

  DB2 reserves a port range (for example, 60000 - 60003) in the /etc/services file for inter-partition communications at the time of installation. This *logicalport* field in db2nodes.cfg specifies which port in that range you want to assign to a particular logical partition server.

  If there is no entry for this field, the default is 0. However, if you add an entry for the *netname* field, you must enter a number for the *logicalport* field.

  If you are using logical database partitions, the *logicalport* value you specify *must* start at 0 and continue in ascending order (for example, 0,1,2).

  Furthermore, if you specify a *logicalport* entry for one database partition server, you must specify a *logicalport* for each database partition server listed in your db2nodes.cfg file.

  This field is optional only if you are *not* using logical database partitions or a high speed interconnect.

netname
: Specifies the host name or the IP address of the high speed interconnect for FCM communication.

  If an entry is specified for this field, all communication between database partition servers (except for communications as a result of the db2start, db2stop, and db2_all commands) is handled through the high speed interconnect.

  This parameter is required only if you are using a high speed interconnect for database partition communications.

resourcesetname
: The *resourcesetname* defines the operating system resource that the node should be started in. The *resourcesetname* is for process affinity support, used for Multiple Logical Nodes (MLNs). This support is provided with a string type field formerly known as quadname.

This parameter is only supported on AIX, HP-UX, and Solaris Operating System.

On AIX, this concept is known as ″resource sets″ and on Solaris Operating System it is called ″projects″. Refer to your operating systems documentation for more information on resource management.

On HP-UX, the *resourcesetname* parameter is the name of a PRM group. Refer to ″HP-UX Process Resource Manager. User Guide. (B8733-90007)″ documentation from HP for more information.

On Windows operating systems, process affinity for a logical node can be defined through the **DB2PROCESSORS** registry variable.

On Linux operating systems, the *resourcesetname* column defines a number that corresponds to a Non-Uniform Memory Access (NUMA) node on the system. The system utility numactl must be available as well as a 2.6 Kernel with NUMA policy support.

The *netname* parameter must be specified if the *resourcesetname* parameter is used.

Example configurations

Use the following example configurations to determine the appropriate configuration for your environment.

One computer, four database partitions servers
If you are not using a clustered environment and want to have four database partition servers on one physical workstation called `ServerA`, update the db2nodes.cfg file as follows:

```
0          ServerA        0
1          ServerA        1
2          ServerA        2
3          ServerA        3
```

Two computers, one database partition server per computer
If you want your partitioned database system to contain two physical workstations, called `ServerA` and `ServerB`, update the db2nodes.cfg file as follows:

```
0          ServerA        0
1          ServerB        0
```

Two computers, three database partition server on one computer
If you want your partitioned database system to contain two physical workstations, called `ServerA` and `ServerB`, and `ServerA` is running 3 database partition servers, update the db2nodes.cfg file as follows:

```
4          ServerA        0
6          ServerA        1
8          ServerA        2
9          ServerB        0
```

Two computers, three database partition servers with high speed switches
If you want your partitioned database system to contain two computers, called `ServerA` and `ServerB` (with `ServerB` running two database partition servers), and use a high speed interconnect called `switch1` and `switch2`, update the db2nodes.cfg file as follows:

```
0          ServerA        0              switch1
1          ServerB        0              switch2
2          ServerB        1              switch2
```

Examples using resourcesetname

These restrictions apply to the following examples:

- This example shows the usage of *resourcesetname* when there is no high speed interconnect in the configuration.
- The *netname* is the fourth column and a *hostname* also can be specified on that column where there is no switch name and you want to use *resourcesetname*. The fifth parameter is *resourcesetname* if it is defined. The resource group specification can only show as the fifth column in the db2nodes.cfg file. This means that for you to specify a resource group, you must also enter a fourth column. The fourth column is intended for a high speed switch.
- If you do not have a high speed switch or you do not want to use it, you must then enter the *hostname* (same as the second column). In other words, the DB2 database management system does not support column gaps (or interchanging them) in the db2nodes.cfg files. This restriction already applies to the first three columns, and now it applies to all five columns.

AIX example

Here is an example of how to set up the resource set for AIX operating systems.

In this example, there is one physical node with 32 processors and 8 logical database partitions (MLNs). This example shows how to provide process affinity to each MLN.

1. Define resource sets in /etc/rset:

```
DB2/MLN1:
    owner     = db2inst1
    group     = system
    perm      = rwr-r-
    resources = sys/cpu.00000,sys/cpu.00001,sys/cpu.00002,sys/cpu.00003

DB2/MLN2:
    owner     = db2inst1
    group     = system
    perm      = rwr-r-
    resources = sys/cpu.00004,sys/cpu.00005,sys/cpu.00006,sys/cpu.00007

DB2/MLN3:
    owner     = db2inst1
    group     = system
    perm      = rwr-r-
    resources = sys/cpu.00008,sys/cpu.00009,sys/cpu.00010,sys/cpu.00011

DB2/MLN4:
    owner     = db2inst1
    group     = system
    perm      = rwr-r-
    resources = sys/cpu.00012,sys/cpu.00013,sys/cpu.00014,sys/cpu.00015

DB2/MLN5:
    owner     = db2inst1
    group     = system
    perm      = rwr-r-
    resources = sys/cpu.00016,sys/cpu.00017,sys/cpu.00018,sys/cpu.00019

DB2/MLN6:
    owner     = db2inst1
    group     = system
    perm      = rwr-r-
    resources = sys/cpu.00020,sys/cpu.00021,sys/cpu.00022,sys/cpu.00023
```

```
DB2/MLN7:
    owner     = db2inst1
    group     = system
    perm      = rwr-r-
    resources = sys/cpu.00024,sys/cpu.00025,sys/cpu.00026,sys/cpu.00027

DB2/MLN8:
    owner     = db2inst1
    group     = system
    perm      = rwr-r-
    resources = sys/cpu.00028,sys/cpu.00029,sys/cpu.00030,sys/cpu.00031
```

2. Enable memory affinity by typing the following command:

```
vmo -p -o memory_affinity=1
```

3. Give instance permissions to use resource sets:

```
chuser capabilities=
    CAP_BYPASS_RAC_VMM,CAP_PROPAGATE,CAP_NUMA_ATTACH  db2inst1
```

4. Add the resource set name as the fifth column in db2nodes.cfg:

```
1 regatta 0 regatta DB2/MLN1
2 regatta 1 regatta DB2/MLN2
3 regatta 2 regatta DB2/MLN3
4 regatta 3 regatta DB2/MLN4
5 regatta 4 regatta DB2/MLN5
6 regatta 5 regatta DB2/MLN6
7 regatta 6 regatta DB2/MLN7
8 regatta 7 regatta DB2/MLN8
```

HP-UX example

This example shows how to use PRM groups for CPU shares on a machine with 4 CPUs and 4 MLNs and 24% of CPU share per MLN, leaving 4% for other applications. The DB2 instance name is db2inst1.

1. Edit GROUP section of /etc/prmconf:

```
 OTHERS:1:4::
db2prm1:50:24::
 db2prm2:51:24::
  db2prm3:52:24::
 db2prm4:53:24::
```

2. Add instance owner entry to /etc/prmconf:

```
db2inst1::::OTHERS,db2prm1,db2prm2,db2prm3,db2prm4
```

3. Initialize groups and enable CPU manager by entering the following command:

```
prmconfig -i
prmconfig -e CPU
```

4. Add PRM group names as a fifth column to db2nodes.cfg:

```
1 voyager 0 voyager db2prm1
2 voyager 1 voyager db2prm2
3 voyager 2 voyager db2prm3
4 voyager 3 voyager db2prm4
```

PRM configuration (steps 1-3) may be done using interactive GUI tool xprm.

Linux example

On Linux operating systems, the *resourcesetname* column defines a number that corresponds to a Non-Uniform Memory Access (NUMA) node on the system. The numactl system utility must be available in addition to a 2.6 kernel with NUMA policy support. Refer to the man page for numact1 for more information about NUMA support on Linux operating systems.

This example shows how to set up a four node NUMA computer with each logical node associated with a NUMA node.

1. Ensure that NUMA capabilities exist on your system.
2. Issue the following command:

   ```
   $ numactl --hardware
   ```

   Output similar to the following displays:

   ```
   available: 4 nodes (0-3)
   node 0 size: 1901 MB
   node 0 free: 1457 MB
   node 1 size: 1910 MB
   node 1 free: 1841 MB
   node 2 size: 1910 MB
   node 2 free: 1851 MB
   node 3 size: 1905 MB
   node 3 free: 1796 MB
   ```

3. In this example, there are four NUMA nodes on the system. Edit the db2nodes.cfg file as follows to associate each MLN with a NUMA node on the system:

   ```
   0 hostname 0 hostname 0
   1 hostname 1 hostname 1
   2 hostname 2 hostname 2
   3 hostname 3 hostname 3
   ```

Solaris example

Here is an example of how to set up the project for Solaris Version 9.

In this example, there is 1 physical node with 8 processors: one CPU will be used for the default project, three (3) CPUs will used by the Application Server, and four (4) CPUs for DB2. The instance name is db2inst1.

1. Create a resource pool configuration file using an editor. For this example, the file will be called pool.db2. Here's the content:

   ```
   create system hostname
   create pset pset_default (uint pset.min = 1)
   create pset db0_pset (uint pset.min = 1; uint pset.max = 1)
   create pset db1_pset (uint pset.min = 1; uint pset.max = 1)
   create pset db2_pset (uint pset.min = 1; uint pset.max = 1)
   create pset db3_pset (uint pset.min = 1; uint pset.max = 1)
   create pset appsrv_pset (uint pset.min = 3; uint pset.max = 3)
   create pool pool_default (string pool.scheduler="TS";
        boolean pool.default = true)
   create pool db0_pool (string pool.scheduler="TS")
   create pool db1_pool (string pool.scheduler="TS")
   create pool db2_pool (string pool.scheduler="TS")
   create pool db3_pool (string pool.scheduler="TS")
   create pool appsrv_pool (string pool.scheduler="TS")
   associate pool pool_default (pset pset_default)
   associate pool db0_pool (pset db0_pset)
   associate pool db1_pool (pset db1_pset)
   associate pool db2_pool (pset db2_pset)
   associate pool db3_pool (pset db3_pset)
   associate pool appsrv_pool (pset appsrv_pset)
   ```

2. Edit the /etc/project file to add the DB2 projects and appsrv project as follows:

   ```
   system:0::::
   user.root:1::::
   noproject:2::::
   default:3::::
   group.staff:10::::
   appsrv:4000:App Serv project:root::project.pool=appsrv_pool
   ```

```
db2proj0:5000:DB2 Node 0 project:db2inst1,root::project.pool=db0_pool
db2proj1:5001:DB2 Node 1 project:db2inst1,root::project.pool=db1_pool
db2proj2:5002:DB2 Node 2 project:db2inst1,root::project.pool=db2_pool
db2proj3:5003:DB2 Node 3 project:db2inst1,root::project.pool=db3_pool
```

3. Create the resource pool: `# poolcfg -f pool.db2`.
4. Activate the resource pool: `# pooladm -c`
5. Add the project name as the fifth column in db2nodes.cfg file:

```
0 hostname 0 hostname db2proj0
1 hostname 1 hostname db2proj1
2 hostname 2 hostname db2proj2
3 hostname 3 hostname db2proj3
```

Specifying the list of machines in a partitioned database environment

By default, the list of computers is taken from the node configuration file, db2nodes.cfg.

You can override this by:

- Specifying a pathname to the file that contains the list of computers by exporting (on Linux and UNIX platforms) or setting (on Windows) the environment variable RAHOSTFILE.
- Specifying the list explicitly, as a string of names separated by spaces, by exporting (on Linux and UNIX platforms) or setting (on Windows) the environment variable RAHOSTLIST.

  Note: If both of these environment variables are specified, RAHOSTLIST takes precedence.

Note: On Windows, to avoid introducing inconsistencies into the node configuration file, do *not* edit it manually. To obtain the list of computers in the instance, use the db2nlist command.

Eliminating duplicate entries from a list of machines in a partitioned database environment

If you are running DB2 Enterprise Server Edition with multiple logical database partition servers on one computer, your db2nodes.cfg file will contain multiple entries for that computer.

In this situation, the rah command needs to know whether you want the command to be executed once only 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 platforms, if you specify computers, rah will normally eliminate 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 will be 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 only specify 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 db2 restart database command. You will need to know the database partition number of the catalog partition to do this.

If you execute db2 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.

Updating the node configuration file (Linux and UNIX)

This task provides steps for updating the db2nodes.cfg file to include entries for participating computers.

The node configuration file (db2nodes.cfg), located in the instance owner's home directory, contains configuration information that tells DB2 which servers participate in an instance of the partitioned database environment. A db2nodes.cfg file exists for each instance in a partitioned database environment.

The db2nodes.cfg file must contain one entry for each server participating in the instance. When you create an instance, the db2nodes.cfg file is automatically created and an entry for the instance-owning server is added.

For example, when you created the DB2 instance using the DB2 Setup wizard, on the instance-owning server `ServerA`, the db2nodes.cfg file is updated as follows:

```
0     ServerA     0
```

Prerequistes

- The DB2 application must be installed on all participating computers.
- A DB2 instance must exist on the primary computer.
- You must be a user with SYSADM authority.
- Review the configuration examples and file format information provided in the Format of the DB2 node configuration file topic if either of the following conditions apply:
  - You plan to use a high speed switch for communication between database partition servers
  - Your partitioned configuration will have multiple logical partitions

Restriction

The hostnames used in the steps of the Procedure section must be fully qualified hostnames.

To update the db2nodes.cfg file:

1. Log on as the instance owner (in our example, db2inst1 is the instance owner).
2. Ensure that the DB2 instance is stopped by entering:

   INSTHOME`/sqllib/adm/db2stop`

   where *INSTHOME* is the home directory of the instance owner (the db2nodes.cfg file is locked when the instance is running and can only be edited when the instance is stopped).

For example, if your instance home directory is /db2home/db2inst1, enter the following command:

```
/db2home/db2inst1/sqllib/adm/db2stop
```

3. Add an entry to the .rhosts file for each DB2 instance. Update the file by adding the following:

   ```
   <hostname> <db2instance>
   ```

   where <hostname> is the TCP/IP host name of the database server and <db2instance> is the name of the instance you use to access the database server.
4. Add an entry to the db2nodes.cfg file of each participating server. When you first view the db2nodes.cfg file, it should contain an entry similar to the following:

   ```
   0     ServerA     0
   ```

   This entry includes the database partition server number (node number), the TCP/IP host name of the server where the database partition server resides, and a logical port number for the database partition server.

   For example, if you are installing a partitioned configuration with four computers and a database partition server on each computer, the updated db2nodes.cfg should appear similar to the following:

   ```
   0     ServerA     0
   1     ServerB     0
   2     ServerC     0
   3     ServerD     0
   ```
5. When you have finished updating the db2nodes.cfg file, enter the INSTHOME/sqllib/adm/db2start command, where *INSTHOME* is the home directory of the instance owner. For example, if your instance home directory is /db2home/db2inst1, enter the following command:

   ```
   /db2home/db2inst1/sqllib/adm/db2start
   ```
6. Log out.

Setting up multiple logical nodes

Typically, you configure DB2 Enterprise Server Edition to have one database partition server assigned to each computer. There are several situations, however, in which it would be 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* or *multiple logical nodes* 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 DB2START NODENUM command. This ensures that user data remains available.

Another benefit is that multiple logical partitions can exploit 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.

- 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 (nodes) 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 partition (node) 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
```

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.

Note: 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.

Enabling parallelism for 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.

Enabling parallelism when creating indexes

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.

Enabling I/O parallelism when backing up a database or table space

To enable I/O parallelism when backing up a database or table space:

1. Use more than one target media.
2. 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.
3. Use the PARALLELISM parameter on the BACKUP command to specify the degree of parallelism.
4. Use the WITH `num-buffers` BUFFERS parameter on the BACKUP command to ensure 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 (extentsize * number of containers) product of the table spaces being backed up.

Enabling I/O parallelism when restoring a database or table space

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 make the decision 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 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 (extentsize * number of containers) product of the table spaces being restored.
  – The same as, or an even multiple of, the backup buffer size.

Enabling intra-partition parallelism for queries

You could also 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 for a specific database or in the database manager configuration file, use the UPDATE DATABASE CONFIGURATION and the UPDATE DATABASE MANAGER CONFIGURATION commands respectively.

Configuration parameters that affect intra-partition parallelism include the *max_querydegree* and *intra_parallel* database manager parameters, and the *dft_degree* database parameter.

In order for intra-partition query parallelism to occur, you must modify one or more database configuration parameters, database manager configuration parameters, precompile or bind options, or a special register.

intra_parallel
: Database manager configuration parameter that specifies whether the database manager can use intra-partition parallelism. The default is not to use intra-partition parallelism.

max_querydegree
: Database manager configuration parameter that specifies the maximum degree of intra-partition parallelism that is used for any SQL statement running on this instance. An SQL statement will not use more than the number given by this parameter when running parallel operations within a database partition. The *intra_parallel* configuration parameter must also be set to "YES" for the value in *max_querydegree* is used. The default value for this configuration parameter is -1. This value means that the system uses the degree of parallelism determined by the optimizer; otherwise, the user-specified value is used.

dft_degree
: Database configuration parameter that provides the default for the DEGREE bind option and the CURRENT DEGREE special register. The default value is 1. A value of ANY means the system uses the degree of parallelism determined by the optimizer.

DEGREE
Precompile or bind option for static SQL.

CURRENT DEGREE
Special register for dynamic SQL.

Management of data server capacity

If data server capacity does not meet your present or future needs, you can expand its capacity by adding disk space and creating additional containers, or by adding memory. If these simple strategies do not add the capacity you need, also consider adding processors or physical partitions. When you scale your system by changing the environment, you should be aware of the impact that such a change can have on your database procedures such as loading data, or backing up and restoring databases.

Adding processors

If a single-partition database configuration with a single processor is used to its maximum capacity, you might either add processors or add database partitions. The advantage of adding processors is greater processing power. In an SMP system, processors share memory and storage system resources. All of the processors are in one system, so there are no additional overhead considerations such as communication between systems and coordination of tasks between systems. Utilities such as load, backup, and restore can take advantage of the additional processors.

Note: Some operating systems, such as the Solaris operating system, can dynamically turn processors on- and off-line.

If you add processors, review and modify some database configuration parameters that determine the number of processors used. The following database configuration parameters determine the number of processors used and might need to be updated:

- Default degree (dft_degree)
- Maximum degree of parallelism (max_querydegree)
- Enable intra-partition parallelism (intra_parallel)

You should also evaluate parameters that determine how applications perform parallel processing.

In an environment where TCP/IP is used for communication, review the value for the DB2TCPCONNMGRS registry variable.

Adding physical partitions

If your database manager is currently in a partitioned database environment, you can increase both data-storage space and processing power by adding separate single-processor or multiple-processor physical partitions. The memory and storage system resources on each database partition are not shared with the other database partitions. Although adding database partitions might result in communication and task-coordination issues, this choice provides the advantage of balancing data and user access across more than one system. The database manager supports this environment.

You can add database partitions either while the database manager system is running or while it is stopped. If you add database partitions while the system is running, however, you must stop and restart the system before databases migrate to the new database partition.

When you add a new database partition, you cannot drop or create a database that takes advantage of the new database partition until the procedure is complete, and the new server is successfully integrated into the system.

Fast communications manager

Fast communications manager (Windows)

The fast communications manager (FCM) provides communications support for DB2 server products belonging to the same instance. Each database partition server has one FCM sender, and one FCM receiver daemon to provide communications between database partition servers to handle agent requests and to deliver message buffers. The FCM daemon is started when you start the instance.

If communications fail between database partition servers or if they re-establish communications, the FCM thread updates information (that you can query with the database system monitor) and causes the appropriate action (such as the rollback of an affected transaction) to be performed. You can use the database system monitor to help you set the FCM configuration parameters.

You can specify the number of FCM message buffers with the *fcm_num_buffers* database manager configuration parameter and the number of FCM channels with the *fcm_num_channels* database manager configuration parameter. The *fcm_num_buffers* and *fcm_num_channels* database manager configuration parameter are set to AUTOMATIC as the default value. FCM monitors resource usage when any of these parameter are set to automatic, and incrementally releases resources. It is recommended to leave these parameters set to AUTOMATIC.

Fast communications manager (Linux and UNIX)

The fast communications manager (FCM) provides communications support for DB2 server products that use the Database Partitioning Feature (DPF).

For multiple partition instances, each database partition server has one FCM sender daemon and one FCM receiver daemon to provide communications between database partition servers to handle agent requests and to deliver message buffers. The FCM daemon is started when you start the multiple partition instance.

If communications fail between database partition servers or if they re-establish communications, the FCM daemons update information. You can query this information with the database system monitor. The FCM daemons also trigger the appropriate action. An example of an appropriate action is the rollback of an affected transaction. You can use the database system monitor to help you set the FCM configuration parameters.

You can specify the number of FCM message buffers with the *fcm_num_buffers* database manager configuration parameter. You can also specify the number of FCM channels with the *fcm_num_channels* database manager configuration parameter. The *fcm_num_buffers* and *fcm_num_channels* database manager configuration parameters are set to AUTOMATIC as the default value. The FCM monitors resource usage when any of these parameter are set to automatic, and incrementally releases resources. It is recommended to leave these parameters set to AUTOMATIC.

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 below. 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 below.

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

If you are using DB2 Enterprise Server Edition in the Windows environment, 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 would 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 you are allocating.

The first line specifies the first port, while the second line indicates the end of the block of ports. In the following example, five ports are allocated for the instance `sales`. 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. Also you must ensure that you include both underscore (_) characters.

Enabling communications between database partition servers (Linux and UNIX)

This task describes how to enable communication between the database partition servers that participate in your partitioned database system. Communication between database partition servers is handled by the Fast Communications Manager (FCM). To enable FCM, a port or port range must be reserved in the /etc/services file on each computer in your partitioned database system.

You must have a user ID with root authority.

You must perform this task on all computers that participate in the instance.

The number of ports to reserve for FCM is equal to the maximum number of database partitions hosted, or potentially hosted, by any computer in the instance.

In the following example, the db2nodes.cfg file contains these entries:

```
0 server1 0
1 server1 1
2 server2 0
3 server2 1
4 server2 2
5 server3 0
6 server3 1
7 server3 2
8 server3 3
```

Assume that the FCM ports are numbered starting at 60000. In this situation:

- server1 uses two ports (60000, 60001) for its two database partitions
- server2 uses three ports (60000, 60001, 60002) for its three database partitions
- server3 uses four ports (60000, 60001, 60002, 60003) for its four database partitions

All computers must reserve 60000, 60001, 60002, and 60003, since this is the largest port range required by any computer in the instance.

If you use a high availability solution such as High Availability Cluster Multi-Processing (HACMP™) or Tivoli System Automation (TSA) to fail over database partitions from one computer to another, you must account for potential port requirements. For example, if a computer normally hosts four database partitions, but another computer's two database partitions could potentially fail over to it, six ports must be planned for that computer.

When you create an instance, a port range is reserved on the primary computer. The primary computer is also known as the instance-owning computer. However,

if the port range originally added to the /etc/services file is not sufficient for your needs, you will need to extend the range of reserved ports by manually adding additional entries.

To enable communications between servers in a partitioned database environment using /etc/services:

1. Log on to the primary computer (instance owning computer) as a user with root authority.
2. Create an instance.
3. View the default port range that has been reserved in the /etc/services file. In addition to the base configuration, the FCM ports should appear similar to the following:

   ```
   db2c_db2inst1         50000/tcp
   #Add FCM port information
   DB2_db2inst1          60000/tcp
   DB2_db2inst1_1        60001/tcp
   DB2_db2inst1_2        60002/tcp
   DB2_db2inst1_END      60003/tcp
   ```

   By default, the first port (50000) is reserved for connection requests, and the first available four ports above 60000 are reserved for FCM communication. One port is for the instance-owning database partition server and three ports are for logical database partition servers that you might choose to add to the computer after installation is complete.

   The port range must include a start and an END entry. Intermediate entries are optional. Explicitly including intermediate values can be useful for preventing other applications from using these ports, but these entries are not verified by the database manager.

   DB2 port entries use the following format:

   ```
   DB2_instance_name_suffix  port_number/tcp  # comment
   ```

   where:

   - *instance_name* is the name of the partitioned instance.
   - *suffix* is not used for the first FCM port. Intermediate entries are those between the lowest and highest port. If you include the intermediate entries between the first and ending FCM port, the *suffix* consists of an integer that you increment by one for each additional port. For example, the second port is numbered 1, and third is numbered 2, and so on to ensure uniqueness. The word END must be used as the *suffix* for the last entry.
   - *port_number* is the port number that you reserve for database partition server communications.
   - *comment* is an optional comment describing an entry.
4. Ensure that there are sufficient ports reserved for FCM communication. If the range of reserved ports is insufficient, add new entries to the file.
5. Log on as a root user to each computer participating in the instance and add identical entries to the /etc/services file.

Chapter 10. Creating and managing partitioned database environments

Initial database partition groups

When a database is initially created, database partitions are created for all database partitions specified in the db2nodes.cfg file. Other database partitions can be added or removed with the ADD DBPARTITIONNUM and DROP DBPARTITIONNUM VERIFY commands.

Three database partition groups are defined:

- 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, by default holding user tables and indexes.

Creating database partition groups

You create a database partition group with 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.

The computers and systems must be available and capable of handling a partitioned database environment. You have purchased and installed DB2 Enterprise Server Edition. The database must exist.

This statement also:

- Creates a distribution map for the database partition group.
- Generates a distribution map ID.
- Inserts records into the following catalog tables:
  - SYSCAT.DBPARTITIONGROUPS
  - SYSCAT.PARTITIONMAPS
  - SYSCAT.DBPARTITIONGROUPDEF

To create a database partition group using the Control Center:

1. Expand the object tree until you see the **Database partition groups** folder.
2. Right-click the **Database partition groups** folder, and select **Create** from the pop-up menu.
3. On the Create Database partition groups window, complete the information, use the arrows to move nodes from the Available nodes box to the Selected database partitions box, and click **OK**.

To create a database partition group using the command line, enter:

```
CREATE DATABASE PARTITION GROUP db-partition-group-name
   ON DBPARTITIONNUM (db-partition-number1,db-partition-number1)
```

or

© Copyright IBM Corp. 1993, 2009

```
CREATE DATABASE PARTITION GROUP db-partition-group-name
   ON DBPARTITIONNUMS (db-partition-number1
   TO db-partition-number2)
```

For example, assume that you want to load some tables on a subset of the database partitions in your database. You would use the following command to create a database partition group of two database partitions (1 and 2) in a database consisting of at least three (0 to 2) database partitions:

```
CREATE DATABASE PARTITION GROUP mixng12 ON DBPARTITIONNUM (1,2)
```

or

```
CREATE DATABASE PARTITION GROUP mixng12 ON DBPARTITIONNUMS (1 TO 2)
```

The CREATE DATABASE command or sqlecrea() API also create the default system database partition groups, IBMDEFAULTGROUP, IBMCATGROUP, and IBMTEMPGROUP.

Table spaces in database partition groups

By placing a table space in a multiple-partition database partition group, all of the tables within the table space are divided or partitioned across each database partition in the database partition group.

The table space is created into a database partition group. Once in a database partition group, the table space must remain there; it cannot be changed to another database partition group. The CREATE TABLESPACE statement is used to associate a table space with a database partition group.

Managing database partitions

You can use the Partitions view in the Control Center to start and stop partitions, drop and trace partitions, and to display the diagnostics log.

To work with database partitions or view the DB2 logs, you will need authority to attach to an instance. Anyone with SYSADM or DBADM authority can grant you with the authority to access a specific instance.

If requested to do so from IBM Support, run the trace utility using the options that they indicate. The trace utility records information about DB2 operations and formats this information into readable form. For more information, see "db2trc - Trace: DB2" topic.

Attention: Only use the trace facility when directed by DB2 Customer Service or by a technical support representative to do so.

Use the Diagnostic Log window to view text information logged by the DB2 trace utility.

The Partitions view displays the following information:

Node Number
This column contains icons and node numbers. The node numbers are unique numbers, and can be from 0 to 999. The numbers are stored in the db2nodes.cfg file. Node numbers are displayed in ascending sequence, though there might be gaps in the sequence.

Node numbers, once assigned, cannot be changed. This safeguard ensures that the information in the distribution map (which details how data is partitioned) is not compromised.

Host Name
The host name is the IP address used by fast communication manager (FCM) for internal communications. (However, if a switch name is specified, FCM uses the switch name. In this situation, the host name is used only for DB2START, DB2STOP, and db2_all.) The host name is stored in the db2nodes.cfg file.

Port Number
The port number is the logical port number for the node. This number is used with the database manager instance name to identify a TCP/IP service name entry in the /etc/services file. This number is stored in the db2nodes.cfg file.

The combination of the IP address (host name) and the logical port is used as a well-known address, and must be unique among all applications to support communication connections between nodes.

For each displayed host name, one port number will be 0. Port number 0 indicates the default node on the host to which clients connect. (To override this behavior, use the **DB2NODE** environment variable in db2profile script.)

Switch Name
The switch name (or netname) is used to support a host that has more than one active TCP/IP interface, each with its own host name. It is also used for fast communications (also known as FCM) between nodes that share a high-speed switch. The switch name is stored in the db2nodes.cfg file. If the switch name is not specified in the db2nodes.cfg file, it is the same as the host name.

Resourcesetname
The resourcesetname defines the operating system resource that the node should be started in. The resourcesetname is for process affinity support, used for multiple logical notes (MLNs), provided with a string type field formerly known as quadname.

1. Open the Partitions view: From the Control Center, expand the object tree until you find the instance for which you want to view the partitions. Right-click on the instance you want and select **Open → Partitions** from the pop-up menu. The Partitions view opens.
2. To start partitions: Highlight one or more partitions and select **Partitions → Start**. The selected partitions are started.
3. To stop partitions: Highlight one or more partitions and select **Partitions → Stop**. The selected partitions are stopped.
4. To run the trace utility on a partition:
   a. Open the DB2 Trace window: Highlight a partition, and select **Partitions → Service → Trace**. The DB2 Trace window opens.
   b. Specify the trace options.
   c. Click **Start** to start recording information, **Stop** to stop recording information, and **Save as** to save the information to a file.
   d. Optional: View the logs.
   e. Send the trace output to IBM Support, if requested to do so.

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 may 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 `db2start`
- With the command-line processor ADD DBPARTITIONNUM command
- With the API function `sqleaddn`
- With the API function `sqlepstart`

If your system is stopped, you use `db2start`. 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.

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 simply 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 Windows and have no databases in the instance, use the DB2NCRT command to scale the database system. If, however, you already have databases, use the DB2START ADDNODE command to ensure that a database partition is created for each existing database when you scale the system. On Windows, you should never manually edit the node configuration file (db2nodes.cfg), as this can introduce inconsistencies into the file.

Adding a database partition to a running database system

You can add new database partitions to a partitioned database environment while it is running and while applications are connected to databases. However, a newly added server does not become available to all databases until the database manager is shut down and restarted.

To add a database partition to a running database manager using the command line:

1. On any existing database partition, run the DB2START 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 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.

   When the DB2START command is complete, the new server is stopped.
2. Stop the database manager on all database partitions by running the DB2STOP command.

   When you stop all the database partitions in the system, the node configuration file is updated to include the new database partition. The node configuration file is not updated with the new server information until DB2STOP is executed. This ensures that the ADD DBPARTITIONNUM command, which is called when you specify the ADDNODE parameter to the DB2START command, runs on the correct database partition. When the utility ends, the new server partition is stopped.
3. Start the database manager by running the DB2START command.

   The newly added database partition is now started along 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 DB2START command twice for all database partition servers to access the new `db2nodes.cfg` file.
4. Optional: Back up all databases on the new database partition.
5. Optional: Redistribute data to the new database partition.

Adding a database partition to a stopped database system (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 up again.

You must install the new server before you can create a database partition on it.

To add a database partition to a stopped partitioned database server using the command line:

1. Issue DB2STOP 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 DB2START command to start the database system. Note that the node configuration file (cfg) has already been updated 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 DB2START command.

      Specify the new database partition values for DBPARTITIONNUM, ADDDBPARTITIONNUM, 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.

      When the DB2START command is complete, the new server is stopped.
   b. Stop the entire database manager by running the DB2STOP command.

      When you stop all the database partitions in the system, the node configuration file is updated to include the new database partition. The node configuration file is not updated with the new server information until DB2STOP is executed. This ensures that the ADD DBPARTITIONNUM command, which is called when you specify the ADDDBPARTITIONNUM parameter to the DB2START command, runs on the correct database partition. When the utility ends, the new server partition is stopped.
5. Start the database manager by running the DB2START 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 DB2START command twice for all database partition servers to access the new `db2nodes.cfg` file.
6. Optional: Back up all databases on the new database partition.
7. Optional: Redistribute data to the new database partition.

Adding a database partition to a stopped database system (UNIX)

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 up again.

You must install the new server if it does not exist. , including the following tasks:

- Making executables accessible (using shared file-system mounts or local copies)
- Synchronizing operating system files with those on existing processors
- Ensuring that the `sqllib` directory is accessible as a shared file system
- Ensuring that the relevant operating system parameters (such as the maximum number of processes) are set to the appropriate values

You must also register the host name with the name server or in the hosts file in the etc directory on all database partitions.

To add a database partition to a stopped partitioned database server using the command line:

1. Issue DB2STOP 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 DB2START command to start the database system. Note that the node configuration file (cfg) has already been updated 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 DB2START command.

      Specify the new database partition values for DBPARTITIONNUM, ADDDBPARTITIONNUM, 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.

      When the DB2START command is complete, the new server is stopped.
   b. Stop the entire database manager by running the DB2STOP command.

      When you stop all the database partitions in the system, the node configuration file is updated to include the new database partition. The node configuration file is not updated with the new server information until DB2STOP is executed. This ensures that the ADD DBPARTITIONNUM command, which is called when you specify the ADDDBPARTITIONNUM parameter to the DB2START command, runs on the correct database partition. When the utility ends, the new server partition is stopped.
5. Start the database manager by running the DB2START 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 DB2START command twice for all database partition servers to access the new `db2nodes.cfg` file.
6. Optional: Back up all databases on the new database partition.
7. Optional: Redistribute data to the new database partition.

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: `DB2START DBPARTITIONNUM partitionnum`

   Specify the number you are assigning to the new database partition as the value of nodenum.
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 NODE command on the new database partition.

This command creates a database partition 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.

5. When the ADD DBPARTITIONNUM command completes, issue the DB2START 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.

Error recovery when adding database partitions

Adding database partitions does not fail as a result of non-existent 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 the system 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 db2start command to add a database partition to the current multi-partition database:

     ```
     DB2START 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 NODE or the db2start command. After doing this, you need to use the CREATE BUFFERPOOL statement to create the buffer pools using , 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:

    ```
    DB2START
    CONNECT TO mpp1
    ALTER DATABASE PARTITION GROUP ng1 ADD NODE (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:

```
DB2START
CONNECT TO mpp1
CREATE BUFFERPOOL bp1 SIZE 1000 PAGESIZE 8192
CONNECT RESET
CONNECT TO mpp1
ALTER DATABASE PARTITION GROUP ng1 ADD NODE (2)
```

Note: If the database partition group has table spaces with the default page size, message SQL1759W is returned.

Dropping database partitions

You can drop a database partition that is not being used by any database and free the computer for other uses.

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 NODEGROUP 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 in-doubt transactions.

To drop a database partition using the command line, issue the DB2STOP command with the DROP NODENUM parameter to drop the database partition. After the command completes successfully, the system is stopped. Then start the database manager with the DB2START command.

Listing database partition servers in an instance

On Windows, use the db2nlist command to obtain a list of database partition servers that participate in an instance.

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.

Adding database partition servers to an instance (Windows)

On Windows, use the db2ncrt command to add a database partition server to an instance.

Note: Do not use the db2ncrt command if the instance already contains databases. Instead, use the db2start addnode 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:node_number
        /u:username,password
        /p:logical_port
```

/n: The unique database partition number to identify the database partition server. The number can be from 1 to 999 in ascending sequence.

/u: 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
```

Adding database partitions to an instance using the Add Partitions wizard

Use the Add Partitions wizard to create a partition and add it to one or more database partition groups. First you add a new partition to your instance and assign the partition to one or more database partition groups, then you make more advanced choices.

To work with database partition groups, you must have SYSADM or DBADM authority.

1. Open the Add Partitions wizard:
   a. From the Control Center, expand the object tree until you find the instance object that you want to work with. Right-click the object, and click Add Partitions in the pop-up menu. The Add Partitions launchpad opens.
   b. Click the **Add Partitions** button. The Add Partitions wizard opens.
2. Complete each of the applicable wizard pages. Click the wizard overview link on the first page for more information. The **Finish** push button is available when you complete enough information for the wizard to add the partition.

Changing database partitions (Windows)

On Windows, use the db2nchg command to change database partitions.

- 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's configuration 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 hostname used by FCM for internal communications. If you do not specify this parameter, the hostname is unchanged.

/m:computer_name
Moves the database partition server to another computer. The database partition server can only be moved 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 will access 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 should 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.

Adding containers to SMS table spaces on database partitions

You can only add a container to a SMS table space on a database partition that currently has no containers.

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.

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 based operating system:

```
ALTER TABLESPACE plans
   ADD ('/dev/rhdisk0')
   ON DBPARTITIONNUM (3)
```

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.

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 will become 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 db2stop drop nodenum 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:node_number /i:instance_name
```

/n: The unique database partition number 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. This is an optional parameter. If not given, the default is the current instance (set by the DB2INSTANCE registry variable).

Dropping database partitions from the instance using the Drop Partitions launchpad

Use the Drop Partitions launchpad to guide you through the tasks necessary to drop database partitions from database partition groups, redistribute data in database partition groups, and drop partitions from an instance.

About this task

Note: When you drop database partitions from database partition groups the database partitions are not immediately dropped. Instead, the database partitions that you want to drop are flagged so that data can be move off them when you redistribute the data in the database partition groups.

It is recommended that you backup all databases in the instance before and after redistributing data in database partition groups. If you do not back up your databases, you might corrupt the databases and you might not be able to recover them.

Procedure

To drop partitions using the Drop Partitions launchpad:

1. Optional: Back up data using the Backup wizard.
2. Open the Drop Partitions launchpad:
   a. Open the Partitions window: From the Control Center, expand the object tree until you find the instance for which you want to view the partitions. Right-click the instance that you want and select **Open Database Partition Servers** from the pop-up menu. The Partitions window opens for the selected instance.
   b. Select the partitions you want to drop.
   c. Right-click the selected partitions and click **Drop** in the pop-up menu. The Drop Partitions launchpad opens.
3. Drop the database partitions from database partition groups:
   a. Confirm the database partitions you want to drop from the database partition groups.

      Note:

      - You must drop the database partitions from database partition groups before you drop partitions from the instance.
      - This operation does not drop the database partitions immediately. Instead, it flags the database partitions that you want to drop so that data can be move off them when you redistribute the data in the database partition group.
4. .Redistribute data in a database partition group.
5. Drop partitions from the instance:
   a. Open the Drop Partitions from Instance Confirmation window:
      - Open the Partitions window as described above.
      - Select the partitions you want to drop.

- Right-click the selected partitions and click **Drop** in the pop-up menu. The Drop Partitions launchpad opens.
- Click the **Drop Partitions from Instance** push button. The Drop Partitions from Instance Confirmation window opens.

b. In the **Drop** column, verify that you want to drop the partitions for the selected instance.

c. Click **OK** to open a window where you can schedule when you want to drop the partition.

6. Optional: Back up data using the Backup wizard.

Scenario: Partitioning data in a database

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.

Scenario:

A database DBPG1 has 2 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

Distribution of data between the database partitions in DBPG1:

To add three new database partitions to the database, issue the following commands:

```
DB2START DBPARTITIONNUM 3 ADD DBPARTITIONNUM HOSTNAME <HOSTNAME3>
 PORT <PORT3>;
DB2START DBPARTITIONNUM 4 ADD DBPARTITIONNUM HOSTNAME <HOSTNAME4>
 PORT <PORT4>;
DB2START DBPARTITIONNUM 5 ADD DBPARTITIONNUM HOSTNAME <HOSTNAME5>
 PORT <PORT5>;
DB2STOP;
DB2START;
```

The following redistribute command will change the DBPG_1 definition from (0, 1) to (0, 1, 3, 4, 5) and redistribute the data as well:

```
DB2 REDISTRIBUTE DATABASE PARTITION GROUP DBPG_1 NOT ROLLFORWARD RECOVERABLE
 UNIFORM ADD DBPARTITIONNUM (3 TO 5) 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:

```
DB2 REDISTRIBUTE DATABASE PARTITION GROUP DBPG_1 NOT ROLLFORWARD
 RECOVERABLE ABORT;
```

You might abort the data redistribution if an error or an interruption occurred during the data redistribution and you do not wish 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:

```
DB2 REDISTRIBUTE DATABASE PARTITION GROUP DBPG_1 NOT ROLLFORWARD RECOVERABLE
 UNIFORM ADD DBPARTITIONNUM (3 TO 5) TABLE (T5, T4) ONLY  DATA BUFFER 5000;
```

If the command ran successfully, the data in tables T4 and T5 will have been redistributed successfully.

To complete the redistribution of data on table T1, T2, and T3 in a specified order, issue:

```
DB2 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 option and the DROP DBPARTITIONNUM option do not need to be specified. Instead, the ALTER DATABASE PARTITION GROUP statement may be used to add or drop database partitions prior to the REDISTRIBUTE DATABASE PARTITION GROUP command being executed, in which case the REDISTRIBUTE DATABASE PARTITION GROUP command will not add or drop partitions, but will simply redistribute the data according to the specified options.
- It is strongly recommended that the user take an offline database backup of the database prior to executing the REDISTRIBUTE DATABASE PARTITION GROUP command. This action is not shown in the examples above.
- The REDISTRIBUTE DATABASE PARTITION GROUP command is not rollforward recoverable. For a full discussion of this issue, refer to the “REDISTRIBUTE DATABASE PARTITION GROUP command”.
- 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 will be accessible for write activity.

The steps above illustrate how you can use variations of the redistribute command to redistribute data between database partitions.

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 (nodes). You can do so using the rah command or the db2_all 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_all command. This section provides an overview of these commands. The information that follows applies to partitioned database environments only.

Note:

1. On Linux and UNIX platforms, 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.
2. Also, on Linux and UNIX platforms, 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). The ssh remote shell program is used to prevent the transmission of passwords in clear text in UNIX operating system environments. If this registry variable is not set, rsh (or remsh for HP-UX) is used.
3. On Windows, to run the rah command or the db2_all command, you must be logged on with a user account that is a member of the Administrators group.

To determine the scope of a command, refer to the *Command Reference*, which indicates whether a command runs on a single database partition server, or on all of them. If the command runs on one database partition server and you want it to run on all of them, use db2_all. The exception is the db2trc command, which runs on all the logical nodes (database partition servers) on a computer. If you want to run db2trc on all logical nodes on all computers, use rah.

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 platforms, 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_all command, type:

```
db2_all command
```

To obtain help about rah syntax, type

```
rah "?"
```

The command can be almost anything which you could type at an interactive prompt, including, for example, multiple commands to be run in sequence. On Linux and UNIX platforms, 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_all command to change the database configuration on all database partitions that are specified in the node configuration file. Because the ; character is placed inside double quotation marks, the request will run concurrently:

```
db2_all ";DB2 GET DB CFG FOR sample USING LOGFILSIZ 100"
```

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 don't specify any parameter.

You should 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 platforms, the command will be 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 need to include a \ in your command, you must type two backslashes (\\).

Note: On Linux and UNIX platforms, 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 need to include one of these characters in your command, you must precede them by three backslashes (\\\). For example, if you need to include a \ in your command, you must type four backslashes (\\\\).

If you need to include a double quotation mark (") in your command, you must precede it by three backslashes, for example, \\\".

Note:

1. On Linux and UNIX platforms, 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 which contains logic to read from stdin in the background, you should 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, you should 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 platforms 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 $RAHBUFDIR/$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 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 already 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.

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 nodes 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 nodes.

At those nodes, 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 nodes on the list. The threshold can be specified by environment variable RAHTREETHRESH, or defaults to 15.

In the case of a multiple-logical-node-per-physical-node system, db2_all will favor sending the recursive invocation to distinct physical nodes, which will then rsh to other logical nodes on the same physical node, thus also reducing inter-physical-node traffic. (This point applies only to db2_all, not rah, since rah always sends only to distinct physical nodes.)

rah and db2_all commands

This topic includes descriptions of the rah and db2_all commands.

Command
Description

rah Runs the command on all computers.

db2_all
Runs the command on all database partition servers that you specify.

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 service.

db2_call_stack
On Linux and UNIX platforms, causes all processes running on all database partition servers to write call traceback to the syslog.

On Linux and UNIX platforms, 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.

rah 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 platforms:

```
rah "};ps -F pid,ppid,etime,args -u $USER"
```

- On Windows:

```
rah "||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 platforms) or background processes (on Windows) 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 platforms, 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 for the `|&` case above.

Note: On Linux and UNIX platforms, specifying `&` degrades performance, because more rsh commands are required.

`;` Same as `||&` above. This is an alternative shorter form.

Note: On Linux and UNIX platforms, 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 platforms only.

`}` Prepends dot-execution of file named in $RAHENV (probably .kshrc) before executing command.

Note: Available on Linux and UNIX platforms 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 platforms only.

`)` Suppresses execution of user's profile and of file named in $RAHENV.

Note: Available on Linux and UNIX platforms 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 node number *nnn*, see the first paragraph following the last prefix sequence in this table).

<<+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).

(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 will never wait 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.

Note:

1. 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 node 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 node configuration file. The computer index for MACH3, however, would be 3.

   On Windows, do not edit the node configuration file. To obtain the computer index, use the db2nlist command.
2. When " is specified, duplicates are not eliminated from the list of computers.

When using the <<-nnn< and <<+nnn< prefix sequences, *nnn* is any 1-, 2- or 3-digit database partition number which must match the *nodenum* value in the db2nodes.cfg file.

Note: 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 13. Environment variables that control the rah command

Name	Meaning	Default
$RAHBUFDIR **Note:** Available on Linux and UNIX platforms only.	Directory for buffer	/tmp/$USER
$RAHBUFNAME **Note:** Available on Linux and UNIX platforms only.	Filename for buffer	rahout
$RAHOSTFILE (on Linux and UNIX platforms); RAHOSTFILE (on Windows)	File containing list of hosts	db2nodes.cfg
$RAHOSTLIST (on Linux and UNIX platforms); RAHOSTLIST (on Windows)	List of hosts as a string	extracted from $RAHOSTFILE

Table 13. Environment variables that control the rah command (continued)

Name	Meaning	Default
$RAHCHECKBUF **Note:** Available on Linux and UNIX platforms only.	If set to ″no″, bypass checks	not set
$RAHSLEEPTIME (on Linux and UNIX platforms); RAHSLEEPTIME (on Windows)	Time in seconds this script will wait for initial output from commands run in parallel	86400 seconds for db2_kill, 200 seconds for all others
$RAHWAITTIME (on Linux and UNIX platforms); RAHWAITTIME (on Windows)	On Windows, interval in seconds between successive checks that remote jobs are still running. On Linux and UNIX platforms, interval in seconds between successive checks that remote jobs are still running and `rah: waiting for pid> ...` messages. On all platforms, specify any positive integer. Prefix value with a leading zero to suppress messages, for example, export RAHWAITTIME=045. It is not necessary to specify a low value as rah does not rely on these checks to detect job completion.	45 seconds
$RAHENV **Note:** Available on Linux and UNIX platforms only.	Specifies filename to be executed if $RAHDOTFILES=E or K or PE or B	$ENV
$RAHUSER (on Linux and UNIX platforms); RAHUSER (on Windows)	On Linux and UNIX platforms, user ID under which the remote command is to be run. On Windows, the logon account associated with the DB2 Remote Command Service	$USER

Note: On Linux and UNIX platforms, 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 platforms 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 will be 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 essential that your .cshrc does 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
```

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 platforms 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 .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 have a 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 somewher -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 will 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 $RAHBUFNAME 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, $RAHBUFDIR/$RAHBUFNAME) for buffering the outputs. To prevent this problem, use a different $RAHBUFNAME for each concurrent rah command, for example in the following ksh:

   ```
   export RAHBUFNAME=rahout
   rah ";$command_1" &
   export RAHBUFNAME=rah2out
   rah ";$command_2" &
   ```

   or use a method that makes the shell choose a unique name automatically such as:

   ```
   RAHBUFNAME=rahout.$$ db2_all ";....."
   ```

   Whatever method you use, you must ensure 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 will leave 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.

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.

Note: The information in this section applies to Linux and UNIX platforms only.

The informative messages are written at an interval controlled by the environment variable `RAHWAITTIME`. Refer to the help information for details on how specify this. All informative messages can be completely 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 will 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 will not affect the remote commands at all, but will prevent 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 will be 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 and UNIX environment, 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 makes the decision as to whether the remote ID's shell 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.

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.

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.

Chapter 11. Creating tables and other related table objects

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 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.

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 2 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:

© Copyright IBM Corp. 1993, 2009

```
CREATE TABLE name>
  (<column_name> <data_type> <null_attribute>)
  IN <tagle_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,
                     MIX_FLT 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.

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. For SMS table spaces, the long data must reside in the same table space as the data object it belongs to. 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 SMS), 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.

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.

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
- SYSADM or DBADM authority

You can create a partitioned table by using the CREATE TABLE statement.

To create a partitioned table from the command line, issue the CREATE TABLE statement:

```
CREATE TABLE <NAME> (<column_name>  <data_type>  <null_attribute>) IN
 <table space list> PARTITION BY RANGE (<column expression>)
 STARTING FROM <constant> ENDING <constant> EVERY <constant>
```

For example, the following statement creates a table where rows with a ≥ 1 and a ≤ 20 are in PART0 (the first data partition), rows with 21 ≤ a ≤ 40 are in PART1 (the second data partition), up to 81 ≤ a ≤ 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.

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 NULL clause of the CREATE TABLE statement specifies 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 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.

Migrating existing tables and views to partitioned tables

To migrate data from a Version 9.1 table into an empty partitioned table, use the LOAD command.

There are three approaches you can use to migrate an existing table or view to a partitioned table:

- When migrating regular tables create a new, empty partitioned table and use the LOAD from CURSOR to move the data from the old table directly into the partitioned table without any intermediate steps.
- When migrating regular tables unload the source table using the export utility or high performance unload, create a new, empty partitioned table and use the LOAD command to populate an empty partitioned table.
- When migrating Union All views create a partitioned table with a single dummy data partition, then attach all of the tables.

Example 1: Suppose you have a regular table t1:

```
CREATE TABLE t1 (c1 int, c2 int);
```

Create a new, empty partitioned table:

```
CREATE TABLE sales_dp (c1 int, c2 int)
   PARTITION BY RANGE (c1)
   (STARTING FROM 0 ENDING AT 10 EVERY 2);
```

Populate table t1:

```
INSERT INTO t1 VALUES (0,1), (4, 2), (6, 3);
```

To avoid creating a third copy of the data in a flat file, issue the LOAD 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;
SELECT * FROM sales_dp;
```

Drop the old table:

```
DROP TABLE t1;
```

Converting UNION ALL views

You can convert Version 9.1 data in a UNION ALL view into a partitioned table. 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. Table partitioning accomplishes all of these and is easier to administer. Using the ALTER TABLE ...ATTACH operation, you can achieve conversion with

no movement of data in the base table. Indexes and dependent views or materialized query tables (MQT's) must be re-created after the conversion.

The recommended strategy 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.

Example 2:

Create table syntax for first table in the UNION:

```
CREATE TABLE sales_0198(
 sales_date DATE NOT NULL,
 prod_id INTEGER,
 city_id INTEGER,
 channel_id INTEGER,
 revenue DECIMAL(20,2),
  CONSTRAINT ck_date
  CHECK
  (sales_date BETWEEN '01-01-1998' AND '01-31-1998'));
```

Create view syntax for a 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'
 );
```

Create a partitioned table with a single dummy partition. The range should be chosen 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');
```

Attach the first table:

```
ALTER TABLE sales_dp ATTACH PARTITION
STARTING FROM '01-01-1998' ENDING AT '01-31-1998'
FROM sales_0198;
```

Drop the dummy partition:

```
ALTER TABLE sales_dp DETACH PARTITION dummy
INTO dummy;
 DROP TABLE dummy;
```

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;
```

Issue the SET INTEGRITY statement to bring the attached data partitions online.

```
SET INTEGRITY FOR sales_dp IMMEDIATE CHECKED
FOR EXCEPTION IN sales_dp USE sales_ex;
```

Create indexes, as appropriate.

Conversion considerations

Attaching a data partition is allowed unless the value of the SYSCAT.COLUMNS IMPLICITVALUE field in a specific column is a non-null 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 non-null value in the SYSCAT.COLUMNS IMPLICITVALUE field if one of the following conditions are met:

- the column is created as the result of an ALTER TABLE ...ADD COLUMN statement
- the IMPLICITVALUE field is propagated from a source table during attach
- the IMPLICITVALUE field is inherited from a source table during detach
- 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. 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.

To avoid these inconsistencies, it is recommended that you 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 a target table of an attach operation.

For best practices in avoiding a mismatch when working with partitioned tables, see “Guidelines for attaching data partitions to partitioned tables” on page 196.

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 dependents:

- If you issue a DETACH PARTITION operation and there are any dependent tables that need to be incrementally maintained with respect to the detached data partition (these dependents table are referred to as detached dependent tables), then the newly detached table is initially inaccessible. The 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 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 detached table is transitioned to a regular table where it becomes fully accessible.

- 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 dependents 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 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 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.

Creating range-clustered tables

Algorithms used for range-clustered tables

An algorithm is used to equate the value of the key for the record with the location of a specific row within a table. The basic algorithm is fairly simple. In its most basic form (using a single column instead of two or more columns to make up the key), the algorithm maps a sequence number to a logical row number.

The algorithm also uses the record's key to determine the logical page number and slot number. This process provides exceptionally fast access to records; that is, to specific rows in the table.

The algorithm does not involve hashing because hashing does not preserve key-value ordering. Preserving key-value ordering is essential because it eliminates the need to reorganize the table data over time.

Each record key in the table should have the following characteristics:

- Unique
- Not null
- An integer (SMALLINT, INTEGER, or BIGINT)
- Monotonically increasing
- Within a predetermined set of ranges based on each column in the key

  The ALLOW OVERFLOW option is used when creating the table to allow key values to exceed the defined range. The DISALLOW OVERFLOW option is used when creating the table where key values will not exceed the defined range. In this case, if a record is inserted out of the boundary indicated by the range, an SQL error message is returned.

Applications where tightly clustered (dense) sequence key ranges are likely are excellent candidates for range-clustered tables. When using this type of key to create a range-clustered table, the key is used to generate the logical location of a row in a table. This process avoids the need for a separate index.

Range-clustered table indexes

In range-clustered tables, indexes locate a record based on a key from the record, apply start and stop scans, and distribute the data vertically. By using an RCT, the only property of an index that is not accounted for is vertical distribution of data.

Differences from regular tables

When deciding to use range-clustered tables, there are some characteristics to consider, which differentiate them from regular tables.

- Range-clustered tables have no free-space control records (FSCR).
- Space is preallocated.

  Space for the table is preallocated and reserved for use by the table even when records for the table are not filled in. At table creation time, there are no records in the table; however, the entire range of pages is preallocated. Preallocation is based on the record size and the maximum number of records to be stored.

  – If variable length fields such as VARCHAR are used in each record, the maximum length of the field is used and the overall record size is a fixed length. The overall fixed length of each record is used with the maximum number of records to determine the space required.
  – This can result in additional space being allocated that cannot be effectively utilized.
  – If key values are sparse, there is unused space and poor range scan performance.
  – Range scans must visit all possible records within a range even if the rows containing those key values have not yet been inserted into the database.
- No schema modifications permitted.

  If a schema modification is required on a range-clustered table, the table must be recreated to include the new schema name for the table and all the data from the old table. In particular:

  – Altering a key range is not supported.

    This is important since if a table's ranges need to be altered, a new table with the desired ranges must be created and the new table populated with the data from the old table.
- Duplicate key values are not allowed.
- Key values outside the defined range are not allowed.

This is true for range-clustered tables defined to DISALLOW OVERFLOW only.

  – NULL values are explicitly disallowed.

- Range-cluster index is not materialized

  An index with RCT key properties is indicated in the system catalogs and can be selected by the optimizer, but the index is not materialized on disk. With a regular table, space also needs to be given to each index associated with a table. With a RCT, no space is required for the RCT index. The optimizer uses the information in the system catalogs that refers to this RCT index to ensure that the correct access method for the table can be chosen.
- Creating a primary or a unique key on the same definition as the range-clustered table index is not permitted since it would be redundant.
- Range-clustered tables retain the original key value ordering, a feature that guarantees the clustering of rows within a table.

Guidelines for using range-clustered tables

When working with range-clustered tables (RCT), there are some guidelines to follow.

- When defining the range of key values, the minimum value is optional; if it is not specified, then the default is one (1). Negative values are allowed for minimum and maximum values. When working with negative values, the minimum value must be stated explicitly. For example, ORGANIZE BY KEY SEQUENCE (F1 STARTING FROM -100 ENDING AT® -10)
- Creating a regular index on the same key values used to define the range-clustered table is not allowed.
- Some ALTER TABLE options are unavailable for use on range-clustered tables. Where the option does not affect the physical structure of the table, the option is allowed.
- Because the process of creating a range-clustered table preallocates the required disk space, that space must be available or the table creation will fail.

How the SQL compiler works with range-clustered tables

The SQL compiler handles the range-clustered table (RCT) in a similar way to a regular table that has a secondary B+ tree index. Rather than working through a B+ tree index to determine the record's location or Record Identifier (RID), RCT uses a functional lookup involving the record key values and the algorithm from the range definition. This is similar to having an index because a key value is used to obtain the RID quickly.

When working to determine the best access path to required data, the SQL compiler uses statistical information kept about the tables. Index statistics are collected during a table scan when a RUNSTATS command is issued. For an RCT, the table is modeled as a regular table, and the index is modeled as a function-based index.

Order of records in the table is not guaranteed when creating a range-clustered table allowing overflow.

Scenarios: Range-clustered tables

These scenarios are simple and demonstrate the ways to create a range-clustered table. They show how you can use either a single column, or multiple columns, as the key to the table. In addition, they show how to create a table that allows data to overflow and a table that does not allow data to overflow.

Scenario 1: Creating a range-clustered table

This scenario shows a range-clustered table that is used to locate a student using a STUDENT_ID. For each student record, the following information is included:

- School ID
- Program ID
- Student number
- Student ID
- Student first name
- Student last name
- Student grade point average (GPA)

In this case, the student records are based solely on the STUDENT_ID. The STUDENT_ID will be used to add, update, and delete student records.

Note: Other indexes can be added separately at another time. However, for the purpose of this example, the organization of the table and how to access the table's data are defined when the table is created.

Here is the syntax needed for this table:

```
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
;
```

The size of each record is the sum of the columns. In this case, there is a 10 byte header + 4 + 4 + 4 + 4 + 30 + 30 + 8 + 3 (for nullable columns) equaling 97 bytes. With a 4 KB page size (or 4096 bytes), after accounting for the overhead there is 4038 bytes, or enough room for 42 records per page. If 1 million student records are allowed, there will be a need for 1 million divided by 42 records per page, or 23809.5 pages. This rounds up to 23810 pages that are needed. Four pages are added for table overhead and three pages for extent mapping. The result is a required preallocation of 23817 pages of 4 KB size. (The extent mapping assumes a single container to hold this table. There should be three pages for each container.)

Scenario 2: Creating a range-clustered table (overflow not allowed)

In this scenario, which is a variation on the first, consider the idea of a school board. In the school board there are 200 schools, each having 20 classrooms with a capacity of 35 students. This school board can accommodate a maximum of 140,000 students.

In this case, the student records are based on three factors: the SCHOOL_ID, the CLASS_ID, and the STUDENT_NUM values. Each of these three columns will have unique values and will be used together to add, update, and delete student records.

Note: As with the previous example, other indexes might be added separately and at some other time.

Here is the syntax needed for this table:

```
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 case, an overflow is not allowed. This makes sense because there is likely a school board policy that restricts the number of students allowed in each class. In this scenario, the largest possible class size is 35. When you couple this factor with the physical limitations imposed by the number of classrooms and schools, it is clear that there is no reason to allow an overflow in the number of students in the school board.

It is possible that schools have varying numbers of classrooms. If this is the case, when defining the range for the number of classrooms (using CLASS_ID), the upper boundary should be the largest number of classrooms when considering all of the schools. This might mean that some smaller schools (schools with fewer classrooms than the largest school) will have space for student records that might never be used (unless, for example, portable classrooms are added to the school).

By using the same 4 KB page size and the same student record size as in the previous example, there can be 42 records per page. With 140,000 student records, there will be a need for 3333.3 pages, or 3334 pages once rounding up is done. There are two pages for table information, and three pages for extent mapping. The result is a required preallocation of 3339 pages of 4 KB size.

Considerations when creating MDC tables

There are many factors that should be considered when creating MDC tables. The following sections discuss how your decisions on how to create, place, and use your MDC tables could be influenced by your current database environment (for example, whether you have a partitioned database or not), and by your choices of dimensions for your MDC table. Also discussed is the DB2 Design Advisor, and how it can be used to provide advice on some of these issues.

Moving data from existing tables to MDC tables

To improve query performance and reduce the overhead 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: export your data, drop the original table (optional), create a multidimensional clustering (MDC) table (using the CREATE TABLE statement with the ORGANIZE BY DIMENSIONS clause), and 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 does the following when altering a table:

- Drop all dependent objects of the table
- Rename the table
- Create the table using the new definition
- Recreate all dependent objects of the table
- 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 may be used to transform from a old data type to a new data type.

MDC tables in SMS table spaces

If you plan to store MDC tables in an SMS table space, you need to use multipage file allocation. (Multipage file allocation is the default for newly created databases in Version 8.2 and later.) The reason for this is that MDC tables are always extended by whole extents, and it is important that all the pages in these extents are physically consecutive. Therefore, there is no space advantage to disabling multipage file allocation; and furthermore, enabling it will significantly increase the chances that the pages in each extent are physically consecutive.

MDC Advisor feature on the DB2 Design Advisor

The DB2 Design Advisor (db2advis), formerly known as the Index Advisor, 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 mathematics expression to reduce the cardinality (the number of distinct values) of a clustering dimension. A common example of a coarsification is the date where coarsification could be 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 will 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 will be created in the same table space as the existing table, and will therefore have the same extent size. The recommendations for MDC dimensions would change depending on the extent size of the table space since the extent size impacts the number of records that can fit into a block or cell. This directly affects the density of the cells.

Only single-column dimensions, and not composite-column dimensions, are considered, although single or multiple dimensions may be recommended for the

table. The MDC feature will recommend 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 VARGRAPH 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 impact 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 impact of any table expansion resulting from the organization of data along particular MDC dimensions.

The MDC feature is enabled 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 will not explore tables that are less than 12 extents in size.

The advisor will analyze 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 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 conjunction with 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 nodes in order to:

- Take advantage of multiple machines to increase processing requests in parallel.
- Increase the physical size of the table beyond a single database partition's limits.
- 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 will contain 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 will be used to divide the data. As a result, the Red and Blue slices may 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 will have a subset of data from each slice. When choosing dimensions and estimating cell occupancy (see the section called "Density of cells"), 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 will be heavily used in queries, you can cluster the table on the columns involved, 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 native 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 will be processed by the relational operators involved. Note that the size of a block (the number of pages) will be 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 will be 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 will have the same number of blocks as the table in Example 1, but one less dimension block index. In Example 1, there will be three dimension block indexes, one for each of the columns c1, c3, and c4. In Example 2, there will be two dimension block indexes, one on the column c1 and the other on the columns c3 and c4. The main differences between these two approaches is that, in Example 1, queries involving just 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 just c4 involve more processing. However, in Example 2 there will be 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. In order to implement the rolling up of dimensions in this way, you can use generated columns. This type of column definition will allow 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, while column c6 rolls up column c1 to a coarser granularity in time. This statement will cluster 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 will be 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 we create an MDC table with dimensions on the generated column month, where `month = INTEGER (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 will only allow 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 usual way, with a starting key of 4 and a stop key of 6.

To make this function monotonic, you would have to include the year as the high order part of the month. Version 9.2 provides 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 DATENUM( ), DAYS( ), 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 will only support the use of equality predicates on its base column.

Chapter 12. Altering a database

Altering an instance

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.

Consistency is required since the SQL compiler compiles distributed SQL statements based on information in the node 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.

Altering a database

Altering a database partition group

To alter a database partition group using the command line processor: use the REDISTRIBUTE DATABASE PARTITION GROUP command. Once you add or drop database partitions, you must redistribute the current data across the new set of database partitions in the database partition group.

Managing database partitions from the Control Center

You can work with database partitions using the Database Partitions view of the Control Center.

To work with database partitions you will need authority to attach to an instance. Anyone with SYSADM or DBADM authority can grant you with the authority to access a specific instance.

To configure a database partition or take a database partition out of the rollforward pending state you must have SYSADM, SYSCTRL, or SYSMAINT authority.

Using the Database Partitions view you can restart a database partition, take a database partition out of the rollforward pending state, backup a database partition, restore a database partition, or configure a database partition using the Configuration Advisor.

To open the Database Partitions view from the Control Center:

1. From the Control Center, expand the object tree until you find the partitioned database for which you want to view the database partitions.
2. Right-click the partitioned database you want and select Open Database Partitions from menu list.
3. The Database Partitions view opens for the selected partitioned database.

To configure a database partition:

© Copyright IBM Corp. 1993, 2009

1. Select the database partitions that you want from the Database Partitions view.
2. Select Database Partitions, right-click and select Configuration Advisor from the list.
3. The Configuration Advisor opens. Use the Configuration Advisor to specify values for the database configuration parameters.

Chapter 13. Altering tables and other related table objects

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.

To alter a partitioned table to detach a data partition the user must have the following authorities or privileges:

- The user performing the DETACH operation must have the authority needed to ALTER, to SELECT from, and to DELETE from the source table.
- The user must also have the authority needed 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:
  - SYSADM or 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:

- SELECT privilege on the source table and DROPIN privilege on the schema of the source table
- CONTROL privilege on the source table
- SYSADM or DBADM authority.

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
- SYSADM
- DBADM
- ALTERIN privilege on the table schema

Usage guidelines

- 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.

© Copyright IBM Corp. 1993, 2009

- Multiple ALTER statements can be executed, followed by a single SET INTEGRITY statement.

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.

Adding or altering a check or foreign key constraint
Adding a check on a foreign key constraint is supported with attached and detached data partitions.

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, the column is not added to the detached data partitions, because the detached data partitions are no longer physically associated to the table.

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.

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 an 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 dependents or staging tables that need to be incrementally refreshed with respect to the data partition. In this case, the index includes the data for this detached data partition.

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

- ACTIVATE NOT LOGGED INITIALLY (WITH EMPTY TABLE)

Attaching 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. Rolling in partitioned table data allows you to easily incorporate a new range into a partitioned table as an additional data partition.

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 and privileges on the source table:

- SELECT privilege on the source table and DROPIN privilege on the schema of the source table
- CONTROL privilege on the source table
- SYSADM or DBADM authority.

The ATTACH PARTITION clause takes an existing table (source table) and attaches it as a new data partition to the target table. The newly attached data partition is initially inaccessible to queries. The remainder of the table remains online. A call to the SET INTEGRITY statement is required to bring the attached data partition online.

Restrictions and usage guidelines

The following conditions must be met before you can attach a data partition:

- The table to which you want to attach the new data partition (that is, the target table) must be an existing partitioned table.
- The source table must be an existing non-partitioned table or a partitioned table with only a single data partition, and with no ATTACHED or DETACHED data partitions. To attach multiple data partitions, it is necessary to issue multiple ATTACH statements.
- The source table cannot be hierarchical (typed table).
- The source table cannot be a range-clustered table (RCT).
- The table definition for a source table must match the target table.
- The number, type, and ordering of columns must match for the source and target tables.
- For both tables, columns must match in terms of whether they contain default values. If the source column is created by using the ALTER TABLE ADD COLUMN, that is, SYSCOLUMNS.ADD_DEFAULT = 'Y', the existDefault value (SYSCOLUMNS.ADDED_DEFAULT) must match that of the target column.
- For both tables, columns must match in terms of whether they allow NULL or not.
- The Compression clause, including both VALUE COMPRESSION and SYSTEM COMPRESSION DEFAULT values must match for the source and target tables.
- Use of the APPEND clause with data capture option, and the not logged initially option must match.
- Attaching a data partition is allowed even when the target column is a generated column and the source column is not a generated column. This statement SET INTEGRITY FOR T ALLOW WRITE ACCESS IMMEDIATE CHECKED FORCE GENERATED generates the values for the generated column

of the attached rows. The column matching a generated column must match in type and nullability. There are no required default values for this column. The recommended approach is to guarantee that the source table of the ATTACH has the correct generated value in the generated column. Then, you are not required to use the FORCE GENERATED option. The following statement can be used:

```
SET INTEGRITY FOR T GENERATED COLUMN IMMEDIATE UNCHECKED
(indicates to the system to bypass checking of generated column)
SET INTEGRITY FOR T ALLOW WRITE ACCESS IMMEDIATE CHECKED FOR EXCEPTION
IN T USE T_EX (performs integrity checking of the attached partition but
will not check for generated column correctness)
```

- Attaching a data partition is allowed even when the target column is identity and the source column is non-identity. 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 matching 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 droppable (that is, it cannot have RESTRICT DROP set).
- If a DATAPARTITIONNAME is specified, it must not already exist in the target table.
- If the target table is an multidimensional clustering (MDC) table, the source table must also be an MDC 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), pageSize, extentSize and database partition group. A warning is returned to the user if the prefetchSize do not match. The large table space for the source table must match the large table spaces for the target table in type, database partition group and pageSize.

To use the DB2 command line to alter a partitioned table and to attach a data partition to the table, issue the ALTER TABLE statement.

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 a table like an existing table*.

To help you prevent a mismatch from occuring, refer to the Restrictions and usage guidelines section of "Attaching data partitions" on page 195. 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
```

Include any necessary SET INTEGRITY statements. If the source table has indexes, you might not be able to attach the source table immediately. Detached data partitions remain detached until all indexes are cleaned-up of detached keys. If you want to perform an attach immediately, drop the index on the source table. Otherwise, create a new source.

The source table is a system table, a view, a typed table, a table ORGANIZED BY KEY SEQUENCE or a declared global 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)
```

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
```

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.

To detach a data partition from a partitioned table you must have the following authorities or privileges:

- The user performing the DETACH operation must have the authority needed to ALTER, to SELECT from and to DELETE from the source table.
- The user must also have the authority needed 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:
  - SYSADM or 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 DBADM or SYSADM have access to the data immediately after the ALTER TABLE ...DETACH PARTITION statement.

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.

Restrictions

If the source table is a multidimensional clustered table (MDC), access to the newly detached table is not allowed in the same unit of work as the ALTER TABLE ...DETACH operation. Block indexes are created upon first access to the table after the ALTER TABLE ...DETACH operation is committed. Access time is reduced while the block indexes are created.

You must meet the following conditions before you can perform a DETACH 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 operation (target table) must not exist.
- DETACH is not allowed on a table that is the parent of an enforced referential integrity (RI) relationship.
- If there are any dependent tables that need to be incrementally maintained with respect to the detached data partition (these dependents table are referred to as detached dependent tables), then the newly detached table is initially inaccessible. The table will be marked with an L in the TYPE column of the SYSCAT.TABLES catalog view. This is referred to as a detached table. This prevents the 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 detached table is transitioned to a regular table where it becomes fully accessible.

To alter a partitioned table and to detach a data partition from the table, from the command line, issue the ALTER TABLE statement with the DETACH PARTITION clause.

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 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.

Note: If there are detached dependents then the detached data partition does not become a stand-alone table at detach time. In this case, the SET INTEGRITY statement must be issued to complete the detach and make the table accessible.

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
  - 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 table (MDC), the target table is also an MDC table, defined with the same dimension columns. Access to the newly detached table is not allowed in the same unit of work as the detach when the source table is MDC.
- 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.
- 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.
- Index table space (INDEX_TBSPACE column of the SYSCAT.TABLES catalog view). The value is set to NULL. Indexes for the table resulting from the DETACH will be in the same table space as the table.
- Triggers
- Primary key
- Statistics
- All other attributes not mentioned in the list of attributes explicitly inherited from the source table.

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. Adding a data partition to a partitioned table is more appropriate than attaching a data partition in situations where 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.

Restrictions and usage guidelines

- You cannot add a data partition to a non-partitioned table. For details on migrating an existing table to a partitioned table, see “Migrating existing tables and views to partitioned tables” on page 175.
- 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 manufactures 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 no IN or LONG IN clause is specified for ADD PARTITION, the table space in which to place the data partition is chosen 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.

Omitting the STARTING or ENDING bound for an ADD or ATTACH operation is also used to fill a gap in range values. Here is an example of filling in a gap 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.
```

To alter a partitioned table and to add a new data partition to the table, from the command line, issue the ALTER TABLE statement with the ADD PARTITION clause.

Example 1: Add a data partition to an existing partitioned table holding a range of values 901 to 1000 inclusive. Assume table sales holds nine ranges 0-100, 101-200, and so on, up to the value of 900. The example adds an additional 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)
```

Dropping data partitions

You can drop a data partition using the ALTER TABLE statement with the DETACH PARTITION clause. However, you can use the DROP TABLE statement to drop the separate table.

To detach a data partition from a partitioned table the user must have the following authorities or privileges:

- The user performing the DETACH must have the authority needed to ALTER, to SELECT from and to DELETE from the source table.
- The user must also have the authority needed 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 able:
  - SYSADM or 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:
  - SYSADM or 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 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 DBADM or SYSADM have access to the data immediately after the ALTER TABLE ... DETACH PARTITION operation.

To detach a data partition of a partitioned table, from the command line, issue the ALTER TABLE statement with the DETACH PARTITION clause.

You can drop a table from the DB2 command line processor (CLP).

To drop a table, from the command line, issue the DROP TABLE statement. In the following example, the dec01 data partition is detached from table stock and placed in table junk. You can then drop table junk, effectively dropping the associated data partition.

```
ALTER TABLE stock DETACH PART dec01 INTO junk;
DROP TABLE junk;
```

Note: To make the DETACH operation as fast as possible, index cleanup on the source table is done automatically using a background asynchronous index cleanup process. If there are detached dependents then the detached data partition does not become a stand-alone table at detach time. In this case, the SET INTEGRITY statement must be issued to complete the detach and make the table accessible.

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 the table then adding new data. Table partitioning functionality allows you to detach the data partition with the obsolete data then attach a new data partition with the latest data.

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 needed to ALTER, to SELECT from, and to DELETE from the source table.
- The user must also have the authority needed 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:
  - SYSADM or 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 table to attach a data partition, the user must have the following authorities or privileges:

- The user performing the attach must have the authority needed to ALTER and to INSERT into the target table
- The user must also be able to SELECT from and to DROP the source table. Therefore, 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 on the source table:
  - SELECT privilege on the source table and DROPIN privilege on the schema of the source table
  - CONTROL privilege on the source table
  - SYSADM or DBADM authority.

To rotate data in a partitioned table, from the command line, issue the ALTER TABLE statement. The following example demonstrates how to update the stock table by removing the data from December 2001 and replacing it with the latest data from December 2003.

1. Remove the old data from table *stock*.

   ```
   ALTER TABLE stock DETACH PARTITION dec01 INTO newtable;
   ```

2. Load the new data. Using LOAD with the REPLACE option overwrites existing data.

   ```
   LOAD FROM data_file OF DEL REPLACE INTO newtable
   ```

   Note: If there are detached dependents, then you must run the SET INTEGRITY statement on the detached dependents before you can load the detached table.

3. If desired, perform data cleansing. Data cleansing activities include:
   - 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 pre-aggregated during roll-in.)

4. Attach the new data as a new range.

```
ALTER TABLE stock ATTACH PARTITION dec03
STARTING '12/01/2003' ENDING '12/31/2003'
 FROM newtable;
```

   Attaching a data partition drains queries and invalidates packages.

5. Use the SET INTEGRITY statement to update indexes and other dependent objects. Read and write access is permitted during the 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

These scenarios address a common administration operation in data warehouses where new data is rolled in at the start of each month and old data is potentially rolled out based on a particular date.

Scenario 1 covers the DETACH operation (roll-out) by removing obsolete data from the table. Variations include deleting the data, and moving the data into another table. The scenarios cover both an ADD operation and an ATTACH operation (roll-in) by loading new data into the table. Variations include:

1. Transforming the data, loading it into the non-partitioned table, then attaching the data partition (traditional extract, transform and load (ETL)).
2. Loading the data into the non-partitioned table, transforming the data
3. Attaching the data partition.

Scenario 1: Using partitioned tables, the roll-out operation is simply a DETACH operation on the appropriate data partition

```
ALTER TABLE stock DETACH PART dec01 INTO stock_drop;
COMMIT WORK
```

To accelerate the DETACH operation, index cleanup on the source table is done automatically through a background asynchronous index cleanup process. If there are no detached dependents defined on the source table, there is no need to issue the SET INTEGRITY statement to complete the DETACH operation.

Instead of dropping the table, it also possible to attach the table to another table, or truncate it and use it as a table to load new data into before reattaching it. You can perform these operations immediately, even before the asynchronous index cleanup has completed, except where the stock table has detached dependents.

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 dependents 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.

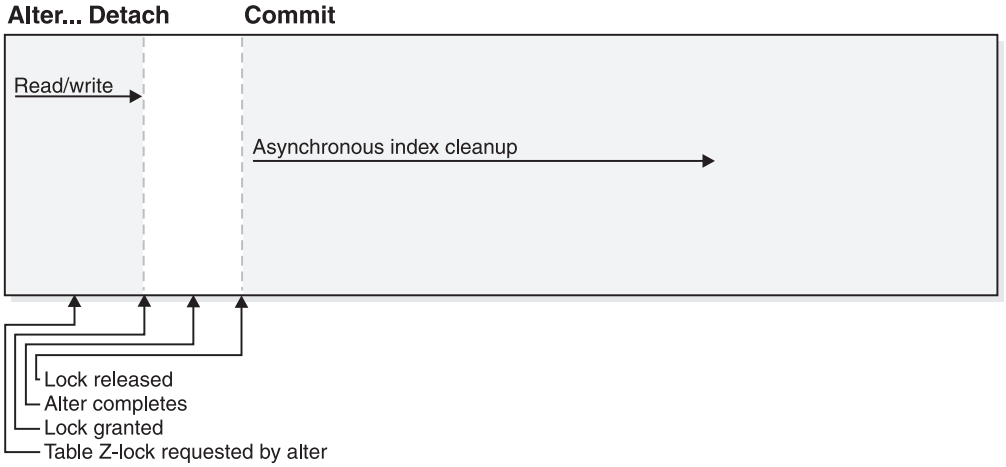


Figure 37. This figure demonstrates the stages of data availability during a DETACH operation. Asynchronous index cleanup commences immediately after the DETACH operation is committed if there are no detached dependents. Otherwise, asynchronous index cleanup commences after the maintenance of the detached dependents is committed.

Scenario 2: Creating a new, empty range

The following scenario illustrates the steps to load data into a non-partitioned table and then add that data partition to the rest of the table.

```
ALTER TABLE stock ADD PARTITION dec03
STARTING FROM '12/01/2003' ENDING AT '12/31/2003';
```

This ALTER TABLE ... ADD operation drains queries running against the stock table and invalidate packages. That is, existing queries complete normally before the ADD operation continues. Once the ADD operation is issued, any new queries accessing the stock table block on a lock.

Load data into the table:

```
LOAD FROM data_file OF DEL INSERT
INTO stock ALLOW READ ACCESS;
```

Use the SET INTEGRITY statement to validate constraints and refresh dependent materialized query tables (MQTs):

```
SET INTEGRITY FOR stock ALLOW READ
ACCESS IMMEDIATE CHECKED FOR EXCEPTION IN stock USE stock_ex;

COMMIT WORK;
```

Tip: One advantage for using ALTER TABLE ... ADD PARTITION followed by a LOAD operation versus a LOAD operation followed by ALTER TABLE ... ATTACH is if the table has no constraints or MQTs defined, the SET INTEGRITY statement is not required to make the new data available. There are disadvantages to adding a new data partition and loading data directly into the table. The principal disadvantage of using the ALTER TABLE ... ADD PARTITION statement is that it prevents updates to the table both during the Load operation itself, and during any subsequent SET INTEGRITY statement. While both the ALTER TABLE ... ADD PARTITION statement and the ALTER TABLE ... ATTACH PARTITION statement cause package invalidation, the LOAD command followed by the ALTER ... ATTACH PARTITION operation yields better data availability. However, the ALTER TABLE ... ADD PARTITION statement followed by the IMPORT command

or a regular INSERT statement makes good sense for situations in which the data is not rolled-in in large blocks, but instead trickles in. Adding a data partition also makes sense in cases where the data being rolled in does not match the data partition boundaries.

Scenario 3: Using partitioned tables, the roll-in operation is simply an ATTACH operation of a newly loaded data partition

In this scenario, ATTACH is used to facilitate loading a new range of data into an existing partitioned table. Typically, data is loaded into a new, empty table to perform any necessary cleaning and checking of the data without impacting the target table. After the data is prepared, the newly loaded data partition is attached.

```
CREATE TABLE dec03(.....);
LOAD FROM data_file OF DEL REPLACE INTO dec03;
```

Before rolling in your table data, data cleansing might be required before the data is attached. Data cleansing activities include:

- 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 values 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 pre-aggregated during roll-in.)

Next, roll in the data:

```
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 and ENDING clauses must be supplied and the lower bound (STARTING) must be less than or equal to the upper bound (ENDING). In addition, the newly attached data partition must not overlap with an existing data partition range in the target table. If the highest range has been defined as MAXVALUE, then any attempt to attach a new high range fails because it overlaps the existing high range. This restriction also applies to MINVALUE. You cannot add or attach a new data partition in the middle unless it falls in an existing gap in the ranges. Boundaries not specified by the user are determined when the table is created.

The ALTER TABLE ... ATTACH operation drains all queries and invalidate packages dependent on the stock table. That is, existing queries complete normally before the ATTACH operation continues. Once the ATTACH operation is issued, any new queries accessing the stock table block on a lock. The stock table is z-locked (completely inaccessible) during this transition. The data in the attached data partition is not yet visible, because it has not yet been validated by the SET INTEGRITY statement. **Tip:**Issue a COMMIT WORK statement immediately after the ATTACH operation to make the table available for use.

```
COMMIT WORK;
```

The SET INTEGRITY statement is necessary to verify that the newly attached data is in range. It also does any necessary maintenance of indexes and other dependent objects such as MQTs. Until the SET INTEGRITY statement is committed, the new

data is not visible. The existing data in the stock table is fully accessible for both reading and writing if online SET INTEGRITY is used. The default while SET INTEGRITY is running is ALLOW NO ACCESS mode.

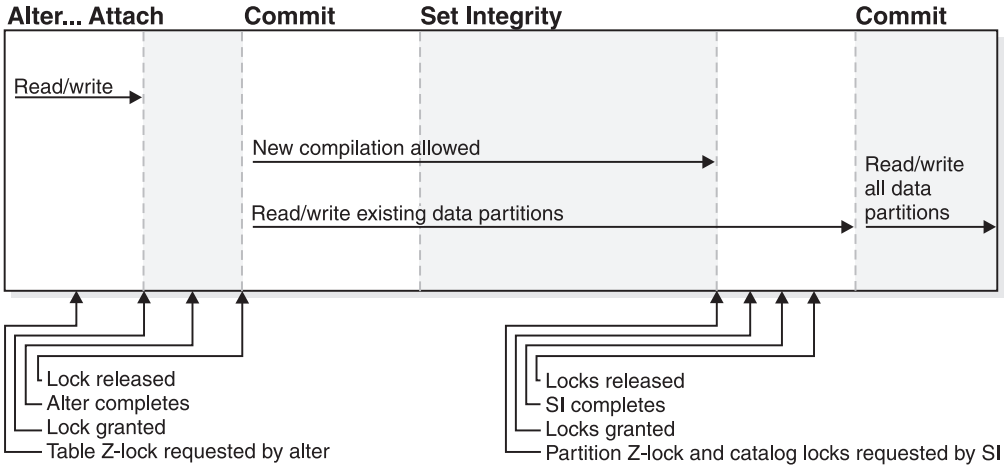


Figure 38. This figure demonstrates the stages of data availability during an ATTACH operation.

Note: While SET INTEGRITY is running, you cannot execute DDL or utility type operations on the table. The operations include but are not restricted to LOAD, REORG, REDISTRIBUTE, ALTER TABLE (for example, add columns, ADD, ATTACH, DETACH, TRUNCATE using ALTER to "not logged initially"), and INDEX CREATE.

```
SET INTEGRITY FOR stock ALLOW WRITE ACCESS
IMMEDIATE CHECKED FOR EXCEPTION IN stock USE stock_ex;
```

Set integrity validates the data in the newly attached data partition.

Next, commit the transaction to make the table available for use.

```
COMMIT WORK;
```

Any rows that are out of range, or violate other constraints, are moved to the exception table stock_ex. You can query stock_ex to inspect the violating rows, and possibly to clean them up and re-insert them into the table.

Chapter 14. Load

Parallelism and loading

The load utility takes advantage of a hardware configuration in which multiple processors or multiple storage devices are used, such as in a symmetric multiprocessor (SMP) environment.

There are several ways in which parallel processing of large amounts of data can take place using the load utility. One way is through the use of multiple storage devices, which allows for I/O parallelism during the load operation (see Figure 39). Another way involves the use of multiple processors in an SMP environment, which allows for intra-partition parallelism (see Figure 40). Both can be used together to provide even faster loading of data.

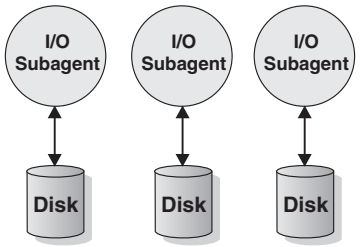


Figure 39. Taking Advantage of I/O Parallelism When Loading Data

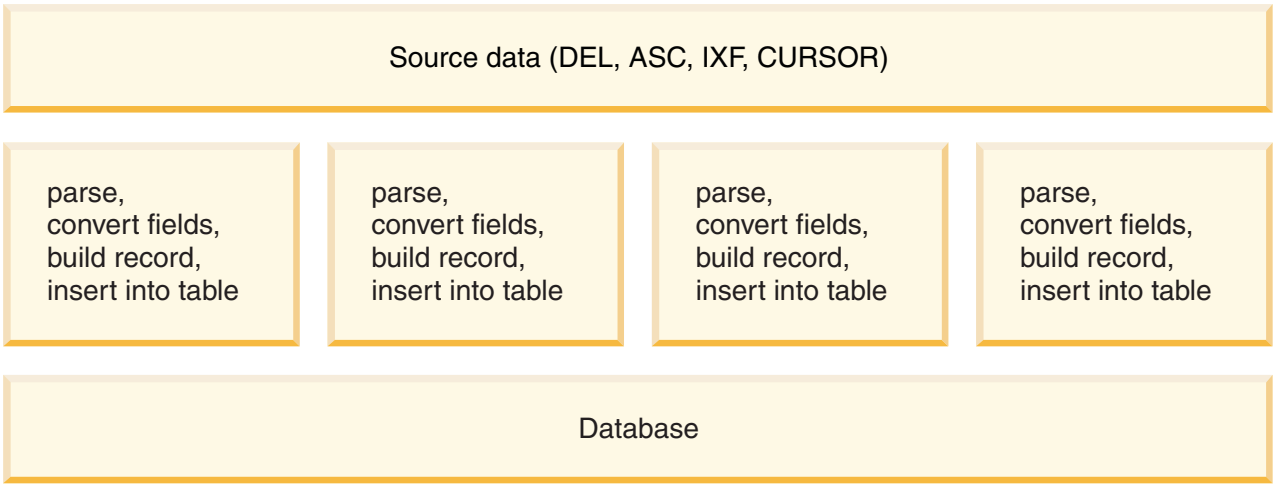


Figure 40. Taking Advantage of Intra-partition Parallelism When Loading Data

Multidimensional clustering considerations

The following restrictions apply when loading data into multidimensional clustering (MDC) tables:

- The SAVECOUNT option of the LOAD command is not supported.
- The `totalfreespace` file type modifier is not supported since these tables manage their own free space.

© Copyright IBM Corp. 1993, 2009

- The `anyorder` file type modifier is required for MDC tables. If a load is executed into an MDC table without the `anyorder` modifier, it will be explicitly enabled by the utility.

When using the LOAD command with an MDC table, violations of unique constraints are be handled as follows:

- If the table included a unique key prior to 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.

Performance Considerations

To improve the performance of the load utility when loading MDC tables, the *util_heap_sz* database configuration parameter value should be increased. The mdc-load algorithm performs significantly 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. When the DATA BUFFER option of the LOAD command is specified, its value should also be increased. If the LOAD command is being used to load several MDC tables concurrently, *util_heap_sz* should be increased accordingly.

MDC load operations always have a build phase since all MDC tables have block indexes.

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 should be set to a value that takes this into account.

A system temporary table with an index is used to load data into MDC tables. The size of the table is proportional to the number of distinct cells loaded. The size of each row in the table is proportional to the size of the MDC dimension key. To minimize disk I/O caused by the manipulation of this table during a load operation, ensure that the buffer pool for the temporary table space is large enough.

Load considerations for partitioned tables

All of the existing load features are supported when the target table is partitioned with the exception of the following general restrictions:

- Consistency points are not supported when the number of partitioning agents is greater than one.
- Loading data into a subset of data partitions while the remaining data partitions remain fully online is not supported.
- The exception table used by a load operation cannot be partitioned.
- A unique index cannot be rebuilt when the load utility is running in insert mode or restart mode, and the load target table has any detached dependents.

- Similar to loading MDC tables, exact ordering of input data records is not preserved when loading partitioned tables. Ordering is only maintained within the cell or data partition.
- Load operations utilizing multiple formatters on each database partition only preserve approximate ordering of input records. Running a single formatter on each database partition, groups the input records by cell or table partitioning key. To run a single formatter on each database partition, explicitly request CPU_PARALLELISM of 1.

General load behavior
The load utility inserts data records into the correct data partition. There is no requirement to use an external utility, such as a splitter, to partition the input data before loading.

The load utility does not access any detached or attached data partitions. Data is inserted into visible data partitions only. Visible data partitions are neither attached nor detached. In addition, a load replace operation does not truncate detached or attached data partitions. Since the load utility acquires locks on the catalog system tables, the load utility waits for any uncommitted ALTER TABLE transactions. Such transactions acquire an exclusive lock on the relevant rows in the catalog tables, and the exclusive lock must terminate before the load operation can proceed. This means that there can be no uncommitted ALTER TABLE ...ATTACH, DETACH, or ADD PARTITION transactions while load operation is running. Any input source records destined for an attached or detached data partition are rejected, and can be retrieved from the exception table if one is specified. An informational message is written to the message file to indicate some of the target table data partitions were in an attached or detached state. Locks on the relevant catalog table rows corresponding to the target table prevent users from changing the partitioning of the target table by issuing any ALTER TABLE ...ATTACH, DETACH, or ADD PARTITION operations while the load utility is running.

Handling of invalid rows
When the load utility encounters a record that does not belong to any of the visible data partitions the record is rejected and the load utility continues processing. The number of records rejected because of the range constraint violation is not explicitly displayed, but is included in the overall number of rejected records. Rejecting a record because of the range violation does not increase the number of row warnings. A single message (SQL0327N) is written to the load utility message file indicating that range violations are found, but no per-record messages are logged. In addition to all columns of the target table, the exception table includes columns describing the type of violation that had occurred for a particular row. Rows containing invalid data, including data that cannot be partitioned, are written to the dump file.

Because exception table inserts are expensive, you can control which constraint violations are inserted into the exception table. For instance, the default behavior of the load utility is to insert rows that were rejected because of a range constraint or unique constraint violation, but were otherwise valid, into the exception table. You can turn off this behavior by specifying, respectively, NORANGEEXC or NOUNIQUEEXC with the FOR EXCEPTION clause. If you specify that these constraint violations should not be inserted into the exception table, or you do not specify an exception table, information about rows violating the range constraint or unique constraint is lost.

History file
If the target table is partitioned, the corresponding history file entry does not

include a list of the table spaces spanned by the target table. A different operation granularity identifier ('R' instead of 'T') indicates that a load operation ran against a partitioned table.

Terminating a load operation
Terminating a load replace completely truncates all visible data partitions, terminating a load insert truncates all visible data partitions to their lengths before the load. Indexes are invalidated during a termination of an ALLOW READ ACCESS load operation that failed in the load copy phase. Indexes are also invalidated when terminating an ALLOW NO ACCESS load operation that touched the index (It is invalidated because the indexing mode is rebuild, or a key was inserted during incremental maintenance leaving the index in an inconsistent state). Loading data into multiple targets does not have any effect on load recovery operations except for the inability to restart the load operation from a consistency point taken during the load phase In this case, the SAVECOUNT load option is ignored if the target table is partitioned. This behavior is consistent with loading data into a MDC target table.

Generated columns
If a generated column is in any of the partitioning, dimension, or distribution keys, the `generatedoverride` file type modifier is ignored and the load utility generates values as if the `generatedignore` file type modifier is specified. Loading an incorrect generated column value in this case can place the record in the wrong physical location, such as the wrong data partition, MDC block or database partition. For example, once a record is on a wrong data partition, set integrity has to move it to a different physical location, which cannot be accomplished during online set integrity operations.

Data availability
The current ALLOW READ ACCESS load algorithm extends to partitioned tables. An ALLOW READ ACCESS load operation allows concurrent readers to access the whole table, including both loading and non-loading data partitions.

Data partition states
After a successful load, visible data partitions might change to either or both Set Integrity Pending or Read Access Only table state, under certain conditions. Data partitions might be placed in these states if there are constraints on the table which the load operation cannot maintain. Such constraints might include check constraints and detached materialized query tables. A failed load operation leaves all visible data partitions in the Load Pending table state.

Error isolation
Error isolation at the data partition level is not supported. Isolating the errors means continuing a load on data partitions that did not run into an error and stopping on data partitions that did run into an error. Errors can be isolated between different database partitions, but the load utility cannot commit transactions on a subset of visible data partitions and roll back the remaining visible data partitions.

Other considerations

- Incremental indexing is not supported if any of the indexes are marked invalid. An index is considered invalid if it requires a rebuild or if detached dependents require validation with the SET INTEGRITY statement.
- Loading into tables partitioned using any combination of partitioned by range, distributed by hash, or organized by dimension algorithms is also supported.

- For log records which include the list of object and table space IDs affected by the load, the size of these log records (LOAD START and COMMIT (PENDING LIST)) could grow considerably and hence reduce the amount of active log space available to other applications.
- When a table is both partitioned and distributed, a partitioned database load might not affect all database partitions. Only the objects on the output database partitions are changed.
- During a load operation, memory consumption for partitioned tables increases with the number of tables. Note, that the total increase is not linear as only a small percentage of the overall memory requirement is proportional to the number of data partitions.

Chapter 15. Loading data in a partitioned database environment

Load overview–partitioned database environments

In a multi-partition database, large amounts of data are located across many database partitions. Distribution keys are used to determine on which database partition each portion of the data resides. The data must be *distributed* before it can be loaded at the correct database partition.

When loading tables in a multi-partition database, the load utility can:

- Distribute input data in parallel
- Load data simultaneously on corresponding database partitions
- Transfer data from one system to another system

Loading data into a multi-partition database takes place in two phases: the *setup phase*, during which database partition resources such as table locks are acquired, and the *load phase*, during which the data is loaded into the database partitions. You can use the ISOLATE_PART_ERRS option of the LOAD command to select how errors are handled during either of these phases, and how errors on one or more of the database partitions affect the load operation on the database partitions that are not experiencing errors.

When loading data into a multi-partition database you can use one of the following modes:

PARTITION_AND_LOAD
Data is distributed (perhaps in parallel) and loaded simultaneously on the corresponding database partitions.

PARTITION_ONLY
Data is distributed (perhaps in parallel) and the output is written to files in a specified location on each loading database partition. Each file includes a partition header that specifies how the data was distributed across the database partitions, and that the file can be loaded into the database using the LOAD_ONLY mode.

LOAD_ONLY
Data is assumed to be already distributed across the database partitions; the distribution process is skipped, and the data is loaded simultaneously on the corresponding database partitions.

LOAD_ONLY_VERIFY_PART
Data is assumed to be already distributed across the database partitions, but the data file does not contain a partition header. The distribution process is skipped, and the data is loaded simultaneously on the corresponding database partitions. During the load operation, each row is checked to verify that it is on the correct database partition. Rows containing database partition violations are placed in a dump file if the `dumpfile` file type modifier is specified. Otherwise, the rows are discarded. If database partition violations exist on a particular loading database partition, a single warning is written to the load message file for that database partition.

© Copyright IBM Corp. 1993, 2009

ANALYZE
An optimal distribution map with even distribution across all database partitions is generated.

Concepts and terminology

The following terminology is used when discussing the behavior and operation of the load utility in a partitioned database environment with multiple database partitions:

- The *coordinator partition* is the database partition to which the user connects in order to perform the load operation. In the PARTITION_AND_LOAD, PARTITION_ONLY, and ANALYZE modes, it is assumed that the data file resides on this database partition unless the CLIENT option of the LOAD command is specified. Specifying CLIENT indicates that the data to be loaded resides on a remotely connected client.
- In the PARTITION_AND_LOAD, PARTITION_ONLY, and ANALYZE modes, the *pre-partitioning agent* reads the user data and distributes it in round-robin fashion to the *partitioning agents* which then distribute the data. This process is always performed on the coordinator partition. A maximum of one partitioning agent is allowed per database partition for any load operation.
- In the PARTITION_AND_LOAD, LOAD_ONLY, and LOAD_ONLY_VERIFY_PART modes, *load agents* run on each output database partition and coordinate the loading of data to that database partition.
- *Load to file agents* run on each output database partition during a PARTITION_ONLY load operation. They receive data from partitioning agents and write it to a file on their database partition.
- The SOURCEUSEREXIT option provides a facility through which the load utility can execute a customized script or executable, referred to herein as the *user exit*.

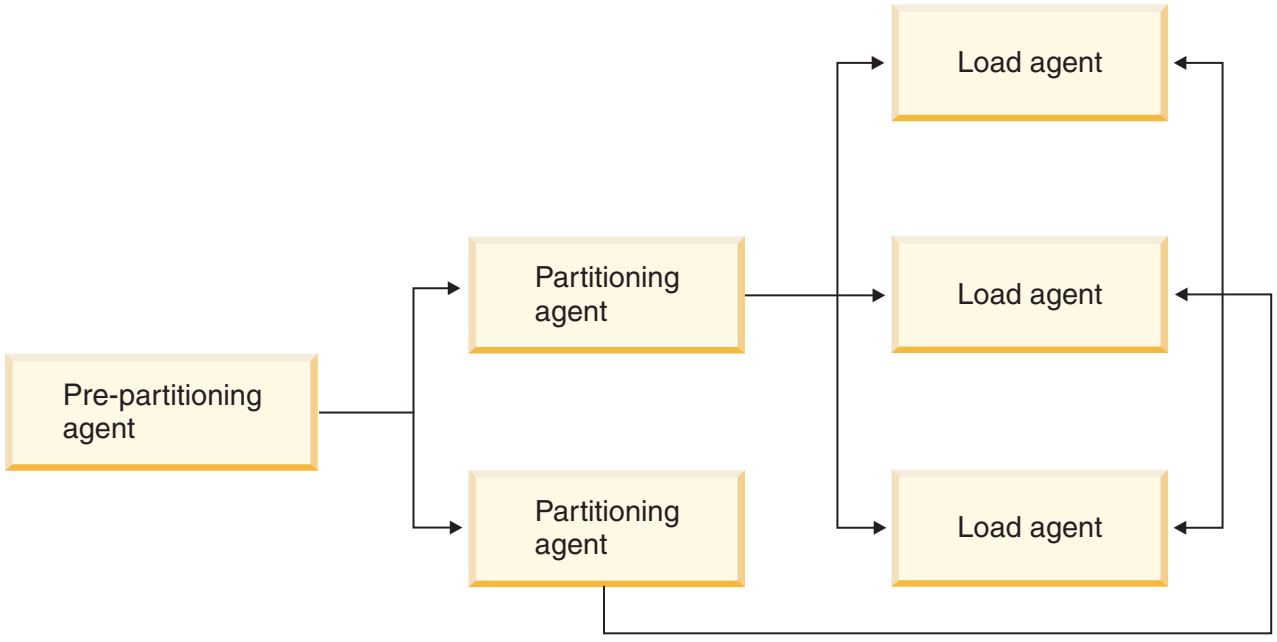


Figure 41. Partitioned Database Load Overview. The source data is read by the pre-partitioning agent, and approximately half of the data is sent to each of two partitioning agents which distribute the data and send it to one of three database partitions. The load agent at each database partition loads the data.

Loading data in a partitioned database environment–hints and tips

The following is some information to consider before loading a table in a multi-partition database:

- Familiarize yourself with the load configuration options by using the utility with small amounts of data.
- If the input data is already sorted, or in some chosen order, and you want to maintain that order during the loading process, only one database partition should be used for distributing. Parallel distribution cannot guarantee that the data is loaded in the same order it was received. The load utility chooses a single partitioning agent by default if the `anyorder` modifier is not specified on the LOAD command.
- If large objects (LOBs) are being loaded from separate files (that is, if you are using the `lobsinfile` modifier through the load utility), all directories containing the LOB files must be read-accessible to all the database partitions where loading is taking place. The LOAD *lob-path* parameter must be fully qualified when working with LOBs.
- You can force a job running in a multi-partition database to continue even if the load operation detects (at startup time) that some loading database partitions or associated table spaces or tables are offline, by setting the ISOLATE_PART_ERRS option to SETUP_ERRS_ONLY or SETUP_AND_LOAD_ERRS.
- Use the STATUS_INTERVAL load configuration option to monitor the progress of a job running in a multi-partition database. The load operation produces messages at specified intervals indicating how many megabytes of data have been read by the pre-partitioning agent. These messages are dumped to the pre-partitioning agent message file. To view the contents of this file during the load operation, connect to the coordinator partition and issue a LOAD QUERY command against the target table.
- Better performance can be expected if the database partitions participating in the distribution process (as defined by the PARTITIONING_DBPARTNUMS option) are different from the loading database partitions (as defined by the OUTPUT_DBPARTNUMS option), since there is less contention for CPU cycles. When loading data into a multi-partition database, invoke the load utility on a database partition that is not participating in either the distributing or the loading operation.
- Specifying the MESSAGES parameter in the LOAD command saves the messages files from the pre-partitioning, partitioning, and load agents for reference at the end of the load operation. To view the contents of these files during a load operation, connect to the desired database partition and issue a LOAD QUERY command against the target table.
- The load utility chooses only one output database partition on which to collect statistics. The RUN_STAT_DBPARTNUM database configuration option can be used to specify the database partition.
- Before loading data in a multi-partition database, run the Design Advisor to determine the best partition for each table. For more information, see "The Design Advisor" in *Tuning Database Performance*.

Troubleshooting

If the load utility is hanging, you can:

- Use the STATUS_INTERVAL parameter to monitor the progress of a multi-partition database load operation. The status interval information is dumped to the pre-partitioning agent message file on the coordinator partition.

- Check the partitioning agent messages file to see the status of the partitioning agent processes on each database partition. If the load is proceeding with no errors, and the TRACE option has been set, there should be trace messages for a number of records in these message files.
- Check the load messages file to see if there are any load error messages.

  Note: You must specify the MESSAGES option of the LOAD command in order for these files to exist.
- Interrupt the current load operation if you find errors suggesting that one of the load processes encountered errors.

Loading data in a partitioned database environment

Using the load utility to load data into a partitioned database environment.

Before loading a table in a multi-partition database:

1. Ensure that the *svcename* database manager configuration parameter and the **DB2COMM** profile registry variable are set correctly. This is important because the load utility uses TCP/IP to transfer data from the pre-partitioning agent to the partitioning agents, and from the partitioning agents to the loading database partitions.
2. Before invoking the load utility, you must be connected to (or be able to implicitly connect to) the database into which the data will be loaded. Since the load utility will issue a COMMIT statement, you should complete all transactions and release any locks by issuing either a COMMIT or a ROLLBACK statement before beginning the load operation. If the PARTITION_AND_LOAD, PARTITION_ONLY, or ANALYZE mode is being used, the data file that is being loaded must reside on this database partition unless:
   a. The CLIENT option has been specified, in which case the data must reside on the client machine;
   b. The input source type is CURSOR, in which case there is no input file.
3. Run the Design Advisor to determine the best database partition for each table. For more information, see "The Design Advisor" in *Tuning Database Performance*.

The following restrictions apply when using the load utility to load data in a multi-partition database:

- The location of the input files to the load operation cannot be a tape device.
- The ROWCOUNT option is not supported unless the ANALYZE mode is being used.
- If the target table has an identity column that is needed for distributing and the `identityoverride` file type modifier is not specified, or if you are using multiple database partitions to distribute and then load the data, the use of a SAVECOUNT greater than 0 on the LOAD command is not supported.
- If an identity column forms part of the distribution key, only the PARTITION_AND_LOAD mode is supported.
- The LOAD_ONLY and LOAD_ONLY_VERIFY_PART modes cannot be used with the CLIENT option of the LOAD command.
- The LOAD_ONLY_VERIFY_PART mode cannot be used with the CURSOR input source type.

- The distribution error isolation modes LOAD_ERRS_ONLY and SETUP_AND_LOAD_ERRS cannot be used with the ALLOW READ ACCESS and COPY YES options of the LOAD command.
- Multiple load operations can load data into the same table concurrently if the database partitions specified by the OUTPUT_DBPARTNUMS and PARTITIONING_DBPARTNUMS options do not overlap. For example, if a table is defined on database partitions 0 through 3, one load operation can load data into database partitions 0 and 1 while a second load operation can load data into database partitions 2 and 3.
- Only Non-delimited ASCII (ASC) and Delimited ASCII (DEL) files can be distributed across tables spanning multiple database partitions. PC/IXF files cannot be distributed, however, you can load a PC/IXF file into a table that is distributed over multiple database partitions using the load operation in the LOAD_ONLY_VERIFY_PART mode.

The following examples illustrate how to use the LOAD command to initiate various types of load operations. The database used in the following examples has five database partitions: 0, 1, 2, 3 and 4. Each database partition has a local directory `/db2/data/`. Two tables, TABLE1 and TABLE2, are defined on database partitions 0, 1, 3 and 4. When loading from a client, the user has access to a remote client that is not one of the database partitions.

Loading from a server partition

Distribute and load example

In this scenario you are connected to a database partition that might or might not be a database partition where TABLE1 is defined. The data file `load.del` resides in the current working directory of this database partition. To load the data from `load.del` into all of the database partitions where TABLE1 is defined, issue the following command:

```
LOAD FROM LOAD.DEL of DEL REPLACE INTO TABLE1
```

Note: In this example, default values are used for all of the configuration parameters for partitioned database environments: The MODE parameter default to PARTITION_AND_LOAD, the OUTPUT_DBPARTNUMS options default to all database partitions on which TABLE1 is defined, and the PARTITIONING_DBPARTNUMS defaults to the set of database partitions selected according to the LOAD command rules for choosing database partitions when none are specified.

To perform a load operation where data is distributed over database partitions 3 and 4, issue the following command:

```
LOAD FROM LOAD.DEL of DEL REPLACE INTO TABLE1
                    PARTITIONED DB CONFIG PARTITIONING_DBPARTNUMS (3,4)
```

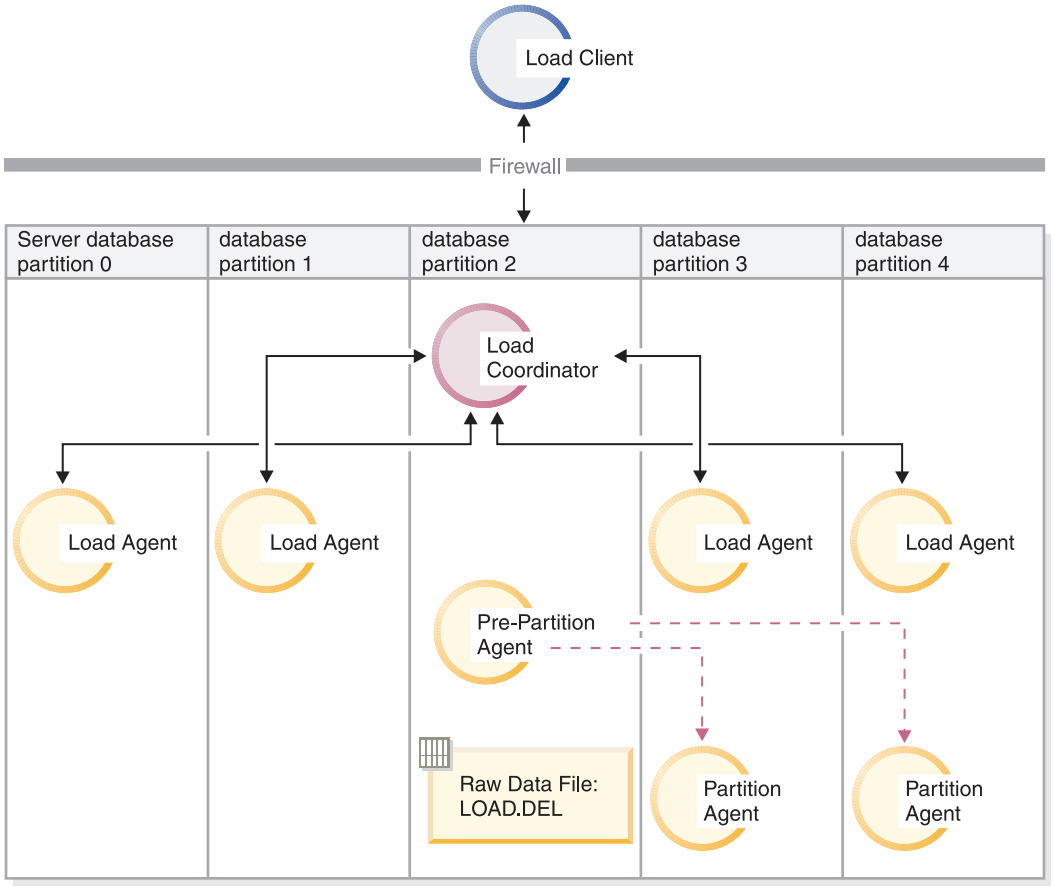


Figure 42. This diagram illustrates the behavior resulting when the previous command is issued. Data is loaded into database partitions 3 and 4.

Distribute only example

In this scenario you are connected to a database partition that might or might not be a database partition where TABLE1 is defined. The data file `load.del` resides in the current working directory of this database partition. To distribute (but not load) `load.del` to all the database partitions on which TABLE1 is defined, using database partitions 3 and 4 issue the following command:

```
LOAD FROM LOAD.DEL of DEL REPLACE INTO TABLE1
                      PARTITIONED DB CONFIG MODE PARTITION_ONLY
                                            PART_FILE_LOCATION /db2/data
                                            PARTITIONING_DBPARTNUMS (3,4)
```

This results in a file `load.del.xxx` being stored in the `/db2/data` directory on each database partition, where `xxx` is a three-digit representation of the database partition number.

To distribute the `load.del` file to database partitions 1 and 3, using only 1 partitioning agent running on database partition 0 (which is the default for PARTITIONING_DBPARTNUMS), issue the following command:

```
LOAD FROM LOAD.DEL OF DEL REPLACE INTO TABLE1
                      PARTITIONED DB CONFIG MODE PARTITION_ONLY
                                            PART_FILE_LOCATION /db2/data
                                            OUTPUT_DBPARTNUMS (1,3)
```

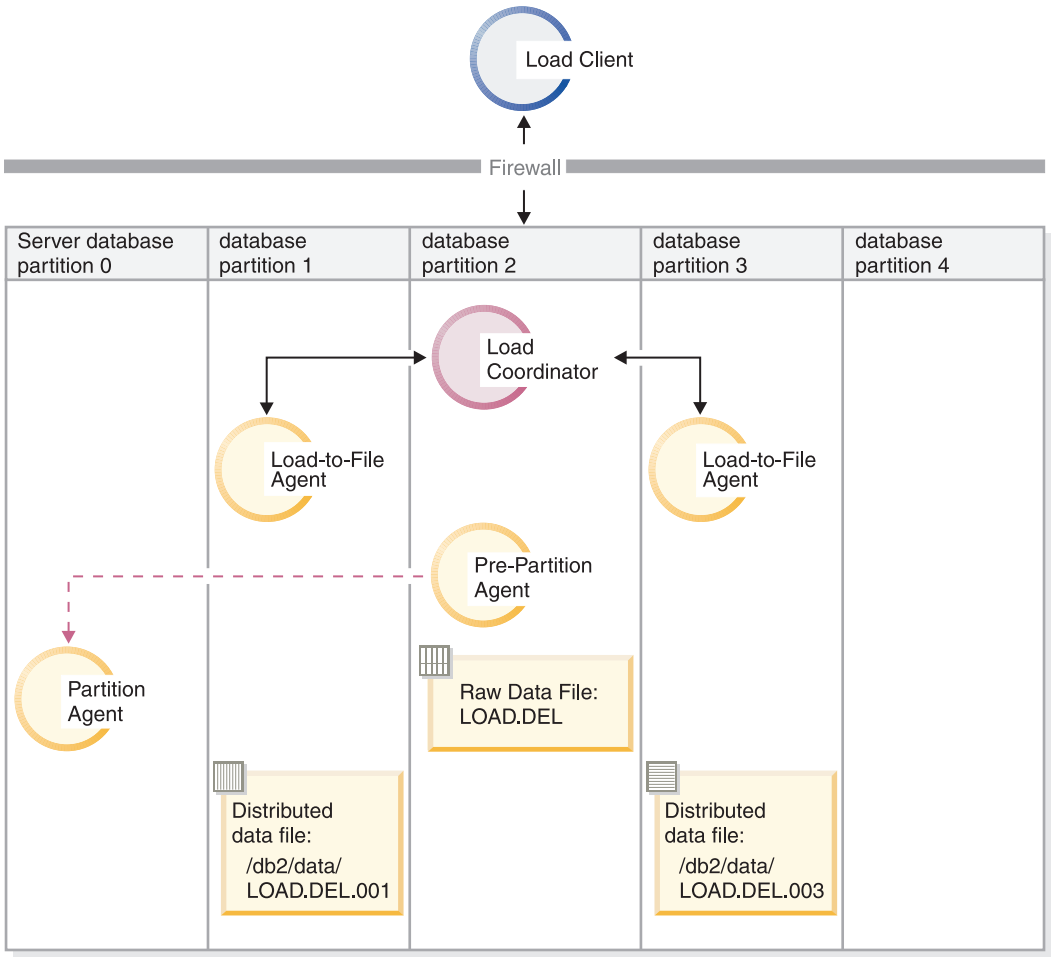


Figure 43. This diagram illustrates the behavior that results when the previous command is issued. Data is loaded into database partitions 1 and 3, using 1 partitioning agent running on database partition 0.

Load only example

If you have already performed a load operation in the PARTITION_ONLY mode and want to load the partitioned files in the `/db2/data` directory of each loading database partition to all the database partitions on which TABLE1 is defined, issue the following command:

```
LOAD FROM LOAD.DEL OF DEL REPLACE INTO TABLE1
                        PARTITIONED DB CONFIG MODE LOAD_ONLY
                                              PART_FILE_LOCATION /db2/data
```

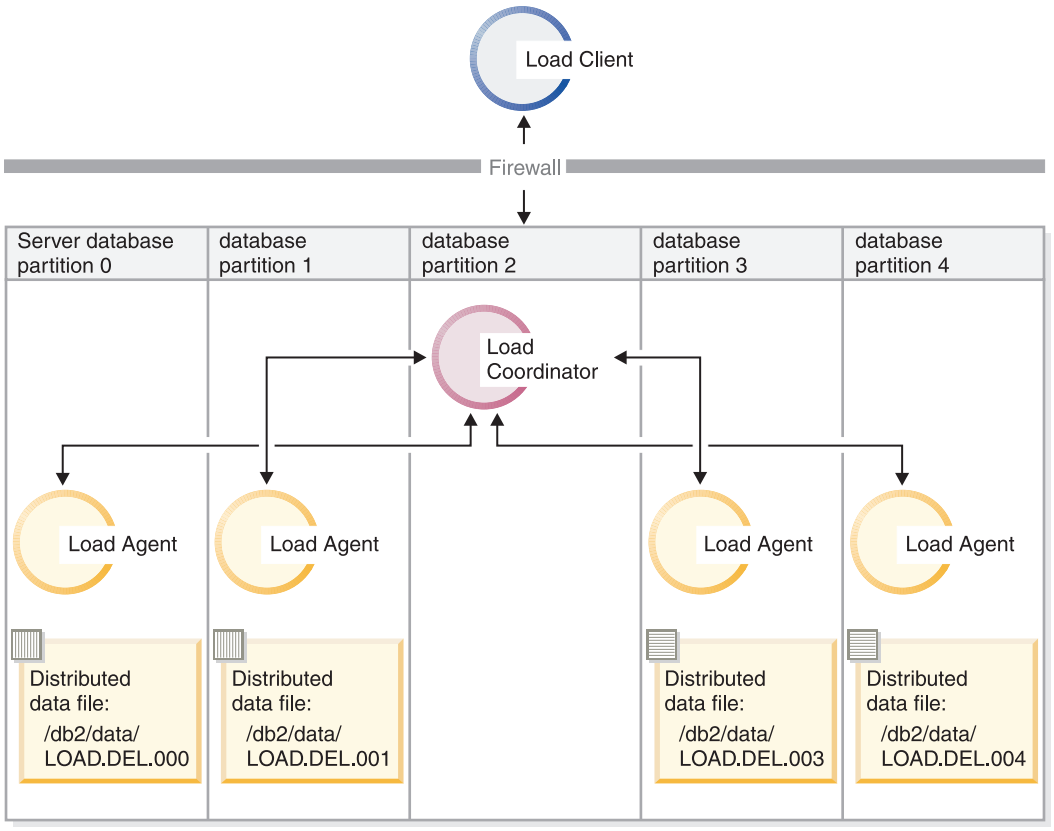


Figure 44. This diagram illustrates the behavior resulting when the previous command is issued. Distributed data is loaded to all database partitions where TABLE1 is defined.

To load into database partition 4 only, issue the following command:

```
LOAD FROM LOAD.DEL OF DEL REPLACE INTO TABLE1
                      PARTITIONED DB CONFIG MODE LOAD_ONLY
                                            PART_FILE_LOCATION /db2/data
                                            OUTPUT_DBPARTNUMS (4)
```

Loading pre-distributed files without distribution map headers

The LOAD command can be used to load data files without distribution headers directly into several database partitions. If the data files exist in the `/db2/data` directory on each database partition where TABLE1 is defined and have the name `load.del.xxx`, where `xxx` is the database partition number, the files can be loaded by issuing the following command:

```
LOAD FROM LOAD.DEL OF DEL modified by dumpfile=rejected.rows
                       REPLACE INTO TABLE1
                       PARTITIONED DB CONFIG MODE LOAD_ONLY_VERIFY_PART
                                             PART_FILE_LOCATION /db2/data
```

To load the data into database partition 1 only, issue the following command:

```
LOAD FROM LOAD.DEL OF DEL modified by dumpfile=rejected.rows
                       REPLACE INTO TABLE1
                       PARTITIONED DB CONFIG MODE LOAD_ONLY_VERIFY_PART
                                             PART_FILE_LOCATION /db2/data
                                             OUTPUT_DBPARTNUMS (1)
```

Note: Rows that do not belong on the database partition from which they were loaded are rejected and put into the dumpfile, if one has been specified.

Loading from a remote client to a multi-partition database

To load data into a multi-partition database from a file that is on a remote client, you must specify the CLIENT option of the LOAD command to indicate that the data file is not on a server partition. For example:

```
LOAD CLIENT FROM LOAD.DEL OF DEL REPLACE INTO TABLE1
```

Note: You cannot use the LOAD_ONLY or LOAD_ONLY_VERIFY_PART modes with the CLIENT option.

Loading from a cursor

As in a single-partition database, you can load from a cursor into a multi-partition database. In this example, for the PARTITION_ONLY and LOAD_ONLY modes, the PART_FILE_LOCATION option must specify a fully qualified file name. This name is the fully qualified base file name of the distributed files that are created or loaded on each output database partition. Multiple files can be created with the specified base name if there are LOB columns in the target table.

To distribute all the rows in the answer set of the statement `SELECT * FROM TABLE1` to a file on each database partition named `/db2/data/select.out.xxx` (where xxx is the database partition number), for future loading into TABLE2, issue the following commands:

```
DECLARE C1 CURSOR FOR SELECT * FROM TABLE1

LOAD FROM C1 OF CURSOR REPLACE INTO TABLE2
                       PARTITIONED DB CONFIG MODE PARTITION_ONLY
                       PART_FILE_LOCATION /db2/data/select.out
```

The data files produced by the above operation can then be loaded by issuing the following LOAD command:

```
LOAD FROM C1 OF CURSOR REPLACE INTO TABLE2
                       PARTITIONED CB CONFIG MODE LOAD_ONLY
                       PART_FILE_LOCATION /db2/data/select.out
```

Monitoring a load operation in a partitioned database environment using the LOAD QUERY command

During a load operation in a partitioned database environment, message files are created by some of the load processes on the database partitions where they are being executed.

The message files store all information, warning, and error messages produced during the execution of the load operation. The load processes that produce message files that can be viewed by the user are the load agent, pre-partitioning agent, and partitioning agent. The content of the message file is only available after the load operation is finished.

You can connect to individual database partitions during a load operation and issue the LOAD QUERY command against the target table. When issued from the CLP, this command displays the contents of all the message files that currently reside on that database partition for the table that is specified in the LOAD QUERY command.

For example, table TABLE1 is defined on database partitions 0 through 3 in database WSDB. You are connected to database partition 0 and issue the following LOAD command:

```
load from load.del of del replace into table1 partitioned db config
    partitioning_dbpartnums (1)
```

This command initiates a load operation that includes load agents running on database partitions 0, 1, 2, and 3; a partitioning agent running on database partition 1; and a pre-partitioning agent running on database partition 0.

Database partition 0 contains one message file for the pre-partitioning agent and one for the load agent on that database partition. To view the contents of these files at the same time, start a new session and issue the following commands from the CLP:

```
set client connect_node 0
connect to wsdb
load query table table1
```

Database partition 1 contains one file for the load agent and one for the partitioning agent. To view the contents of these files, start a new session and issue the following commands from the CLP:

```
set client connect_node 1
connect to wsdb
load query table table1
```

Note: The messages generated by the STATUS_INTERVAL load configuration option appear in the pre-partitioning agent message file. To view these message during a load operation, you must connect to the coordinator partition and issue the LOAD QUERY command.

Saving the contents of message files

If a load operation is initiated through the db2Load API, the messages option (piLocalMsgFileName) must be specified and the message files are brought from the server to the client and stored for you to view.

For multi-partition database load operations initiated from the CLP, the message files are not displayed to the console or retained. To save or view the contents of these files after a multi-partition database load is complete, the MESSAGES option of the LOAD command must be specified. If this option is used, once the load operation is complete the message files on each database partition are transferred to the client machine and stored in files using the base name indicated by the MESSAGES option. For multi-partition database load operations, the name of the file corresponding to the load process that produced it is listed below:

Process Type	File Name
Load Agent	<message-file-name>.LOAD.<dbpartition-number>
Partitioning Agent	<message-file-name>.PART.<dbpartition-number>
Pre-partitioning Agent	<message-file-name>.PREP.<dbpartition-number>

For example, if the MESSAGES option specifies `/wsdb/messages/load`, the load agent message file for database partition 2 is `/wsdb/messages/load.LOAD.002`.

Note: It is strongly recommended that the MESSAGES option be used for multi-partition database load operations initiated from the CLP.

Resuming, restarting, or terminating load operations in a partitioned database environment

The steps you need to take following failed load operations in a partitioned database environment depend on when the failure occurred.

The load process in a multi-partition database consists of two stages:

1. The *setup stage*, during which database partition-level resources such as table locks on output database partitions are acquired

   In general, if a failure occurs during the setup stage, restart and terminate operations are not necessary. What you need to do depends on the error isolation mode that was specified for the failed load operation.

   If the load operation specified that setup stage errors were not to be isolated, the entire load operation is cancelled and the state of the table on each database partition is rolled back to the state it was in prior to the load operation.

   If the load operation specified that setup stage errors were to be isolated, the load operation continues on the database partitions where the setup stage was successful, but the table on each of the failing database partitions is rolled back to the state it was in prior to the load operation. This means that a single load operation can fail at different stages if some partitions fail during the setup stage and others fail during the load stage
2. The *load stage*, during which data is formatted and loaded into tables on the database partitions

   If a load operation fails on at least one database partition during the load stage of a multi-partition database load operation, a load RESTART or load TERMINATE command must be issued. This is necessary because loading data in a multi-partition database is done through a single transaction.

   You should choose a load RESTART if you can fix the problems that caused the failed load to occur. This saves time because if a load restart operation is initiated, the load operation continues from where it left off on all database partitions.

   You should choose a load TERMINATE if you want the table returned to the state it was in prior to the initial load operation.

Procedure:

Determining when a load failed

The first thing you need to do if your load operation in a partitioned environment fails is to determine on which partitions it failed and at what stage each of them failed. This is done by looking at the partition summary. If the load command was issued from the CLP, the partition summary is displayed at the end of the load (see example below). If the load command was issued from the db2Load API, the partition summary is contained in the poAgentInfoList field of the db2PartLoadOut structure.

If there is an entry of "LOAD" for "Agent Type", for a given partition, then that partition reached the load stage, otherwise a failure occurred during the setup stage. A negative SQL Code indicates that it failed. In the following example, the load failed on partition 1 during the load stage.

```
Agent Type     Node     SQL Code     Result
_____________________________________________________________________
LOAD           000      +00000000    Success.
_____________________________________________________________________
LOAD           001      -00000289    Error. May require RESTART.
_____________________________________________________________________
LOAD           002      +00000000    Success.
_____________________________________________________________________
LOAD           003      +00000000    Success.
.
.
.
```

Resuming, restarting, or terminating a failed load

Only loads with the ISOLATE_PART_ERRS option specifying SETUP_ERRS_ONLY or SETUP_AND_LOAD_ERRS should fail during the setup stage. For loads that fail on at least one output database partition fail during this stage, you can issue a LOAD REPLACE or LOAD INSERT command. Use the OUTPUT_DBPARTNUMS option to specify only those database partitions on which it failed.

For loads that fail on at least one output database partition during the load stage, issue a load RESTART or load TERMINATE command.

For loads that fail on at least one output database partition during the setup stage and at least one output database partition during the load stage, you need to perform two load operations to resume the failed load–one for the setup stage failures and one for the load stage failures, as previously described. To effectively undo this type of failed load operation, issue a load TERMINATE command. However, after issuing the command, you have to account for all partitions because no changes were made to the table on the partitions that failed during the setup stage, and all the changes have been undone for the partitions that failed during the load stage.

For example, TABLE1 is defined on database partitions 0 through 3 in database WSDB. The following command is issued:

```
load from load.del of del insert into table1 partitioned db config
isolate_part_errs setup_and_load_errs
```

There is a failure on output database partition 1 during the setup stage. Since setup stage errors are isolated, the load operation continues, but there is a failure on partition 3 during the load stage. To resume the load operation, you would issue the following commands:

```
load from load.del of del replace into table1 partitioned db config
output_dbpartnums (1)

load from load.del of del restart into table1 partitioned db config
isolate_part_errs setup_and_load_errs
```

Note: For load restart operations, the options specified in the LOAD RESTART command will be honored, so it is important that they are identical to the ones specified in the original LOAD command.

Load configuration options for partitioned database environments

MODE X

Specifies the mode in which the load operation occurs when loading a multi-partition database. PARTITION_AND_LOAD is the default. Valid values are:

- PARTITION_AND_LOAD. Data is distributed (perhaps in parallel) and loaded simultaneously on the corresponding database partitions.
- PARTITION_ONLY. Data is distributed (perhaps in parallel) and the output is written to files in a specified location on each loading database partition. For file types other than CURSOR, the format of the output file name on each database partition is `filename.xxx`, where `filename` is the input file name specified in the LOAD command and `xxx` is the 3-digit database partition number. For the CURSOR file type, the name of the output file on each database partition is determined by the PART_FILE_LOCATION option. See the PART_FILE_LOCATION option for details on how to specify the location of the distribution file for each database partition.

  Note:

  1. This mode cannot be used for a CLI load operation.
  2. If the table contains an identity column that is needed for distribution, then this mode is not supported, unless the `identityoverride` file type modifier is specified.
  3. Distribution files generated for file type CURSOR are not compatible between DB2 releases. This means that distribution files of file type CURSOR that were generated in a previous release cannot be loaded using the LOAD_ONLY mode. Similarly, distribution files of file type CURSOR that were generated in the current release cannot be loaded in a future release using the LOAD_ONLY mode.
- LOAD_ONLY. Data is assumed to be already distributed; the distribution process is skipped, and the data is loaded simultaneously on the corresponding database partitions. For file types other than CURSOR, the format of the input file name for each database partition should be `filename.xxx`, where `filename` is the name of the file specified in the LOAD command and `xxx` is the 3-digit database partition number. For the CURSOR file type, the name of the input file on each database partition is determined by the PART_FILE_LOCATION option. See the PART_FILE_LOCATION option for details on how to specify the location of the distribution file for each database partition.

  Note:

  1. This mode cannot be used for a CLI load operation, or when the CLIENT option of LOAD command is specified.
  2. If the table contains an identity column that is needed for distribution, then this mode is not supported, unless the `identityoverride` file type modifier is specified.
- LOAD_ONLY_VERIFY_PART. Data is assumed to be already distributed, but the data file does not contain a partition header. The distributing process is skipped, and the data is loaded simultaneously on the corresponding database partitions. During the load operation, each row is checked to verify that it is on the correct database partition. Rows containing database partition violations are placed in a dumpfile if the `dumpfile` file type modifier is specified. Otherwise, the rows are discarded. If database partition violations exist on a particular loading database partition, a single warning is written to the load message file for that database partition. The format of the input file name for each database partition should be `filename.xxx`, where `filename` is the name of the file specified in the LOAD command and `xxx` is the 3-digit

database partition number. See the PART_FILE_LOCATION option for details on how to specify the location of the distribution file for each database partition.

Note:

1. This mode cannot be used for a CLI load operation, or when the CLIENT option of LOAD command is specified.
2. If the table contains an identity column that is needed for distribution, then this mode is not supported, unless the `identityoverride` file type modifier is specified.

- ANALYZE. An optimal distribution map with even distribution across all database partitions is generated.

PART_FILE_LOCATION X

In the PARTITION_ONLY, LOAD_ONLY, and LOAD_ONLY_VERIFY_PART modes, this parameter can be used to specify the location of the distributed files. This location must exist on each database partition specified by the OUTPUT_DBPARTNUMS option. If the location specified is a relative path name, the path is appended to the current directory to create the location for the distributed files.

For the CURSOR file type, this option must be specified, and the location must refer to a fully qualified file name. This name is the fully qualified base file name of the distributed files that are created on each output database partition in the PARTITION_ONLY mode, or the location of the files to be read from for each database partition in the LOAD_ONLY mode. When using the PARTITION_ONLY mode, multiple files can be created with the specified base name if the target table contains LOB columns.

For file types other than CURSOR, if this option is not specified, the current directory is used for the distributed files.

OUTPUT_DBPARTNUMS X

`X` represents a list of database partition numbers. The database partition numbers represent the database partitions on which the load operation is to be performed. The database partition numbers must be a subset of the database partitions on which the table is defined. All database partitions are selected by default. The list must be enclosed in parentheses and the items in the list must be separated by commas. Ranges are permitted (for example, (0, 2 to 10, 15)).

PARTITIONING_DBPARTNUMS X

`X` represents a list of database partition numbers that are used in the distribution process. The list must be enclosed in parentheses and the items in the list must be separated by commas. Ranges are permitted (for example, (0, 2 to 10, 15)). The database partitions specified for the distribution process can be different from the database partitions being loaded. If PARTITIONING_DBPARTNUMS is not specified, the load utility determines how many database partitions are needed and which database partitions to use in order to achieve optimal performance.

If the `anyorder` file type modifier is not specified in the LOAD command, only one partitioning agent is used in the load session. Furthermore, if there is only one database partition specified for the OUTPUT_DBPARTNUMS option, or the coordinator partition of the load operation is not an element of OUTPUT_DBPARTNUMS, the coordinator partition of the load operation is used in the distribution process.

Otherwise, the first database partition (not the coordinator partition) in OUTPUT_DBPARTNUMS is used in the distribution process.

If the `anyorder` file type modifier is specified, the number of database partitions used in the distribution process is determined as follows: (number of partitions in OUTPUT_DBPARTNUMS/4 + 1).

MAX_NUM_PART_AGENTS X
Specifies the maximum numbers of partitioning agents to be used in a load session. The default is 25.

ISOLATE_PART_ERRS X
Indicates how the load operation reacts to errors that occur on individual database partitions. The default is LOAD_ERRS_ONLY, unless both the ALLOW READ ACCESS and COPY YES options of the LOAD command are specified, in which case the default is NO_ISOLATION. Valid values are:

- SETUP_ERRS_ONLY. Errors that occur on a database partition during setup, such as problems accessing a database partition, or problems accessing a table space or table on a database partition, cause the load operation to stop on the failing database partitions but to continue on the remaining database partitions. Errors that occur on a database partition while data is being loaded cause the entire operation to fail.
- LOAD_ERRS_ONLY. Errors that occur on a database partition during setup cause the entire load operation to fail. If an error occurs while data is being loaded, the load operation will stop on the database partition where the error occurred. The load operation continues on the remaining database partitions until a failure occurs or until all the data is loaded. The newly loaded data will not be visible until a load restart operation is performed and completes successfully.

  Note: This mode cannot be used when both the ALLOW READ ACCESS and the COPY YES options of the LOAD command are specified.
- SETUP_AND_LOAD_ERRS. In this mode, database partition-level errors during setup or loading data cause processing to stop only on the affected database partitions. As with the LOAD_ERRS_ONLY mode, when partition errors do occur while data is loaded, newly loaded data will not be visible until a load restart operation is performed and completes successfully.

  Note: This mode cannot be used when both the ALLOW READ ACCESS and the COPY YES options of the LOAD command are specified.
- NO_ISOLATION. Any error during the load operation causes the load operation to fail.

STATUS_INTERVAL X
`X` represents how often you are notified of the volume of data that has been read. The unit of measurement is megabytes (MB). The default is 100 MB. Valid values are whole numbers from 1 to 4000.

PORT_RANGE X
`X` represents the range of TCP ports used to create sockets for internal communications. The default range is from 6000 to 6063. If defined at the time of invocation, the value of the **DB2ATLD_PORTS** registry variable

replaces the value of the PORT_RANGE load configuration option. For the **DB2ATLD_PORTS** registry variable, the range should be provided in the following format:

```
<lower-port-number:higher-port-number>
```

From the CLP, the format is:

```
( lower-port-number, higher-port-number )
```

CHECK_TRUNCATION
Specifies that the program should check for truncation of data records at input/output. The default behavior is that data is not checked for truncation at input/output.

MAP_FILE_INPUT X
`X` specifies the input file name for the distribution map. This parameter must be specified if the distribution map is customized, as it points to the file containing the customized distribution map. A customized distribution map can be created by using the db2gpmap program to extract the map from the database system catalog table, or by using the ANALYZE mode of the LOAD command to generate an optimal map. The map generated by using the ANALYZE mode must be moved to each database partition in your database before the load operation can proceed.

MAP_FILE_OUTPUT X
X represents the output filename for the distribution map. The output file is created on the database partition issuing the LOAD command assuming that database partition is participating in the database partition group where partitioning is performed. If the LOAD command is invoked on a database partition that is not participating in partitioning (as defined by PARTITIONING_DBPARTNUMS), the output file is created at the first database partition defined with the PARTITIONING_DBPARTNUMS parameter. Consider the following partitioned database environment setup:

```
1 serv1 0
2 serv1 1
3 serv2 0
4 serv2 1
5 serv3 0
```

Running the following LOAD command on serv3, creates the distribution map on serv1.

```
LOAD FROM file OF ASC METHOD L ( ...) INSERT INTO table CONFIG
MODE ANALYZE PARTITIONING_DBPARTNUMS(1,2,3,4)
MAP_FILE_OUTPUT '/home/db2user/distribution.map'
```

This parameter should be used when the ANALYZE mode is specified. An optimal distribution map with even distribution across all database partitions is generated. If this parameter is not specified and the ANALYZE mode is specified, the program exits with an error.

TRACE X
Specifies the number of records to trace when you require a review of a dump of the data conversion process and the output of the hashing values. The default is 0.

NEWLINE
Used when the input data file is an ASC file with each record delimited by a new line character and the `reclen` file type modifier is specified in the

LOAD command. When this option is specified, each record is checked for a new line character. The record length, as specified in the `reclen` file type modifier, is also checked.

DISTFILE X
If this option is specified, the load utility generates a database partition distribution file with the given name. The database partition distribution file contains 4096 integers: one for each entry in the target table's distribution map. Each integer in the file represents the number of rows in the input files being loaded that hashed to the corresponding distribution map entry. This information can help you identify skew in your data and also help you decide whether a new distribution map should be generated for the table using the ANALYZE mode of the utility. If this option is not specified, the default behavior of the load utility is to not generate the distribution file.

Note: When this option is specified, a maximum of one partitioning agent is used for the load operation. Even if you explicitly request multiple partitioning agents, only one is used.

OMIT_HEADER
Specifies that a distribution map header should not be included in the distribution file. If not specified, a header is generated.

RUN_STAT_DBPARTNUM X
If the STATISTICS YES parameter is specified in the LOAD command, statistics are collected only on one database partition. This parameter specifies on which database partition to collect statistics. If the value is -1 or not specified at all, statistics are collected on the first database partition in the output database partition list.

Load sessions in a partitioned database environment - CLP examples

The following examples demonstrate loading data in a multi-partition database.

The database has four database partitions numbered 0 through 3. Database WSDB is defined on all of the database partitions, and table TABLE1 resides in the default database partition group which is also defined on all of the database partitions.

Example 1

To load data into TABLE1 from the user data file `load.del` which resides on database partition 0, connect to database partition 0 and then issue the following command:

```
load from load.del of del replace into table1
```

If the load operation is successful, the output will be as follows:

```
Agent Type     Node     SQL Code     Result
_____________________________________________________
LOAD           000      +00000000    Success.
_____________________________________________________
LOAD           001      +00000000    Success.
_____________________________________________________
LOAD           002      +00000000    Success.
_____________________________________________________
LOAD           003      +00000000    Success.
_____________________________________________________
PARTITION      001      +00000000    Success.
_____________________________________________________
```

```
PRE_PARTITION  000      +00000000    Success.
_____________________________________________________________
RESULTS:       4 of 4 LOADs completed successfully.
_____________________________________________________________

Summary of Partitioning Agents:
Rows Read                   = 100000
Rows Rejected               = 0
Rows Partitioned            = 100000

Summary of LOAD Agents:
Number of rows read         = 100000
Number of rows skipped      = 0
Number of rows loaded       = 100000
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 100000
```

The output indicates that there was one load agent on each database partition and each ran successfully. It also shows that there was one pre-partitioning agent running on the coordinator partition and one partitioning agent running on database partition 1. These processes completed successfully with a normal SQL return code of 0. The statistical summary shows that the pre-partitioning agent read 100,000 rows, the partitioning agent distributed 100,000 rows, and the sum of all rows loaded by the load agents is 100,000.

Example 2

In the following example, data is loaded into TABLE1 in the PARTITION_ONLY mode. The distributed output files is stored on each of the output database partitions in the directory /db/data:

```
load from load.del of del replace into table1 partitioned db config mode
   partition_only part_file_location /db/data
```

The output from the load command is as follows:

```
Agent Type     Node     SQL Code     Result
_____________________________________________________________
LOAD_TO_FILE   000      +00000000    Success.
_____________________________________________________________
LOAD_TO_FILE   001      +00000000    Success.
_____________________________________________________________
LOAD_TO_FILE   002      +00000000    Success.
_____________________________________________________________
LOAD_TO_FILE   003      +00000000    Success.
_____________________________________________________________
PARTITION      001      +00000000    Success.
_____________________________________________________________
PRE_PARTITION  000      +00000000    Success.
_____________________________________________________________

Summary of Partitioning Agents:
Rows Read                   = 100000
Rows Rejected               = 0
Rows Partitioned            = 100000
```

The output indicates that there was a load-to-file agent running on each output database partition, and these agents ran successfully. There was a pre-partitioning agent on the coordinator partition, and a partitioning agent running on database partition 1. The statistical summary indicates that 100,000 rows were successfully read by the pre-partitioning agent and 100,000 rows were successfully distributed by the partitioning agent. Since no rows were loaded into the table, no summary of the number of rows loaded appears.

Example 3

To load the files that were generated during the PARTITION_ONLY load operation above, issue the following command:

```
load from load.del of del replace into table1 partitioned db config mode
    load_only part_file_location /db/data
```

The output from the load command will be as follows:

```
Agent Type     Node     SQL Code     Result
_____________________________________________________
LOAD           000      +00000000    Success.
_____________________________________________________
LOAD           001      +00000000    Success.
_____________________________________________________
LOAD           002      +00000000    Success.
_____________________________________________________
LOAD           003      +00000000    Success.
_____________________________________________________
RESULTS:       4 of 4 LOADs completed successfully.
_____________________________________________________

Summary of LOAD Agents:
Number of rows read         = 100000
Number of rows skipped      = 0
Number of rows loaded       = 100000
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 100000
```

The output indicates that the load agents on each output database partition ran successfully and that the sum of the number of rows loaded by all load agents is 100,000. No summary of rows distributed is indicated since distribution was not performed.

Example 4 - Failed Load Operation

If the following LOAD command is issued:

```
load from load.del of del replace into table1
```

and one of the loading database partitions runs out of space in the table space during the load operation, the following output might be returned:

```
SQL0289N  Unable to allocate new pages in table space "DMS4KT".
SQLSTATE=57011

Agent Type     Node     SQL Code     Result
_____________________________________________________________
LOAD           000      +00000000    Success.
_____________________________________________________________
LOAD           001      -00000289    Error. May require RESTART.
_____________________________________________________________
LOAD           002      +00000000    Success.
_____________________________________________________________
LOAD           003      +00000000    Success.
_____________________________________________________________
PARTITION      001      +00000000    Success.
_____________________________________________________________
PRE_PARTITION  000      +00000000    Success.
_____________________________________________________________
RESULTS:       3 of 4 LOADs completed successfully.
_____________________________________________________________

Summary of Partitioning Agents:
```

```
Rows Read                  = 0
Rows Rejected              = 0
Rows Partitioned           = 0

Summary of LOAD Agents:
Number of rows read        = 0
Number of rows skipped     = 0
Number of rows loaded      = 0
Number of rows rejected    = 0
Number of rows deleted     = 0
Number of rows committed   = 0
```

The output indicates that the load operation returned error SQL0289. The database partition summary indicates that database partition 1 ran out of space. If additional space is added to the containers of the table space on database partition 1, the load operation can be restarted as follows:

```
load from load.del of del restart into table1
```

Migration and version compatibility

The **DB2_PARTITIONEDLOAD_DEFAULT** registry variable can be used to revert to pre- DB2® Universal Database™ Version 8 load behavior in a multi-partition database

Note: As of Version 9.5, the DB2_PARTITIONEDLOAD_DEFAULT registry variable is deprecated and may be removed in a later release.

Reverting to the pre- DB2 UDB Version 8 behavior of the LOAD command in a multi-partition database, allows you to load a file with a valid distribution header into a single database partition without specifying any extra partitioned database configuration options. You can do this by setting the value of **DB2_PARTITIONEDLOAD_DEFAULT** to NO. You may choose to use this option if you want to avoid modifying existing scripts that issue the LOAD command against single database partitions. For example, to load a distribution file into database partition 3 of a table that resides in a database partition group with four database partitions, issue the following command:

```
db2set DB2_PARTITIONEDLOAD_DEFAULT=NO
```

Then issue the following commands from the DB2 Command Line Processor:

```
CONNECT RESET

SET CLIENT CONNECT_NODE 3

CONNECT TO DB MYDB

LOAD FROM LOAD.DEL OF DEL REPLACE INTO TABLE1
```

In a multi-partition database, when no multi-partition database load configuration options are specified, the load operation takes place on all the database partitions on which the table is defined. The input file does not require a distribution header, and the MODE option defaults to PARTITION_AND_LOAD. To load a single database partition, the OUTPUT_DBPARTNUMS option must be specified.

Chapter 16. Migration of partitioned database environments

Migrating partitioned databases

Migrating partitioned database environments requires that you install DB2 Version 9.5 in all database partition servers, migrate the instances and then migrate the databases.

You can migrate database partition servers from the catalog database partition server or any other database partition server. Should the migration process fail, you can retry migration from the catalog database partition server or any other database partition server again.

Since a migration of this sort is a significant undertaking, a description of the migration procedure, its prerequisites and restrictions, is beyond the scope of this book. A detailed description is provided in the topic "Migrating partitioned database environments" in the *Migration Guide*, which, in addition, will refer you to numerous other topics to review prior to performing the migration.

© Copyright IBM Corp. 1993, 2009

Chapter 17. Using snapshot and event monitors

Using snapshot monitor data to monitor the reorganization of a partitioned table

The following information describes some of the most useful methods of monitoring the global status of a table reorganization.

There is no separate data group indicating the overall table reorganization status for 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. However, you can deduce the global status of a table reorganization from the values of elements in the individual data partition data group being reorganized. The following information describes some of the most useful methods of monitoring the global status of a table reorganization.

Determining the number of data partitions being reorganized

You can determine the total number of data partitions being reorganized on a table by counting the number of monitor data blocks for table data that have the same table name and schema name. This value indicates the number of data partitions on which reorganization has started. Examples 1 and 2 indicate that three data partitions are being reorganized.

Identifying the data partition being reorganized

You can deduce the current data partition being reorganized from the phase start time (reorg_phase_start). During the SORT/BUILD/REPLACE phase, the monitor data corresponding to the data partition that is being reorganized shows the most recent phase start time. During the INDEX_RECREATE phase, the phase start time is the same for all the data partitions. In Examples 1 and 2, the INDEX_RECREATE phase is indicated, so the start time is the same for all the data partitions.

Identifying an index rebuild requirement

You can determine if an index rebuild is required by obtaining the value of the maximum reorganize phase element (reorg_max_phase), corresponding to any one of the data partitions being reorganized. If reorg_max_phase has a value of 3 or 4, then an Index Rebuild is required. Examples 1 and 2 report a reorg_max_phase value of 3, indicating an index rebuild is required.

The following sample output is from a three-node server that contains a table with three data partitions:

```
CREATE TABLE  sales (c1 INT, c2 INT, c3 INT)
    PARTITION BY RANGE (c1)
        (PART P1 STARTING FROM (1)  ENDING AT (10) IN parttbs,
         PART P2 STARTING FROM (11) ENDING AT (20) IN parttbs,
         PART P3 STARTING FROM (21) ENDING AT (30) IN parttbs)
    DISTRIBUTE BY (c2)
```

Statement executed:

```
REORG TABLE sales ALLOW NO ACCESS ON ALL DBPARTITIONNUMS
```

© Copyright IBM Corp. 1993, 2009

Example 1:

```
GET SNAPSHOT FOR TABLES ON DPARTDB GLOBAL
```

The output is modified to include table information for the relevant table only.

```
           Table Snapshot

First database connect timestamp     = 06/28/2005 13:46:43.061690
Last reset timestamp                 = 06/28/2005 13:46:47.440046
Snapshot timestamp                   = 06/28/2005 13:46:50.964033
Database name                        = DPARTDB
Database path                        = /work/sales/NODE0000/SQL00001/
Input database alias                 = DPARTDB
Number of accessed tables            = 5

Table List
 Table Schema        = NEWTON
 Table Name          = SALES
 Table Type          = User
 Data Partition Id   = 0
 Data Object Pages   = 3
 Rows Read           = 12
 Rows Written        = 1
 Overflows           = 0
 Page Reorgs         = 0
 Table Reorg Information:
   Node number       = 0
   Reorg Type        =
        Reclaiming
        Table Reorg
        Allow No Access
        Recluster Via Table Scan
        Reorg Data Only
   Reorg Index       = 0
   Reorg Tablespace  = 3
 Long Temp space ID  = 3
   Start Time        = 06/28/2005 13:46:49.816883
   Reorg Phase       = 3 - Index Recreate
   Max Phase         = 3
   Phase Start Time  = 06/28/2005 13:46:50.362918
   Status            = Completed
   Current Counter   = 0
   Max Counter       = 0
   Completion        = 0
   End Time          = 06/28/2005 13:46:50.821244

 Table Reorg Information:
   Node number       = 1
   Reorg Type        =
        Reclaiming
        Table Reorg
        Allow No Access
        Recluster Via Table Scan
        Reorg Data Only
   Reorg Index       = 0
   Reorg Tablespace  = 3
 Long Temp space ID  = 3
   Start Time        = 06/28/2005 13:46:49.822701
   Reorg Phase       = 3 - Index Recreate
   Max Phase         = 3
   Phase Start Time  = 06/28/2005 13:46:50.420741
   Status            = Completed
   Current Counter   = 0
   Max Counter       = 0
   Completion        = 0
   End Time          = 06/28/2005 13:46:50.899543
```

```
Table Reorg Information:
  Node number       = 2
  Reorg Type        =
       Reclaiming
       Table Reorg
       Allow No Access
       Recluster Via Table Scan
       Reorg Data Only
  Reorg Index       = 0
  Reorg Tablespace  = 3
Long Temp space ID  = 3
  Start Time        = 06/28/2005 13:46:49.814813
  Reorg Phase       = 3 - Index Recreate
  Max Phase         = 3
  Phase Start Time  = 06/28/2005 13:46:50.344277
  Status            = Completed
  Current Counter   = 0
  Max Counter       = 0
  Completion        = 0
  End Time          = 06/28/2005 13:46:50.803619


Table Schema        = NEWTON
Table Name          = SALES
Table Type          = User
Data Partition Id   = 1
Data Object Pages   = 3
Rows Read           = 8
Rows Written        = 1
Overflows           = 0
Page Reorgs         = 0
Table Reorg Information:
  Node number       = 0
  Reorg Type        =
       Reclaiming
       Table Reorg
       Allow No Access
       Recluster Via Table Scan
       Reorg Data Only
  Reorg Index       = 0
  Reorg Tablespace  = 3
Long Temp space ID  = 3
  Start Time        = 06/28/2005 13:46:50.014617
  Reorg Phase       = 3 - Index Recreate
  Max Phase         = 3
  Phase Start Time  = 06/28/2005 13:46:50.362918
  Status            = Completed
  Current Counter   = 0
  Max Counter       = 0
  Completion        = 0
  End Time          = 06/28/2005 13:46:50.821244

Table Reorg Information:
  Node number       = 1
  Reorg Type        =
       Reclaiming
       Table Reorg
       Allow No Access
       Recluster Via Table Scan
       Reorg Data Only
  Reorg Index       = 0
  Reorg Tablespace  = 3
Long Temp space ID  = 3
  Start Time        = 06/28/2005 13:46:50.026278
  Reorg Phase       = 3 - Index Recreate
```

```
  Max Phase         = 3
  Phase Start Time  = 06/28/2005 13:46:50.420741
  Status            = Completed
  Current Counter   = 0
  Max Counter       = 0
  Completion        = 0
  End Time          = 06/28/2005 13:46:50.899543

Table Reorg Information:
  Node number       = 2
  Reorg Type        =
       Reclaiming
       Table Reorg
       Allow No Access
       Recluster Via Table Scan
       Reorg Data Only
  Reorg Index       = 0
  Reorg Tablespace  = 3
Long Temp space ID  = 3
  Start Time        = 06/28/2005 13:46:50.006392
  Reorg Phase       = 3 - Index Recreate
  Max Phase         = 3
  Phase Start Time  = 06/28/2005 13:46:50.344277
  Status            = Completed
  Current Counter   = 0
  Max Counter       = 0
  Completion        = 0
  End Time          = 06/28/2005 13:46:50.803619


Table Schema        = NEWTON
Table Name          = SALES
Table Type          = User
Data Partition Id   = 2
Data Object Pages   = 3
Rows Read           = 4
Rows Written        = 1
Overflows           = 0
Page Reorgs         = 0
Table Reorg Information:
  Node number       = 0
  Reorg Type        =
       Reclaiming
       Table Reorg
       Allow No Access
       Recluster Via Table Scan
       Reorg Data Only
  Reorg Index       = 0
  Reorg Tablespace  = 3
Long Temp space ID  = 3
  Start Time        = 06/28/2005 13:46:50.199971
  Reorg Phase       = 3 - Index Recreate
  Max Phase         = 3
  Phase Start Time  = 06/28/2005 13:46:50.362918
  Status            = Completed
  Current Counter   = 0
  Max Counter       = 0
  Completion        = 0
  End Time          = 06/28/2005 13:46:50.821244

Table Reorg Information:
  Node number       = 1
  Reorg Type        =
       Reclaiming
       Table Reorg
       Allow No Access
       Recluster Via Table Scan
```

```
       Reorg Data Only
  Reorg Index       = 0
  Reorg Tablespace  = 3
Long Temp space ID  = 3
  Start Time        = 06/28/2005 13:46:50.223742
  Reorg Phase       = 3 - Index Recreate
  Max Phase         = 3
  Phase Start Time  = 06/28/2005 13:46:50.420741
  Status            = Completed
  Current Counter   = 0
  Max Counter       = 0
  Completion        = 0
  End Time          = 06/28/2005 13:46:50.899543

Table Reorg Information:
  Node number       = 2
  Reorg Type        =
       Reclaiming
       Table Reorg
       Allow No Access
       Recluster Via Table Scan
       Reorg Data Only
  Reorg Index       = 0
  Reorg Tablespace  = 3
Long Temp space ID  = 3
  Start Time        = 06/28/2005 13:46:50.179922
  Reorg Phase       = 3 - Index Recreate
  Max Phase         = 3
  Phase Start Time  = 06/28/2005 13:46:50.344277
  Status            = Completed
  Current Counter   = 0
  Max Counter       = 0
  Completion        = 0
  End Time          = 06/28/2005 13:46:50.803619
```

Example 2:

```
GET SNAPSHOT FOR TABLES ON DPARTDB AT DBPARTITIONNUM 2
```

The output is modified to include table information for the relevant table only.

```
           Table Snapshot

First database connect timestamp     = 06/28/2005 13:46:43.617833
Last reset timestamp                 =
Snapshot timestamp                   = 06/28/2005 13:46:51.016787
Database name                        = DPARTDB
Database path                        = /work/sales/NODE0000/SQL00001/
Input database alias                 = DPARTDB
Number of accessed tables            = 3

Table List
 Table Schema        = NEWTON
 Table Name          = SALES
 Table Type          = User
 Data Partition Id   = 0
 Data Object Pages   = 1
 Rows Read           = 0
 Rows Written        = 0
 Overflows           = 0
 Page Reorgs         = 0
 Table Reorg Information:
   Node number       = 2
   Reorg Type        =
        Reclaiming
        Table Reorg
        Allow No Access
```

```
       Recluster Via Table Scan
       Reorg Data Only
  Reorg Index       = 0
  Reorg Tablespace  = 3
Long Temp space ID  = 3
  Start Time        = 06/28/2005 13:46:49.814813
  Reorg Phase       = 3 - Index Recreate
  Max Phase         = 3
  Phase Start Time  = 06/28/2005 13:46:50.344277
  Status            = Completed
  Current Counter   = 0
  Max Counter       = 0
  Completion        = 0
  End Time          = 06/28/2005 13:46:50.803619


Table Schema        = NEWTON
Table Name          = SALES
Table Type          = User
Data Partition Id   = 1
Data Object Pages   = 1
Rows Read           = 0
Rows Written        = 0
Overflows           = 0
Page Reorgs         = 0
Table Reorg Information:
  Node number       = 2
  Reorg Type        =
       Reclaiming
       Table Reorg
       Allow No Access
       Recluster Via Table Scan
       Reorg Data Only
  Reorg Index       = 0
  Reorg Tablespace  = 3
Long Temp space ID  = 3
  Start Time        = 06/28/2005 13:46:50.006392
  Reorg Phase       = 3 - Index Recreate
  Max Phase         = 3
  Phase Start Time  = 06/28/2005 13:46:50.344277
  Status            = Completed
  Current Counter   = 0
  Max Counter       = 0
  Completion        = 0
  End Time          = 06/28/2005 13:46:50.803619


Table Schema        = NEWTON
Table Name          = SALES
Table Type          = User
Data Partition Id   = 2
Data Object Pages   = 1
Rows Read           = 4
Rows Written        = 1
Overflows           = 0
Page Reorgs         = 0
Table Reorg Information:
  Node number       = 2
  Reorg Type        =
       Reclaiming
       Table Reorg
       Allow No Access
       Recluster Via Table Scan
       Reorg Data Only
  Reorg Index       = 0
  Reorg Tablespace  = 3
Long Temp space ID  = 3
```

```
   Start Time        = 06/28/2005 13:46:50.179922
   Reorg Phase       = 3 - Index Recreate
   Max Phase         = 3
   Phase Start Time  = 06/28/2005 13:46:50.344277
   Status            = Completed
   Current Counter   = 0
   Max Counter       = 0
   Completion        = 0
   End Time          = 06/28/2005 13:46:50.803619
```

Example 3:

```
SELECT * FROM SYSIBMADM.SNAPLOCK WHERE tabname = 'SALES';
```

The output is modified to include a subset of table information for the relevant table only.

```
...   TBSP_NAME TABNAME LOCK_OBJECT_TYPE   LOCK_MODE  LOCK_STATUS ...
      --------- ------- ------------------ ---------- -----------
...   PARTTBS   SALES      ROW_LOCK        X          GRNT        ...
...   -         SALES      TABLE_LOCK      IX         GRNT        ...
...   PARTTBS   SALES      TABLE_PART_LOCK IX         GRNT        ...
...   PARTTBS   SALES      ROW_LOCK        X          GRNT        ...
...   -         SALES      TABLE_LOCK      IX         GRNT        ...
...   PARTTBS   SALES      TABLE_PART_LOCK IX         GRNT        ...
...   PARTTBS   SALES      ROW_LOCK        X          GRNT        ...
...   -         SALES      TABLE_LOCK      IX         GRNT        ...
...   PARTTBS   SALES      TABLE_PART_LOCK IX         GRNT        ...

 9 record(s) selected.
```

Output from this query (continued).

```
... LOCK_ESCALATION LOCK_ATTRIBUTES DATA_PARTITION_ID DBPARTITIONNUM
    --------------- --------------- ------------------ --------------
...               0 INSERT          2                  2
...               0 NONE            -                  2
...               0 NONE            2                  2
...               0 INSERT          0                  0
...               0 NONE            -                  0
...               0 NONE            0                  0
...               0 INSERT          1                  1
...               0 NONE            -                  1
...               0 NONE            1                  1
```

Example 4:

```
SELECT * FROM SYSIBMADM.SNAPTAB WHERE tabname = 'SALES';
```

The output is modified to include a subset of table information for the relevant table only.

```
... TABSCHEMA TABNAME TAB_FILE_ID TAB_TYPE   DATA_OBJECT_PAGES ROWS_WRITTEN ...
... --------- ------- ----------- ---------- ----------------- ------------ ...
... NEWTON    SALES             2 USER_TABLE                 1            1 ...
... NEWTON    SALES             4 USER_TABLE                 1            1 ...
... NEWTON    SALES             3 USER_TABLE                 1            1 ...

 3 record(s) selected.
```

Output from this query (continued).

```
... OVERFLOW_ACCESSES PAGE_REORGS DBPARTITIONNUM TBSP_ID DATA_PARTITION_ID
... ----------------- ----------- -------------- ------- -----------------
...                 0           0              0       3                 0
...                 0           0              2       3                 2
...                 0           0              1       3                 1
```

Example 5:

```
SELECT * FROM SYSIBMADM.SNAPTAB_REORG WHERE tabname = 'SALES';;
```

The output is modified to include a subset of table information for the relevant table only.

```
  REORG_PHASE   REORG_MAX_PHASE REORG_TYPE                                      ...
     ------------- -------------- ------------------------------------------
 INDEX_RECREATE              3  RECLAIM+OFFLINE+ALLOW_NONE+TABLESCAN+DATAONLY  ...
 INDEX_RECREATE              3  RECLAIM+OFFLINE+ALLOW_NONE+TABLESCAN+DATAONLY  ...
 INDEX_RECREATE              3  RECLAIM+OFFLINE+ALLOW_NONE+TABLESCAN+DATAONLY  ...
 INDEX_RECREATE              3  RECLAIM+OFFLINE+ALLOW_NONE+TABLESCAN+DATAONLY  ...
 INDEX_RECREATE              3  RECLAIM+OFFLINE+ALLOW_NONE+TABLESCAN+DATAONLY  ...
 INDEX_RECREATE              3  RECLAIM+OFFLINE+ALLOW_NONE+TABLESCAN+DATAONLY  ...
 INDEX_RECREATE              3  RECLAIM+OFFLINE+ALLOW_NONE+TABLESCAN+DATAONLY  ...
 INDEX_RECREATE              3  RECLAIM+OFFLINE+ALLOW_NONE+TABLESCAN+DATAONLY  ...
 INDEX_RECREATE              3  RECLAIM+OFFLINE+ALLOW_NONE+TABLESCAN+DATAONLY  ...

 9 record(s) selected.
```

Output from this query (continued).

```
... REORG_STATUS REORG_TBSPC_ID DBPARTITIONNUM DATA_PARTITION_ID
    ------------------------- -------------- -----------------
... COMPLETED                3              2                 0
... COMPLETED                3              2                 1
... COMPLETED                3              2                 2
... COMPLETED                3              1                 0
... COMPLETED                3              1                 1
... COMPLETED                3              1                 2
... COMPLETED                3              0                 0
... COMPLETED                3              0                 1
... COMPLETED                3              0                 2
```

Global snapshots on partitioned database systems

On a partitioned database system, you can take a snapshot of the current partition, a specified partition, or all partitions. When taking a global snapshot across all the partitions of a partitioned database, data is aggregated before the results are returned.

Data is aggregated for the different element types as follows:

- **Counters, Time, and Gauges**

  Contains the sum of all like values collected from each partition in the instance. For example, GET SNAPSHOT FOR DATABASE XYZ ON TEST GLOBAL would return the number of rows read (rows_read) from the database for all partitions in the partitioned database instance.
- **Watermarks**

  Returns the highest (for high water) or lowest (for low water) value found for any partition in the partitioned database system. If the value returned is of concern, then snapshots for individual partitions can be taken to determine if a particular partition is over utilized, or if the problem is instance-wide.
- **Timestamp**

Set to the timestamp value for the partition where the snapshot monitor instance agent is attached. Note that all timestamp values are under control of the `timestamp` monitor switch.

- **Information**

  Returns the most significant information for a partition that may be impeding work. For example, for the element `appl_status`, if the status on one partition was UOW Executing, and on another partition Lock Wait, Lock Wait would be returned, since it is the state that's holding up execution of the application.

You can also reset counters, set monitor switches, and retrieve monitor switch settings for individual partitions or all partitions in your partitioned database.

Note: When taking a global snapshot, if one or more partitions encounter an error, then data is collected from the partitions where the snapshot was successful and a warning (sqlcode 1629) is also returned. If a global get or update of monitor switches, or a counter reset fails on one or more partitions, then those partitions will not have their switches set, or data reset.

Creating an event monitor for partitioned databases

When creating a file or pipe event monitor on partitioned database systems you need to determine the scope of the monitoring data you wish to collect.

You will need DBADM authority to create event monitors for partitioned databases.

An event monitor uses an operating system process or a thread to write the event records. The database partition where this process or thread runs is called the monitor partition. File and pipe event monitors can be monitoring events as they occur locally on the monitor partition, or globally as they occur on any partition where the DB2 database manager is running. A global event monitor writes a single trace on the monitoring partition that contains activity from all partitions. Whether an event monitor is local or global is referred to as its monitoring scope.

Both the monitor partition and monitor scope are specified with the CREATE EVENT MONITOR statement.

An event monitor can only be activated if the monitor partition is active. If the SET EVENT MONITOR statement is used to activate an event monitor but the monitor partition is not yet active, then event monitor activation will occur when the monitor partition is next started. Furthermore, event monitor activation will automatically occur until the event monitor is explicitly deactivated or the instance is explicitly deactivated. For example, on database partition 0:

```
db2 connect to sample
db2 create event monitor foo ... on dbpartitionnum 2
db2 set event monitor foo state 1
```

After running the above commands, event monitor `foo` will activate automatically whenever the database `sample` activates on database partition 2. This automatic activation occurs until `db2 set event monitor foo state 0` is issued, or partition 2 is stopped.

For write-to-table event monitors, the notion of local or global scope is not applicable. When a write-to-table event monitor is activated, an event monitor process runs on all of the partitions. (More specifically, the event monitor process will run on partitions that belong to the database partition groups in which the

target tables reside.) Each partition where the event monitor process is running also has the same set of target tables. The data in these tables will be different as it represents the individual partition's view of the monitoring data. You can get aggregate values from all the partitions by issuing SQL statements that access the desired values in each partition's event monitor target tables.

The first column of each target table is named PARTITION_KEY, and is used as the partitioning key for the table. The value of this column is chosen so that each event monitor process inserts data into the database partition on which the process is running; that is, insert operations are performed locally on the database partition where the event monitor process is running. On any database partition, the PARTITION_KEY field will contain the same value. This means that if a data partition is dropped and data redistribution is performed, all data on the dropped database partition will go to one other database partition instead of being evenly distributed. Therefore, before removing a database partition, consider deleting all table rows on that database partition.

In addition, a column named PARTITION_NUMBER can be defined for each table. This column contains the number of the partition on which the data was inserted. Unlike the PARTITION_KEY column, the PARTITION_NUMBER column is not mandatory.

The table space within which target tables are defined must exist across all partitions that will have event monitor data written to them. Failure to observe this rule will result in records not being written to the log on partitions (with event monitors) where the table space does not exist. Events will still be written on partitions where the table space does exist, and no error will be returned. This behavior allows users to choose a subset of partitions for monitoring, by creating a table space that exists only on certain partitions.

During write-to-table event monitor activation, the CONTROL table rows for FIRST_CONNECT and EVMON_START are only inserted on the catalog database partition. This requires that the table space for the control table exist on the catalog database partition. If it does not exist on the catalog database partition, these inserts are not performed.

If a partition is not yet active when a write-to-table event monitor is activated, the event monitor will be activated when that partition next activates.

Note: The lock list in the detailed deadlock connection event will only contain the locks held by the application on the partition where it is waiting for the lock. For example, if an application involved in a deadlock is waiting for a lock on node 20, only the locks held by that application on node 20 will be included in the list.

1. Specify the partition to be monitored.

   ```
   CREATE EVENT MONITOR dlmon FOR DEADLOCKS
                       WRITE TO FILE '/tmp/dlevents'
                       ON PARTITION 3
   ```

   `dlmon` represents the name of the event monitor.

   `/tmp/dlevents` is the name of the directory path (on UNIX) where the event monitor is to write the event files.

   `3` represents the partition number to be monitored.
2. Specify if the event monitor data is to be collected at a local or global scope. To collect event monitor reports from all partitions issue the following statement:

```
CREATE EVENT MONITOR dlmon FOR DEADLOCKS
                    WRITE TO FILE '/tmp/dlevents'
                    ON PARTITION 3 GLOBAL
```

Only deadlock and deadlock with details event monitors can be defined as GLOBAL. All partitions will report deadlock-related event records to partition 3.

3. To collect event monitor reports from only the local partition issue the following statement:

```
CREATE EVENT MONITOR dlmon FOR DEADLOCKS
                    WRITE TO FILE '/tmp/dlevents'
                    ON PARTITION 3 LOCAL
```

This is the default behavior for file and pipe event monitors in partitioned databases. The LOCAL and GLOBAL clauses are ignored for write-to-table event monitors.

4. It is possible to review the monitor partition and scope values for existing event monitors. To do this query the SYSCAT.EVENTMONITORS table with the following statement:

```
SELECT EVMONNAME, NODENUM, MONSCOPE FROM SYSCAT.EVENTMONITORS
```

Once an event monitor is created and activated, it will record monitoring data as its specified events occur.

Chapter 18. Developing a good backup and recovery strategy

Crash recovery

Transactions (or units of work) against a database can be interrupted unexpectedly. If a failure occurs before all of the changes that are part of the unit of work are completed and committed, the database is left in an inconsistent and unusable state. *Crash recovery* is the process by which the database is moved back to a consistent and usable state. This is done by rolling back incomplete transactions and completing committed transactions that were still in memory when the crash occurred (Figure 45). When a database is in a consistent and usable state, it has attained what is known as a "point of consistency".

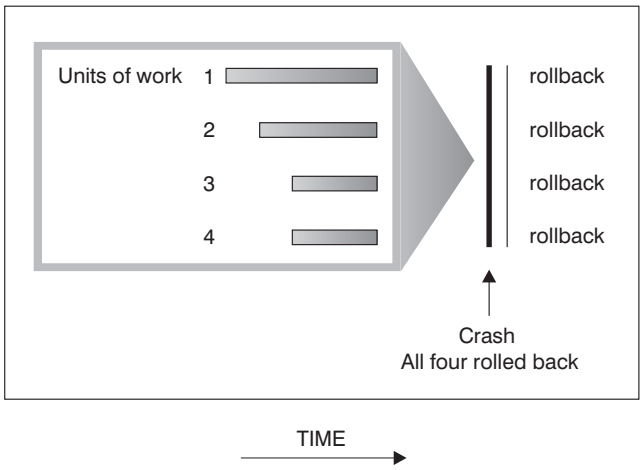


Figure 45. Rolling Back Units of Work (Crash Recovery)

A *transaction failure* results from a severe error or condition that causes the database or the database manager to end abnormally. Partially completed units of work, or UOW that have not been flushed to disk at the time of failure, leave the database in an inconsistent state. Following a transaction failure, the database must be recovered. Conditions that can result in transaction failure include:

- A power failure on the machine, causing the database manager and the database partitions on it to go down
- A hardware failure such as memory corruption, or disk, CPU, or network failure.
- A serious operating system error that causes DB2 to go down
- An application terminating abnormally.

If you want the rollback of incomplete units of work to be done automatically by the database manager, enable the automatic restart (*autorestart*) database configuration parameter by setting it to `ON`. (This is the default value.) If you do not want automatic restart behavior, set the *autorestart* database configuration parameter to `OFF`. As a result, you will need to issue the RESTART DATABASE command when a database failure occurs. If the database I/O was suspended before the crash occurred, you must specify the WRITE RESUME option of the RESTART DATABASE command in order for the crash recovery to continue. The administration notification log records when the database restart operation begins.

© Copyright IBM Corp. 1993, 2009

If crash recovery is applied to a database that is enabled for forward recovery (that is, the *logarchmeth1* configuration parameter is not set to OFF), and an error occurs during crash recovery that is attributable to an individual table space, that table space will be taken offline, and cannot be accessed until it is repaired. Crash recovery continues. At the completion of crash recovery, the other table spaces in the database will be accessible, and connections to the database can be established. However, if the table space that is taken offline is the table space that contains the system catalogs, it must be repaired before any connections will be permitted.

Recovering from transaction failures in a partitioned database environment

If a transaction failure occurs in a partitioned database environment, database recovery is usually necessary on both the failed database partition server and any other database partition server that was participating in the transaction:

- Crash recovery occurs on the failed database partition server after the antecedent condition is corrected.
- *Database partition failure recovery* on the other (still active) database partition servers occurs immediately after the failure has been detected.

In a partitioned database environment, the database partition server on which an application is submitted is the coordinator partition, and the first agent that works for the application is the coordinator agent. The coordinator agent is responsible for distributing work to other database partition servers, and it keeps track of which ones are involved in the transaction. When the application issues a COMMIT statement for a transaction, the coordinator agent commits the transaction by using the two-phase commit protocol. During the first phase, the coordinator partition distributes a PREPARE request to all the other database partition servers that are participating in the transaction. These servers then respond with one of the following:

READ-ONLY
: No data change occurred at this server

YES
: Data change occurred at this server

NO
: Because of an error, the server is not prepared to commit

If one of the servers responds with a NO, the transaction is rolled back. Otherwise, the coordinator partition begins the second phase.

During the second phase, the coordinator partition writes a COMMIT log record, then distributes a COMMIT request to all the servers that responded with a YES. After all the other database partition servers have committed, they send an acknowledgement of the COMMIT to the coordinator partition. The transaction is complete when the coordinator agent has received all COMMIT acknowledgments from all the participating servers. At this point, the coordinator agent writes a FORGET log record.

Transaction failure recovery on an active database partition server

If any database partition server detects that another server is down, all work that is associated with the failed database partition server is stopped:

- If the still active database partition server is the coordinator partition for an application, and the application was running on the failed database partition

server (and not ready to COMMIT), the coordinator agent is interrupted to do failure recovery. If the coordinator agent is in the second phase of COMMIT processing, SQL0279N is returned to the application, which in turn loses its database connection. Otherwise, the coordinator agent distributes a ROLLBACK request to all other servers participating in the transaction, and SQL1229N is returned to the application.

- If the failed database partition server was the coordinator partition for the application, then agents that are still working for the application on the active servers are interrupted to do failure recovery. The transaction is rolled back locally on each database partition where the transaction is not in prepared state. On those database partitions where the transaction is in prepared state, the transaction becomes indoubt. The coordinator database partition is not aware that the transaction is indoubt on some database partitions because the coordinator database partition is not available.
- If the application connected to the failed database partition server (before it failed), but neither the local database partition server nor the failed database partition server is the coordinator partition, agents working for this application are interrupted. The coordinator partition will either send a ROLLBACK or a disconnect message to the other database partition servers. The transaction will only be indoubt on database partition servers that are still active if the coordinator partition returns SQL0279.

Any process (such as an agent or deadlock detector) that attempts to send a request to the failed server is informed that it cannot send the request.

Transaction failure recovery on the failed database partition server

If the transaction failure causes the database manager to end abnormally, you can issue the db2start command with the RESTART option to restart the database manager once the database partition has been restarted. If you cannot restart the database partition, you can issue db2start to restart the database manager on a different database partition.

If the database manager ends abnormally, database partitions on the server can be left in an inconsistent state. To make them usable, crash recovery can be triggered on a database partition server:

- Explicitly, through the RESTART DATABASE command
- Implicitly, through a CONNECT request when the *autorestart* database configuration parameter has been set to `ON`

Crash recovery reapplies the log records in the active log files to ensure that the effects of all complete transactions are in the database. After the changes have been reapplied, all uncommitted transactions are rolled back locally, *except* for indoubt transactions. There are two types of indoubt transaction in a partitioned database environment:

- On a database partition server that is not the coordinator partition, a transaction is in doubt if it is prepared but not yet committed.
- On the coordinator partition, a transaction is in doubt if it is committed but not yet logged as complete (that is, the FORGET record is not yet written). This situation occurs when the coordinator agent has not received all the COMMIT acknowledgments from all the servers that worked for the application.

Crash recovery attempts to resolve all the indoubt transactions by doing one of the following. The action that is taken depends on whether the database partition server was the coordinator partition for an application:

- If the server that restarted is not the coordinator partition for the application, it sends a query message to the coordinator agent to discover the outcome of the transaction.
- If the server that restarted *is* the coordinator partition for the application, it sends a message to all the other agents (subordinate agents) that the coordinator agent is still waiting for COMMIT acknowledgments.

It is possible that crash recovery might not be able to resolve all the indoubt transactions. For example, some of the database partition servers might not be available. If the coordinator partition completes crash recovery before other database partitions involved in the transaction, crash recovery will not be able to resolve the indoubt transaction. This is expected because crash recovery is performed by each database partition independently. In this situation, the SQL warning message SQL1061W is returned. Because indoubt transactions hold resources, such as locks and active log space, it is possible to get to a point where no changes can be made to the database because the active log space is being held up by indoubt transactions. For this reason, you should determine whether indoubt transactions remain after crash recovery, and recover all database partition servers that are required to resolve the indoubt transactions as quickly as possible.

Note: In a partitioned database server environment, the RESTART database command is run on a per-node basis. In order to ensure that the database is restarted on all nodes, use the following recommended command:

```
db2_all "db2 restart database <database_name>"
```

If one or more servers that are required to resolve an indoubt transaction cannot be recovered in time, and access is required to database partitions on other servers, you can manually resolve the indoubt transaction by making an heuristic decision. You can use the LIST INDOUBT TRANSACTIONS command to query, commit, and roll back the indoubt transaction on the server.

Note: The LIST INDOUBT TRANSACTIONS command is also used in a distributed transaction environment. To distinguish between the two types of indoubt transactions, the *originator* field in the output that is returned by the LIST INDOUBT TRANSACTIONS command displays one of the following:

- DB2 Enterprise Server Edition, which indicates that the transaction originated in a partitioned database environment.
- XA, which indicates that the transaction originated in a distributed environment.

Identifying the failed database partition server

When a database partition server fails, the application will typically receive one of the following SQLCODEs. The method for detecting which database manager failed depends on the SQLCODE received:

SQL0279N
: This SQLCODE is received when a database partition server involved in a transaction is terminated during COMMIT processing.

SQL1224N
: This SQLCODE is received when the database partition server that failed is the coordinator partition for the transaction.

SQL1229N
This SQLCODE is received when the database partition server that failed is not the coordinator partition for the transaction.

Determining which database partition server failed is a two-step process. The SQLCA associated with SQLCODE SQL1229N contains the node number of the server that detected the error in the sixth array position of the *sqlerrd* field. (The node number that is written for the server corresponds to the node number in the db2nodes.cfg file.) On the database partition server that detects the error, a message that indicates the node number of the failed server is written to the administration notification log.

Note: If multiple logical nodes are being used on a processor, the failure of one logical node can cause other logical nodes on the same processor to fail.

Recovering from the failure of a database partition server

To recover from the failure of a database partition server, perform the following steps.

1. Correct the problem that caused the failure.
2. Restart the database manager by issuing the db2start command from any database partition server.
3. Restart the database by issuing the RESTART DATABASE command on the failed database partition server or servers.

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.

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 BKxy 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.

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
   ```

At this point the database is connectable on all database partitions and all table spaces are in NORMAL state.

Recovering data using db2adutl

The examples that follow show how to perform cross-node recovery using the db2adutl command, and the *logarchopt1* and *vendoropt* database configuration parameters.

For the following examples, computer 1 is called `bar` and is running AIX. The owner of this machine is `roecken`. The database on `bar` is called `zample`. Computer 2 is called `dps`. This machine is also running AIX, and is owned by `regress9`.

PASSWORDACCESS = generate

Computer 1

1. Set up the database for log archiving to TSM. Update the database configuration parameter *logarchmeth1* for the `zample` database:

```
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.
```

Note: Before updating the database configuration, you might have to take an offline backup of the database.

2. Force off applications:

   ```
   db2 force applications all
   ```

3. Verify that all applications have been forced off:

   ```
   db2 list applications
   ```

   You should receive a message that says no data was returned.

   Note: In a partitioned database environment, you must perform this step on all database partitions.

4. Take a backup of the database:

   ```
   db2 backup db zample use tsm
   ```

   Information similar to the following is returned:

   ```
   Backup successful. The timestamp for this backup image is : 20040216151025
   ```

   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 "Using backup" on page 413.

5. Connect to the `zample` database, then create a table in it.

6. Load data into the new table. In this example, the table is called `a`, 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 Tivoli Storage Manager.

   Note: You can only specify the COPY YES option if the database is enabled for rollforward recovery; that is, the *logarchmeth1* database configuration parameter must be set to either USEREXIT or LOGRETAIN.

   ```
   bar:/home/roecken> db2 load from mr of del modified by noheader replace
      into a copy yes use tsm
   ```

   The utility returns a series of messages to indicate its progress:

   ```
   SQL3109N  The utility is beginning to load data from file "/home/roecken/mr".

   SQL3500W  The utility is beginning the "LOAD" phase at time "02/16/2004
   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 "02/16/2004
   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
```

There should now be one backup image, one load copy image and one log file on TSM. A query on the zample database can be run as follows:

```
bar:/home/roecken/sqllib/adsm> db2adutl query db zample
```

The following information is returned:

```
Retrieving FULL DATABASE BACKUP information.
    1 Time: 20040216151025  Oldest log: S0000000.LOG  DB Partition Number: 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: 20040216151213


Retrieving LOG ARCHIVE information.
   Log file: S0000000.LOG, Chain Num: 0, DB Partition Number: 0,
   Taken at: 2004-02-16-15.10.38
```

7. To enable cross-node recovery, another node and account must be given access to the objects on the bar computer. In this example, access is given to the node dps and the user regress9.

   ```
   bar:/home/roecken/sqllib/adsm> db2adutl 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.
   ```

   To query the results of the db2adutl grant operation, issue the following command:

   ```
   bar:/home/roecken/sqllib/adsm> db2adutl queryaccess
   ```

   The following information is returned:

   ```
   Node                Username            Database Name   Type
   ------------------------------------------------------------
   DPS                 regress9            ZAMPLE          A
   ------------------------------------------------------------
    Access Types:  B - backup images   L - logs   A - both
   ```

PASSWORDACCESS = generate environment

Computer 2

Computer 2, dps, is not yet set up. A db2adutl query on dps for the zample database returns the following results:

```
dps:/home/regress9/sqllib/adsm> db2adutl query db zample
--- 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.


dps:/home/regress9/sqllib/adsm> db2adutl query db zample nodename
    bar owner roecken
--- Database directory is empty ---

Query for database ZAMPLE


Retrieving FULL DATABASE BACKUP information.
    1 Time: 20040216151025  Oldest log: S0000000.LOG  DB Partition Number: 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: 20040216151213


Retrieving LOG ARCHIVE information.
   Log file: S0000000.LOG, Chain Num: 0, DB Partition Number: 0,
   Taken at: 2004-02-16-15.10.38
```

The zample database does not yet exist on the dps computer.

1. Restore the zample database to the dps computer:

   ```
   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.

A rollforward operation on the zample database will fail because the rollforward utility cannot find the log files. A rollforward operation such as the following:

```
dps:/home/regress9> db2 rollforward db zample to end of logs and stop
```

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".
```

2. To force the rollforward utility to look for log files on another machine, you must configure the proper *logarchopt* value, in this situation the *logarchopt1* database configuration parameter:

   ```
   dps:/home/regress9> db2 update db cfg for zample using logarchopt1
   "'-fromnode=bar -fromowner=roecken'"
   ```

3. For the rollforward utility to be able to use the load copy images, you must also set the *vendoropt* database configuration parameter:

   ```
   dps:/home/regress9> db2 update db cfg for zample using VENDOROPT
   "'-fromnode=bar -fromowner=roecken'"
   ```

4. The zample database can now be rolled forward::

   ```
   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 nodes have returned status   = 1

Node number                            = 0
Rollforward status                     = not pending
Next log file to be read               =
Log files processed                    = S0000000.LOG - S0000000.LOG
Last committed transaction             = 2004-02-16-20.10.38.000000 UTC

DB20000I  The ROLLFORWARD command completed successfully.
```

PASSWORDACCESS = prompt environment

In a PROMPT environment, extra information is required, specifically the TSM nodename and password of the machine where the objects were created.

For db2adutl, update the dsm.sys file (called the *dsm.opt* file on Windows-based platforms) and add NODENAME bar (because bar is the name of the source computer) to the server clause:

```
dps:/home/regress9/sqllib/adsm> db2adutl query db zample nodename bar
owner roecken password *******
```

The following information is returned:

```
Query for database ZAMPLE


Retrieving FULL DATABASE BACKUP information.
    1 Time: 20040216151025  Oldest log: S0000000.LOG  DB Partition Number: 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: 20040216151213


Retrieving LOG ARCHIVE information.
   Log file: S0000000.LOG, Chain Num: 0, DB Partition Number: 0,
   Taken at: 2004-02-16-15.10.38
```

1. If the database does not exist, create an empty `zample` database. If the `zample` database already exists, this step, and the next two steps that update the database configuration, can be skipped.

   ```
   dps:/home/regress9> db2 create db zample
   ```

2. Update the database configuration parameter *tsm_nodename* for the `zample` database:

   ```
   dps:/home/regress9> db2 update db cfg for zample using tsm_nodename bar
   ```

3. Update the database configuration parameter *tsm_password* for the `zample` database:

   ```
   dps:/home/regress9> db2 update db cfg for zample using
   tsm_password ********
   ```

4. Restore the `zample` database:

   ```
   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.
   ```

   Again, at this point, the rollforward utility cannot find the correct log files:

   ```
   dps:/home/regress9> db2 rollforward db zample to end of logs and stop
   ```

   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".
   ```

5. Because the database restore operation replaces the database configuration file, the TSM database configuration values must be set to the correct values. First the *tsm_nodename* configuration parameter must be reset:

   ```
   dps:/home/regress9> db2 update db cfg for zample using tsm_nodename bar
   ```

6. The *tsm_password* database configuration parameter must be reset:

   ```
   dps:/home/regress9> db2 update db cfg for zample using tsm_password *******
   ```

7. The *logarchopt1* database configuration parameter must be reset so the rollforward utility can find the correct log files:

   ```
   dps:/home/regress9> db2 update db cfg for zample using logarchopt1
   "'-fromnode=bar -fromowner=roecken'"
   ```

8. The *vendoropt* database configuration parameter must also be reset so that the load recovery file can also be used:

```
dps:/home/regress9> db2 update db cfg for zample using VENDOROPT
"'-fromnode=bar -fromowner=roecken'"
```

9. When the database configuration parameters are set, the database can be rolled forward:

```
dps:/home/regress9> db2 rollforward db zample to end of logs and stop
```

A ROLLFORWARD QUERY STATUS command on the `zample` database shows the following:

```
                                 Rollforward Status

 Input database alias                   = zample
 Number of nodes have returned status   = 1

 Node number                            = 0
 Rollforward status                     = not pending
 Next log file to be read               =
 Log files processed                    = S0000000.LOG - S0000000.LOG
 Last committed transaction             = 2004-02-16-20.10.38.000000 UTC

DB20000I  The ROLLFORWARD command completed successfully.
```

Synchronizing clocks in a partitioned database environment

You should maintain relatively synchronized system clocks across the database partition servers to ensure smooth database operations and unlimited forward recoverability. Time differences among the database partition servers, plus any potential operational and communications delays for a transaction should be less than the value specified for the *max_time_diff* (maximum time difference among nodes) database manager configuration parameter.

To ensure that the log record time stamps reflect the sequence of transactions in a partitioned database environment, DB2 uses the system clock on each machine as the basis for the time stamps in the log records. If, however, the system clock is set ahead, the log clock is automatically set ahead with it. Although the system clock can be set back, the clock for the logs cannot, and remains at the *same* advanced time until the system clock matches this time. The clocks are then in synchrony. The implication of this is that a short term system clock error on a database node can have a long lasting effect on the time stamps of database logs.

For example, assume that the system clock on database partition server A is mistakenly set to November 7, 2005 when the year is 2003, and assume that the mistake is corrected *after* an update transaction is committed in the database partition at that database partition server. If the database is in continual use, and is regularly updated over time, any point between November 7, 2003 and November 7, 2005 is virtually unreachable through rollforward recovery. When the COMMIT on database partition server A completes, the time stamp in the database log is set to 2005, and the log clock remains at November 7, 2005 until the system clock matches this time. If you attempt to roll forward to a point in time within this time frame, the operation will stop at the first time stamp that is beyond the specified stop point, which is November 7, 2003.

Although DB2 cannot control updates to the system clock, the *max_time_diff* database manager configuration parameter reduces the chances of this type of problem occurring:

- The configurable values for this parameter range from 1 minute to 24 hours.

- When the first connection request is made to a non-catalog partition, the database partition server sends its time to the catalog partition for the database. The catalog partition then checks that the time on the database partition requesting the connection, and its own time are within the range specified by the *max_time_diff* parameter. If this range is exceeded, the connection is refused.
- An update transaction that involves more than two database partition servers in the database must verify that the clocks on the participating database partition servers are in synchrony before the update can be committed. If two or more database partition servers have a time difference that exceeds the limit allowed by *max_time_diff*, the transaction is rolled back to prevent the incorrect time from being propagated to other database partition servers.

Chapter 19. Troubleshooting

Troubleshooting DB2

In general, troubleshooting requires that you isolate and identify a problem, then seek a resolution. This section will provide troubleshooting information related to specific features of DB2 products.

As common problems are identified, the findings will be added to this section in the form of checklists. If the checklist does not lead you to a resolution, you can collect additional diagnostic data and analyze it yourself, or submit the data to IBM Software Support for analysis.

The following questions direct you to appropriate troubleshooting tasks:

1. Have you applied all known fix packs? If not, consider "Applying fix packs" in *Quick Beginnings for DB2 Servers*.
2. Does the problem occur when you are:
   - Installing DB2 database servers or clients? If so, see the topic "Collect data for installation problems" elsewhere in this book.
   - Creating, dropping, updating or migrating an instance or the DB2 Administration Server (DAS)? If so, see the topic "Collect data for DAS and instance management problems" elsewhere in this book..
   - Moving data using EXPORT, IMPORT, LOAD or db2move commands? If so, see the topic "Collect data for data movement problems" elsewhere in this book.

If your problem does not fall into one of these categories, basic diagnostic data might still be required if you are contacting IBM Software Support. You should .see the topic "Collect data for DB2" elsewhere in this book.

Troubleshooting partitioned database environments

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 (nodes). You can do so using the rah command or the db2_all 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_all command. This section provides an overview of these commands. The information that follows applies to partitioned database environments only.

Note:

1. On Linux and UNIX platforms, 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.
2. Also, on Linux and UNIX platforms, 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). The ssh remote shell program is used to prevent the transmission of

© Copyright IBM Corp. 1993, 2009

passwords in clear text in UNIX operating system environments. If this registry variable is not set, rsh (or remsh for HP-UX) is used.

3. On Windows, to run the rah command or the db2_all command, you must be logged on with a user account that is a member of the Administrators group.

To determine the scope of a command, refer to the *Command Reference*, which indicates whether a command runs on a single database partition server, or on all of them. If the command runs on one database partition server and you want it to run on all of them, use db2_all. The exception is the db2trc command, which runs on all the logical nodes (database partition servers) on a computer. If you want to run db2trc on all logical nodes on all computers, use rah.

Part 4. Performance issues

© Copyright IBM Corp. 1993, 2009

Chapter 20. Performance issues in database design

Performance enhancing features

Table partitioning and multidimensional clustering tables

A table can be both multi-dimensional clustered and partitioned. In a table that is both multi-dimensional clustered and partitioned, columns can be used both in the table partitioning range-partition-spec and in the MDC key. This is useful for achieving 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 multi-column, while MDC is multi-dimension.

Characteristics of a mainstream DB2 V9.1 data warehouse

The following recommendations were focused on typical, mainstream warehouses that are new for DB2 V9.1. The following characteristics are assumed:

- The database runs on multiple machines or multiple AIX logical partitions.
- Database partitioning feature (DPF) is 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 million to 100 billion 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.

© Copyright IBM Corp. 1993, 2009

- 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 14. Using table partitioning with MDC tables

Issue	Recommended scheme	Recommendation
Data availability during roll-out	Table partitioning	Use the DETACH PARTITION clause to roll out large amounts of data with minimal disruption.
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 (either ALLOW READ ACCESS or ALLOW NO ACCESS)	Table partitioning and MDC	MDC provides most of the benefit here. Table partitioning provides incremental benefits.
Load data ″continually″ (ALLOW READ ACCESS)	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.

Table 14. Using table partitioning with MDC tables (continued)

Issue	Recommended scheme	Recommendation
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, range partitioning helps reduce the need for reorg by maintaining some course 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 6 on page 32.

```
CREATE TABLE orders (YearAndMonth INT, Province CHAR(2))
PARTITION BY RANGE (YearAndMonth)
(STARTING 9901 ENDING 9904 EVERY 2)
ORGANIZE BY (Province);
```

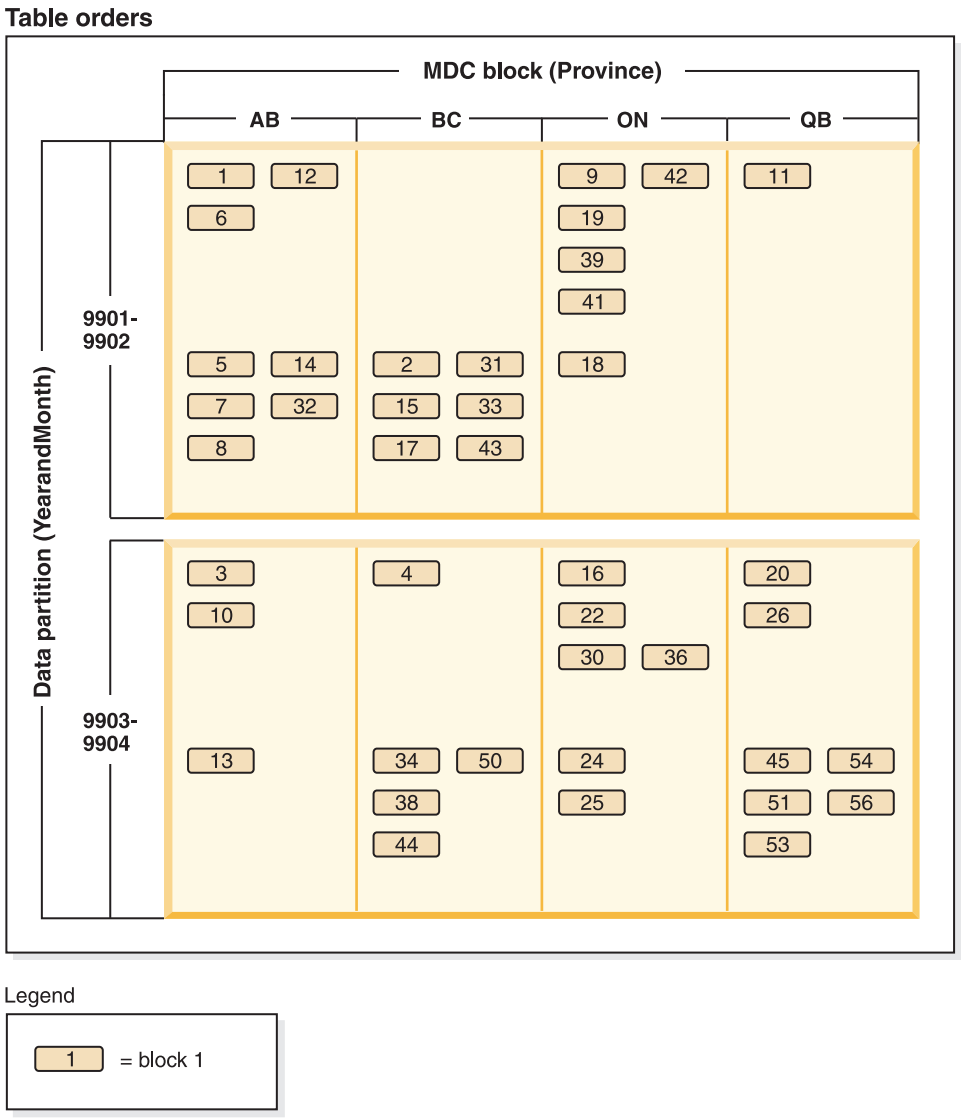


Figure 46. A table partitioned by YearAndMonth and organized by Province

Example 2:

Finer granularity can be achieved by adding YearAndMonth to the ORGANIZE BY clause, as shown in Figure 7 on page 33.

```
CREATE TABLE orders (YearAndMonth INT, Province CHAR(2))
PARTITION BY RANGE (YearAndMonth)
(STARTING 9901 ENDING 9904 EVERY 2)
ORGANIZE BY (YearAndMonth, Province);
```

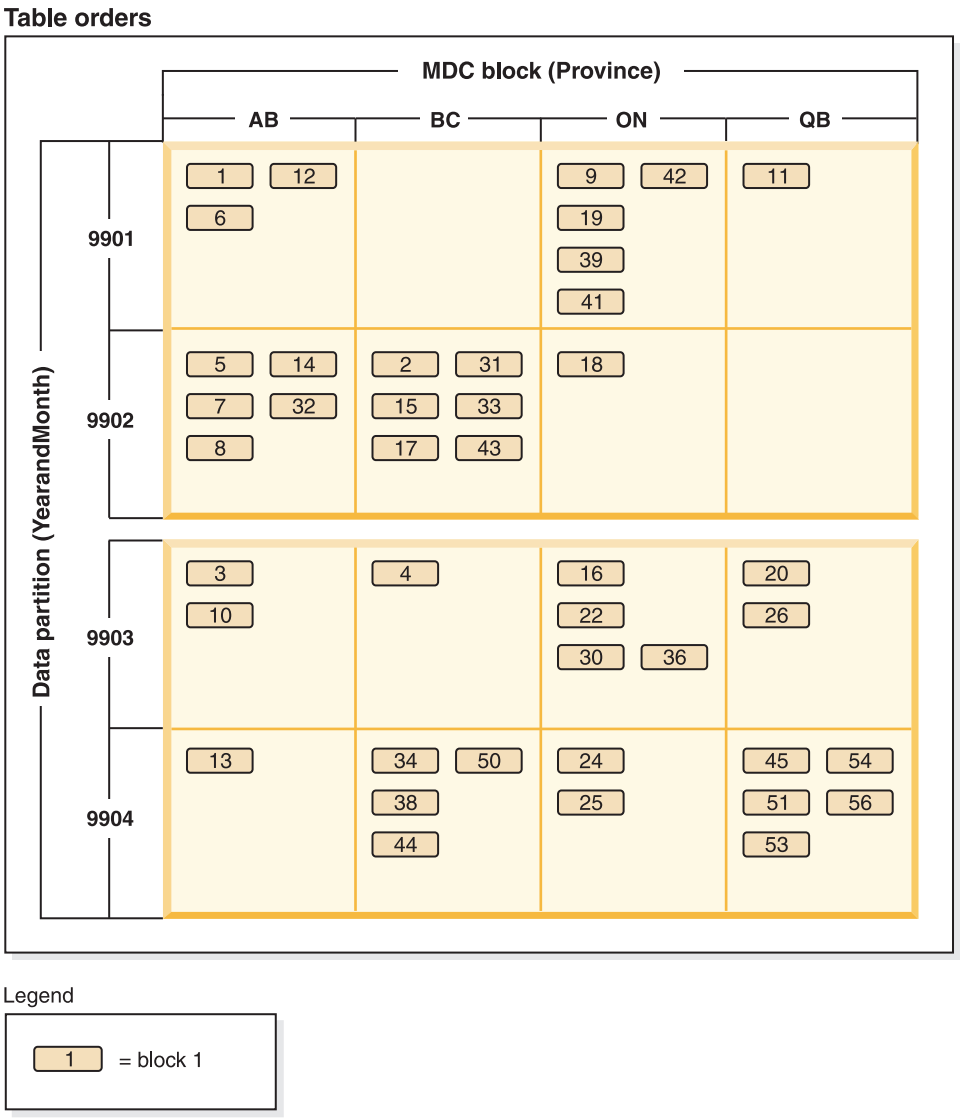


Figure 47. A table partitioned by YearAndMonth and organized by Province and YearAndMonth

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.

Optimization strategies for partitioned tables

Data partition elimination refers to the database servers ability to determine, based on the query predicates, that only a subset of the data partitions of a table need to

be accessed to answer a query. Data partition elimination offers particular benefit 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 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.

The following example demonstrates the performance benefits of data partition elimination. If you issue the following statement:

```
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 you are interested in customer information for the year 2000. If you issue the following query:

```
SELECT * FROM custlist WHERE subsdate BETWEEN '1/1/2000' AND '12/31/2000';
```

As Figure 48 shows, the database server determines that only one data partition in table space 4 (ts4) must be accessed to resolve this query.

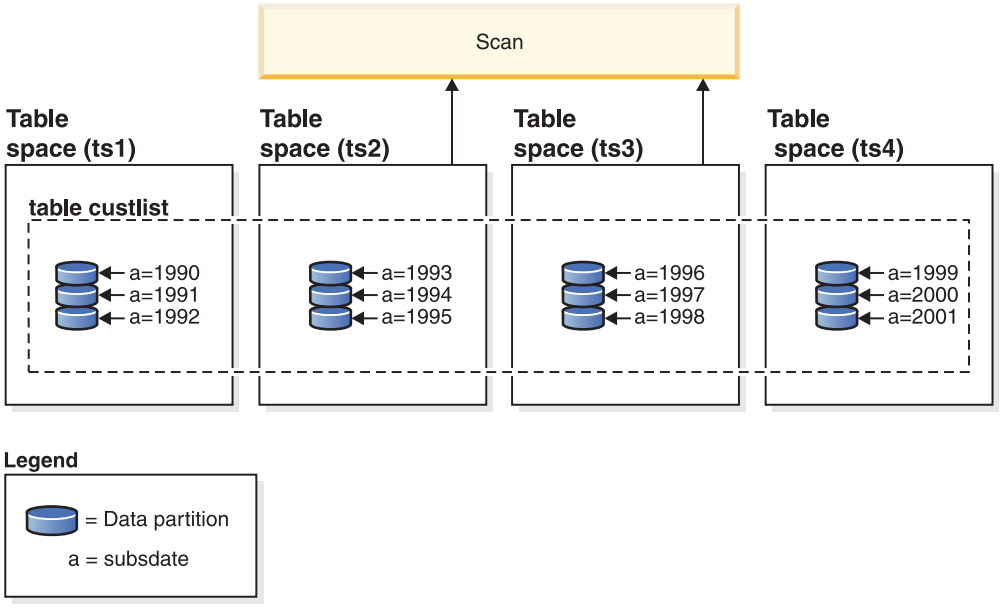


Figure 48. The performance benefits of data partition elimination on a partitioned table

Another example of data partition elimination shown in figure Figure 49 on page 275, is an index scan involving two indexes and 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);
```

If you issue the following query:

```
SELECT * FROM multi WHERE
sale_date BETWEEN '6/1/2005' AND '7/31/2005' AND REGION = 'NW';
```

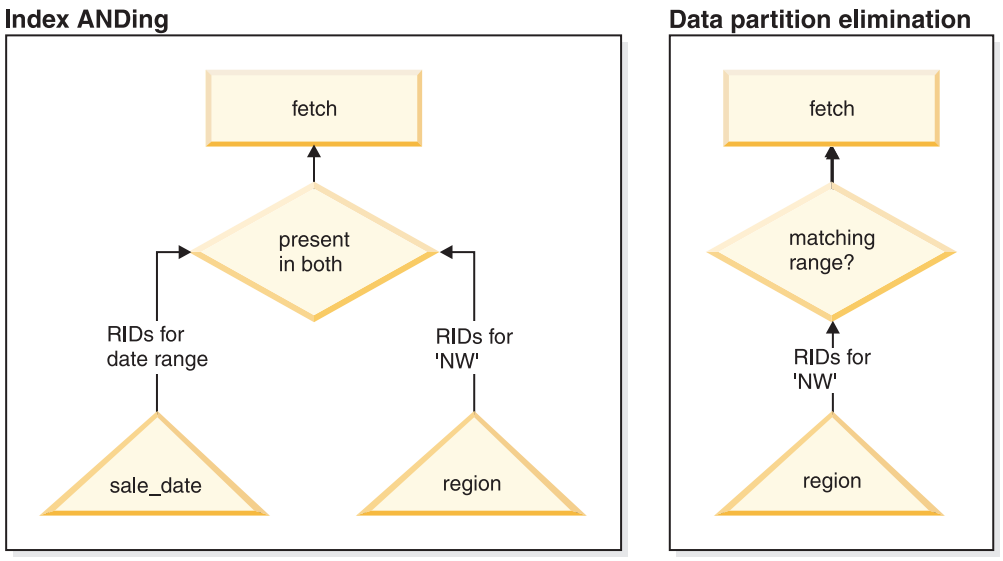


Figure 49. Optimizer decision path for both table partitioning and index ANDing

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 49 demonstrates, with table partitioning, the index is read to find matches for both region and sale_date, allowing for fast retrieval of matching rows.

DB2 Explain

You can also use DB2 Explain to determine the partition elimination chosen by the DB2 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 for cases where multiple columns are used as the table partitioning key.

For example, if you issue the following statement:

```
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)
```

Next, issue the following query:

```
 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, since the non-leading columns used for the table partitioning key are not independent.

Multi-range support

It is possible to achieve data partition elimination on data partitions with multiple ranges (that is, OR'ed together). Using the table 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, you can issue the following statement:

```
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 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, if you have the following query:

```
 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 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)
```

Predicates 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 runtime when the necessary values are bound.

During runtime when T1 is the inner of a NLJN, data partition elimination occurs dynamically, based on the predicates, for every outer value of T2.A. During runtime 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 spaces ts6 and ts7 to be accessed.

Consider that column A in tables T1 and T2 have the following values:

Outer table T2: column A	Inner table T1: column A	Inner table T1: column B	Inner table T1: data 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 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 spaces ts4 and ts5.
4. Scans the data partitions in table spaces ts4 and ts5 of table T1 until a row is found after applying T1.A = 2 and T1.B > 15. The first row found that qualifies is row 3 of T1.
5. Joins the matching row.
6. Scans the data partitions in table spaces ts4 and ts5 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 next row (replacing the value of A with 3) of T2 until all the rows from T2 are exhausted.

Optimization strategies for MDC tables

If you create multidimensional clustering (MDC) tables, the performance of many queries might improve because the optimizer can apply additional optimization strategies. These strategies are primarily based on the improved efficiency of block indexes, but the advantage of clustering on more than one dimension also permits faster data retrieval.

Note: MDC table optimization strategies can also implement the performance advantages of intra-partition parallelism and inter-partition parallelism.

Consider the following specific advantages of MDC tables:

- Dimension block index lookups can identify the required portions of the table and quickly scan only the required blocks.
- Because block indexes are smaller than RID indexes, lookups are faster.
- Index ANDing and ORing can be performed at the block level and combined with RIDs.
- Data is guaranteed to be clustered on extents, which makes retrieval faster.
- Rows can be deleted faster when rollout can be used.

Consider the following simple example for an MDC table named SALES with dimensions defined on the **region** and **month** columns:

```
SELECT * FROM SALES
    WHERE MONTH='March' AND REGION='SE'
```

For this query, the optimizer can perform a dimension block index lookup to find blocks in which the month of March and the SE region occur. Then it can quickly scan only the resulting blocks of the table to fetch the result set.

Rollout deletion

When conditions are met to allow delete using rollout, a more efficient way to delete rows from MDC tables is used. The conditions are:

- The DELETE statement is searched, not positioned (that is, does not use the "WHERE CURRENT OF" clause).
- No WHERE clause (all rows are to be deleted) or the only conditions in the WHERE clause are on dimensions.
- The table is not defined with the DATA CAPTURE CHANGES clause.
- The table is not the parent in a referential integrity relationship.
- The table does not have on delete triggers defined.
- The table is not used in any MQTs that are refreshed immediately.
- A cascaded delete operation may qualify for rollout, if its foreign key is a subset of its table's dimension columns.
- The DELETE statement cannot appear in a SELECT statement executing against the temporary table that identifies the set of affected rows prior to a triggering SQL operation (specified by the OLD TABLE AS clause on the CREATE TRIGGER statement).

For a rollout deletion, 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.

The default behavior, *immediate cleanup rollout*, is to clean up RID indexes at delete time. This mode can also be specified by setting the **DB2_MDC_ROLLOUT** registry variable to IMMEDIATE or by specifying `IMMEDIATE` with the SET CURRENT MDC ROLLOUT MODE statement. There is no change in the logging of index updates, as compared to a standard delete, so the performance improvement depends on how many RID indexes there are. The fewer RID indexes, the better the improvement is, as a percentage of the total time and log space.

An estimate of the amount of space saved in the log can be made with this formula, where `N` is the number of records deleted, `S` is total size of the records deleted, including overhead such as null indicators and varchar lengths, and `P` is the number of pages in the blocks containing the records deleted:

```
S + 38*N - 50*P
```

This figure is the reduction in actual log data. The saving on active log space requirement is double due to saving for space reserved for rollback.

Alternatively, you can have the RID indexes updated after the transaction commits, using *deferred cleanup rollout*. This mode can also be specified by setting the **DB2_MDC_ROLL_OUT** registry variable to DEFER or by specifying `DEFERRED` with the SET CURRENT MDC ROLLOUT MODE statement. In a deferred rollout, RID indexes are cleaned up asynchronously in the background after the commit of the delete. This method of rollout can result in significantly faster deletion times for very large deletes or when a number of RID indexes exist on a table. The speed of the overall cleanup operation is increased because during a deferred index

cleanup, the indexes are cleaned up in parallel, whereas in an immediate index cleanup, each row in the index is cleaned up one by one. As well, the transactional log space requirement for the DELETE statement is significantly reduced because the asynchronous index cleanup logs the index updates by index page instead of by index key.

Note: Deferred cleanup rollout requires additional memory resources, which are taken from the database heap. If DB2 is unable to allocate the memory structures it requires, the deferred cleanup rollout fails and a message is written to the administrator log.

When to use deferred cleanup rollout

If delete performance is the most important factor to you, and there are RID indexes defined on the table, use deferred cleanup rollout. Note that prior to index cleanup, index-based scans of the rolled out blocks suffer a small performance penalty, depending on the amount of rolled out data. Here are other issues to consider when deciding between immediate index cleanup and deferred index cleanup:

- Size of delete: Choose deferred cleanup rollout for very large deletes. In cases where dimensional delete statements are frequently issued on many small MDC tables, the overhead to asynchronously clean index objects might outweigh the benefit of the time saved during the delete.
- Number and type of indexes: If the table contains a number of RID indexes, which require row-level processing, use deferred cleanup rollout.
- Block availability: If you want the block space freed by the delete statement to be available immediately after the delete statement commits, use immediate cleanup rollout.
- Log space: If log space is limited, use deferred cleanup rollout for large deletions.
- Memory constraints: Deferred cleanup rollout consumes additional database heap on all tables which have deferred cleanup pending.

To disable rollout behavior in deletions, you can set the **DB2_MDC_ROLLOUT** registry variable to OFFor specify `NONE` with the SET CURRENT MDC ROLLOUT MODE statement.

Chapter 21. Indexes

Indexes in partitioned tables

Understanding index behavior on partitioned tables

Indexes on partitioned tables are similar to indexes for ordinary tables, that is each index contains pointers to rows in all the data partitions of the table. One important difference however is that each index on a partitioned table is an independent object. In partitioned database environments, the index is distributed across the database partitions in the same manner as the table. Because an index on a partitioned table can act independently of other indexes, some special considerations are needed with respect to which table space is used when creating an index on a partitioned table.

An index on a partitioned table is created in a single table space even if the table's data partitions span multiple table spaces. Both DMS and SMS table spaces support the use of indexes in a different location than the table. All table spaces specified must be in the same database partition group. Each 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.

Additional benefits of an index on a partitioned table include:

- Improved performance of drop index and online index create.
- The ability to use different values for any of the table space characteristics between each index on the table (for example, different page sizes for each index may be appropriate to ensure better space utilization).
- Reduced IO contention providing more efficient concurrent access to the index data for the table.
- When individual indexes are dropped space will immediately become available to the system without the need for an index reorganization.
- If you choose to perform index reorganization, an individual index can be reorganized.

Figure 50 on page 282 shows a non-partitioned index on a partitioned table residing in a single table space.

© Copyright IBM Corp. 1993, 2009

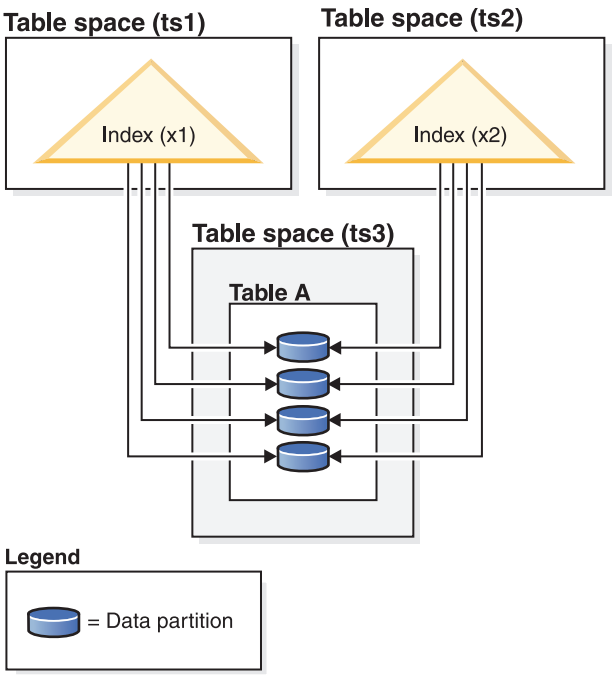


Figure 50. Index behavior on a partitioned table

Figure 51 on page 283 shows index behavior on a partitioned table that is also distributed across multiple database partitions.

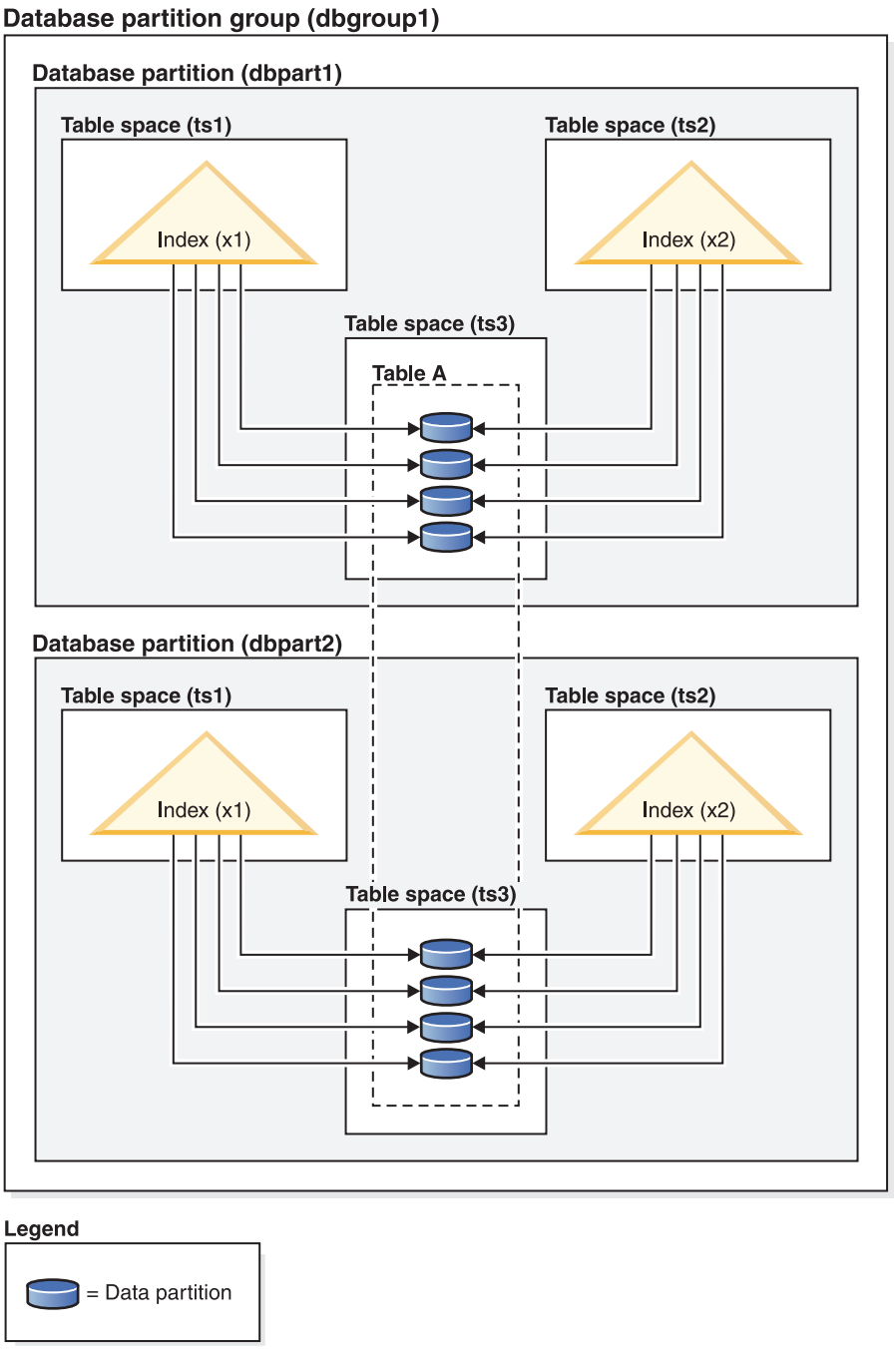


Figure 51. Index behaviour on a table that is both distributed and partitioned.

You can specify an index table space for a partitioned table in the CREATE INDEX ...IN <tbspace1> statement, which can be different from the index table space specified in the CREATE TABLE .. INDEX IN <tbspace2> statement.

For partitioned tables only, you can override the index location by using the IN clause on the CREATE INDEX statement, which allows you to specify a table space location for the index. This approach allows you to place different indexes on a partitioned table in a different table space as required. When you create a partitioned table without specifying where to place its non-partitioned indexes, and you create an index using the CREATE INDEX statement which does not specify a specific 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 created. The evaluation stops when there is a match to one of the cases:

```
Case 1:
When an index table space is specified in CREATE INDEX ... IN
<tbspace1> statement the table space specified in <tbspace1>
is used for this index.
Case 2:
When an index table space is specified in the CREATE TABLE .. INDEX IN
<tbspace2> statement the table space specified in <tbspace2>
is used for this index.
Case 3:
When no table space is specified, use the table space used by the first
attached or visible data partition.
```

Where the index is created depends on what is done during the CREATE TABLE statement. For non-partitioned tables, if you do not specify any INDEX IN clause, the database fills it in for you and is the same as your data table space. For partitioned tables, if you leave it blank, it remains as blank, and case 3 applies.

Example 1: This example assumes the existence of a partitioned table sales (a int, b int, c int), and creates a unique index 'a_idx' in the table space 'ts1'.

```
CREATE UNIQUE INDEX a_idx ON sales ( a ) IN ts1
```

Example 2: This example assumes the existence of a partitioned table sales (a int, b int, c int), and creates an index 'b_idx' in the table space 'ts2'.

```
CREATE INDEX b_idx ONsales ( b ) IN ts2
```

Understanding clustering index behavior on partitioned tables

Clustering indexes offer the same benefits for partitioned tables as they do for regular tables. However, care must be taken in choosing a clustering index with regards to the table partitioning key definitions.

You can create clustering indexes on a partitioned table using any clustering key. The database server attempts to use the clustering index to cluster data locally within each data partition. During a clustered insert, a lookup is done in the index to look for a suitable row identifier (RID). This RID is used as a starting point when looking for space in the table to insert the record. To achieve well-maintained clustering with good performance, there should be a correlation between the index columns and the table partitioning key columns. One way to ensure such correlation is to prefix the index columns by the table partitioning key columns, as shown in the following example :

```
PARTITION BY RANGE (Month, Region)
CREATE INDEX ...(Month, Region, Department) CLUSTER
```

Although the database server does not enforce this correlation, there is an expectation that all keys in the index are grouped together by partition IDs to achieve good clustering. For example, if a table is partitioned on quarter, and a clustering index is defined on date, because the relation between quarter and date exists, optimal clustering of the data with good performance can be achieved because all keys of any data partition are grouped together within the index.

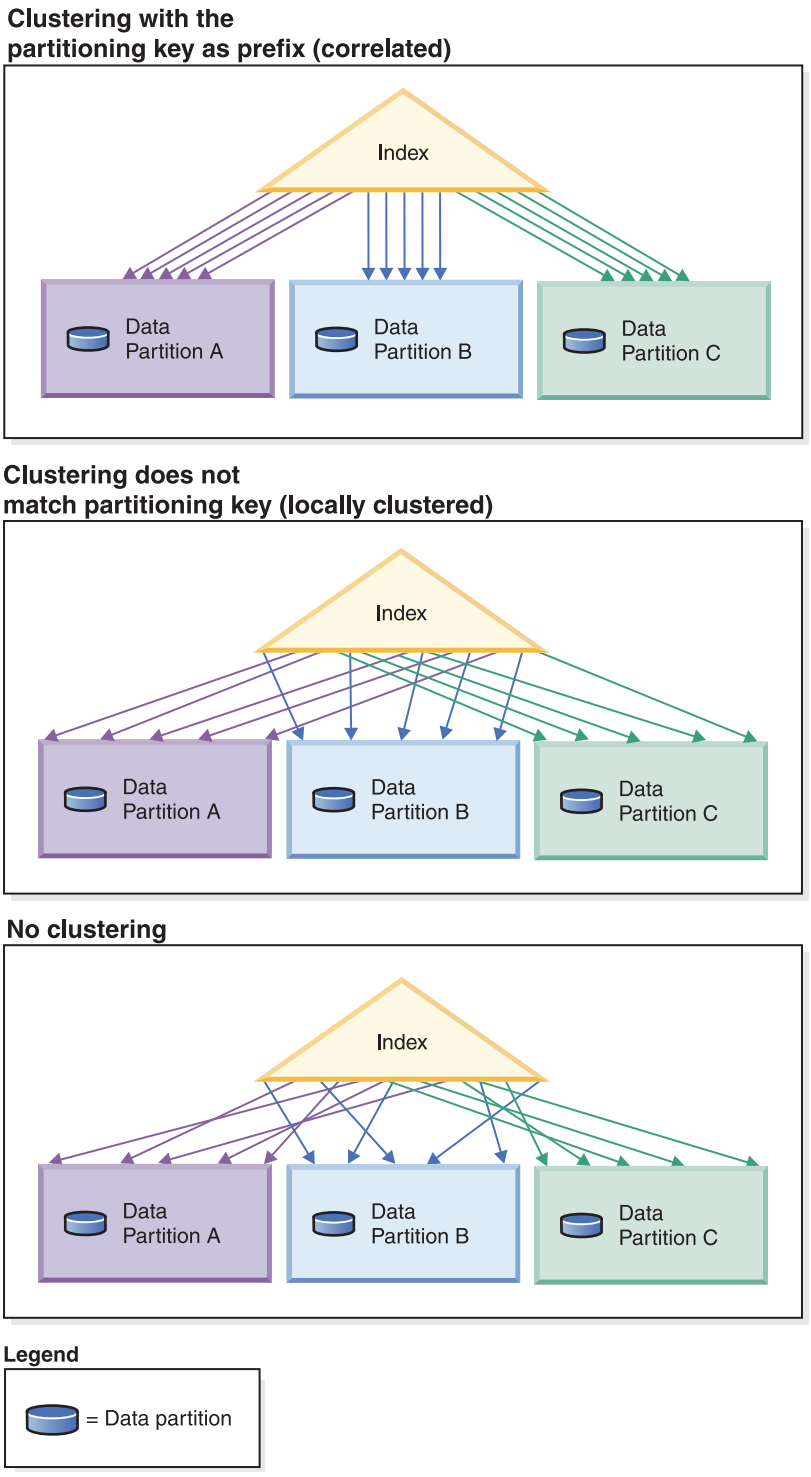


Figure 52. The possible effects of a clustered index on a partitioned table. In the first figure, data is both globally and locally clustered.

As Figure 52 shows, given the layout of the index and data in each example, optimal scan performance is achieved when the clustering correlates to the table partitioning key. When clustering is not correlated to the table partitioning key it is unlikely that the index will be locally clustered. Because a correlation between the table partitioning columns and index columns is expected, a perfect locally clustered scenario is highly unlikely.

Benefits of clustering include:

- Within each data partition, rows are in key order.
- Clustering indexes improve the performance of scans that traverse the table in the order of the keys. This is because the scan fetches the first row of the first page, then each row in that same page until it has fetched all of the rows for that page and moves on to the next. This means that only one page of the table needs to be in the buffer pool at any given time. In contrast, if the table is not clustered, then each row fetched is likely to be from a different page. Unless there is room in the buffer pool to hold the entire table, this will result in each page being fetched more than once, greatly slowing the scan.

  For partitioned tables, the ideal case of fetching each page only once during the scan can be guaranteed only if the table partitioning key is a prefix of the clustering key (see first figure in Figure 52 on page 285). However, if the clustering key is not correlated to the table partitioning key as described previously, and the data is locally clustered, you can still achieve the full benefit of the clustered index if there is enough space to hold one page of each data partition in the buffer pool. This is because each row fetched for a given data partition is near the previous row fetched for that same data partition. (see the second figure of Figure 52 on page 285). As previously mentioned, the clustering may not be well maintained in the case where the clustering key is unrelated to the table partitioning key, but if you do not expect a high level of insert, update and delete activity on your table this approach should be beneficial.

  Even if there is not sufficient space for a page of every data partition to be held in the buffer pool, there is still some benefit to be gained from defining a clustered index.

Chapter 22. Design advisor

Using the Design Advisor to migrate from a single-partition to a multiple-partition database

You can use the Design Advisor to help you migrate from a single-partition to a multiple-partition database.

About this task

The Design Advisor can provide you with recommendations for distributing your data and, at the same time, provide you with recommendations for new indexes, materialized query tables (MQTs), and multidimensional clustering (MDC) tables.

Procedure

1. Register a DB2 product or feature license key using the db2licm command.
2. Create at least one table space in a multiple-partition database partition group.

   Note: Because the Design Advisor can only recommend data redistribution to existing table spaces, the table spaces that you want the Design Advisor to consider must exist in the partitioned database before the Design Advisor is run.
3. Run the Design Advisor, with the database partitioning feature selected in the Design Advisor GUI, or with the partitioning option specified for the db2advis command.
4. If you are using the Design Advisor in the Control Center, you can implement the database partitioning recommendations automatically. If you are using the db2advis command you will need to modify the db2advis output file slightly before running the DDL statements generated by the Design Advisor.

© Copyright IBM Corp. 1993, 2009

Chapter 23. Managing concurrency

Lock modes for table and RID index scans of MDC tables

In a multi-dimensional clustering (MDC) environment, an additional lock level, BLOCK, is used. The following tables list the types of locks obtained at each level for different access plans. Each entry is made up of three parts: table lock, block lock, and row lock. A dash indicates that a particular level of locking is not used.

Note: Lock modes can be changed explicitly with the lock-request-clause of a select statement.

Table 15. Lock Modes for Table Scans with No Predicates

Isolation Level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan or delete	Update
RR	S/-/-	U/-/-	SIX/IX/X	X/-/-	X/-/-
RS	IS/IS/NS	IX/IX/U	IX/IX/U	IX/X/-	IX/I/-
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-

Table 16. Lock Modes for Table Scans with Predicates on Dimension Columns Only

Isolation Level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan or delete	Update
RR	S/-/-	U/-/-	SIX/IX/X	U/-/-	SIX/X/-
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/U/-	X/X/-
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/U/-	X/X/-
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/U/-	X/X/-

Table 17. Lock Modes for Table Scans with Other Predicates (sargs, resids)

Isolation Level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan or delete	Update
RR	S/-/-	U/-/-	SIX/IX/X	U/-/-	SIX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

© Copyright IBM Corp. 1993, 2009

The following two tables show lock modes for RID indexes on MDC tables.

Table 18. Lock Modes for RID Index Scans with No Predicates

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	S/-/-	IX/IX/S	IX/IX/X	X/-/-	X/-/-
RS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/X	X/X/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/X	X/X/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	X/X/X	X/X/X

Table 19. Lock Modes for RID Index Scans with Single Qualifying Row

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/IS/S	IX/IX/U	IX/IX/X	X/X/X	X/X/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/X	X/X/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/X	X/X/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	X/X/X	X/X/X

Table 20. Lock Modes for RID Index Scans with Start and Stop Predicates Only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/IS/S	IX/IX/S	IX/IX/X	IX/IX/X	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X

Table 21. Lock Modes for RID Index Scans with Index Predicates Only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/S/S	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

Table 22. Lock Modes for RID Index Scans with Other Predicates (sargs, resids)

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/S/S	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

Note: In the following tables, which shows lock modes for RID index scans used for deferred data-page access, at UR isolation level with IN lock for type-1 indexes or if there are predicates on include columns in the index, the isolation level is upgraded to CS and the locks are upgraded to an IS table lock, an IS block lock, and NS row locks.

Table 23. Lock modes for RID index scans used for deferred data-page access: RID index scan with no predicates

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/S/S	IX/IX/S		X/-/-	
RS	IN/IN/-	IN/IN/-		IN/IN/-	
CS	IN/IN/-	IN/IN/-		IN/IN/-	
UR	IN/IN/-	IN/IN/-		IN/IN/-	

Table 24. Lock modes for RID index scans used for deferred data-page access: Deferred data-page access after a RID index scan with no predicates

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IN/IN/-	IX/IX/S	IX/IX/X	X/-/-	X/-/-
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X

Table 25. Lock modes for RID index scans used for deferred data-page access: RID index scan with predicates (sargs, resids)

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/S/-	IX/IX/S		IX/IX/S	
RS	IN/IN/-	IN/IN/-		IN/IN/-	
CS	IN/IN/-	IN/IN/-		IN/IN/-	

Table 25. Lock modes for RID index scans used for deferred data-page access: RID index scan with predicates (sargs, resids) (continued)

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
UR	IN/IN/-	IN/IN/-		IN/IN/-	

Table 26. Lock modes for RID index scans used for deferred data-page access: Deferred data-page access after a RID index scan with predicates (sargs, resids)

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IN/IN/-	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

Table 27. Lock modes for RID index scans used for deferred data-page access: RID index scan with start and stop predicates only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/IS/S	IX/IX/S		IX/IX/X	
RS	IN/IN/-	IN/IN/-		IN/IN/-	
CS	IN/IN/-	IN/IN/-		IN/IN/-	
UR	IN/IN/-	IN/IN/-		IN/IN/-	

Table 28. Lock modes for RID index scans used for deferred data-page access: Deferred data-page access after a RID index scan with start and stop predicates only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IN/IN/-	IX/IX/S	IX/IX/X	IX/IX/X	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IS/-/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

Locking for block index scans for MDC tables

The following tables list the types of locks obtained at each level for different access plans. Each entry is made up of three parts: table lock, block lock, and row lock. A dash indicates that a particular level of locking is not done.

Note: Lock modes can be changed explicitly with the lock-request-clause of a select statement.

Table 29. Lock Modes for Index Scans with No Predicates

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	S/--/--	IX/IX/S	IX/IX/X	X/--/--	X/--/--
RS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/--	X/X/--
CS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/--	X/X/--
UR	IN/IN/-	IX/IX/U	IX/IX/X	X/X/--	X/X/--

Table 30. Lock Modes for Index Scans with Dimension Predicates Only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/-/-	IX/IX/S	IX/IX/X	X/-/-	X/-/-
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-

Table 31. Lock Modes for Index Scans with Start and Stop Predicates Only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/S/-	IX/IX/S	IX/IX/S	IX/IX/S	IX/IX/S
RS	IX/IX/S	IX/IX/U	IX/IX/X	IX/IX/-	IX/IX/-
CS	IX/IX/S	IX/IX/U	IX/IX/X	IX/IX/-	IX/IX/-
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/-	IX/IX/-

Table 32. Lock Modes for Index Scans with Predicates

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/S/-	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

The following table lists lock modes for block index scans used for deferred data-page access:

Table 33. Lock modes for block index scans used for deferred data-page access: Block index scan with no predicates

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/S/--	IX/IX/S		X/--/--	
RS	IN/IN/--	IN/IN/--		IN/IN/--	
CS	IN/IN/--	IN/IN/--		IN/IN/--	
UR	IN/IN/--	IN/IN/--		IN/IN/--	

Table 34. Lock modes for block index scans used for deferred data-page access: Deferred data-page access after a block index scan with no predicates

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IN/IN/--	IX/IX/S	IX/IX/X	X/--/--	X/--/--
RS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/--	X/X/--
CS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/--	X/X/--
UR	IN/IN/--	IX/IX/U	IX/IX/X	X/X/--	X/X/--

Table 35. Lock modes for block index scans used for deferred data-page access: Block index scan with dimension predicates only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/S/--	IX/IX/--		IX/S/--	
RS	IS/IS/NS	IX/--/--		IX/--/--	
CS	IS/IS/NS	IX/--/--		IX/--/--	
UR	IN/IN/--	IX/--/--		IX/--/--	

Table 36. Lock modes for block index scans used for deferred data-page access: Deferred data-page access after a block index scan with dimension predicates only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IN/IN/--	IX/IX/S	IX/IX/X	IX/S/--	IX/X/--
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/U/--	IX/X/--
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/U/--	IX/X/--
UR	IN/IN/--	IX/IX/U	IX/IX/X	IX/U/--	IX/X/--

Table 37. Lock modes for block index scans used for deferred data-page access: Block index scan with start and stop predicates only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/S/--	IX/IX/--		IX/X/--	
RS	IN/IN/--	IN/IN/--		IN/IN/--	
CS	IN/IN/--	IN/IN/--		IN/IN/--	
UR	IN/IN/--	IN/IN/--		IN/IN/--	

Table 38. Lock modes for block index scans used for deferred data-page access: Deferred data-page access after a block index scan with start and stop predicates only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IN/IN/--	IX/IX/X		IX/X/--	
RS	IS/IS/NS	IN/IN/--		IN/IN/--	
CS	IS/IS/NS	IN/IN/--		IN/IN/--	
UR	IS/--/--	IN/IN/--		IN/IN/--	

Table 39. Lock modes for block index scans used for deferred data-page access: Block index scan other predicates (sargs, resids)

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/S/--	IX/IX/--		IX/IX/--	
RS	IN/IN/--	IN/IN/--		IN/IN/--	
CS	IN/IN/--	IN/IN/--		IN/IN/--	
UR	IN/IN/--	IN/IN/--		IN/IN/--	

Table 40. Lock modes for block index scans used for deferred data-page access: Deferred data-page access after a block index scan with other predicates (sargs, resids)

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IN/IN/--	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/--	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

Locking behavior on partitioned tables

In addition to an overall table lock, there is a lock for each data partition of a partitioned table. This allows for finer granularity and increased concurrency compared to a non-partitioned table. The new data partition lock is identified in the output of the db2pd command, event monitors, administrative views, and table functions.

When accessing a table, locking behavior obtains the table lock first, and then acquires data partition locks as dictated by the data accessed. Access methods and isolation levels might require locking of data partitions not in the result set. Once these data partition locks are acquired they might be held as long as the table lock. For example, a Cursor stability (CS) scan over an index might keep the locks on previously accessed data partitions to reduce the costs of re-acquiring the data partition lock if that data partition is referenced in subsequent keys. The data partition lock also carries the cost of ensuring access to the table spaces. For non-partitioned tables, table space access is handled by the table lock. Therefore, data partition locking occurs even if there is an exclusive or share lock at the table level for a partitioned table.

Finer granularity allows one transaction to have exclusive access to a given data partition and avoid row locking while other transactions are able to access other data partitions. This can be a result of the plan chosen for a mass update or due to escalation of locks to the data partition level. The table lock for many access methods is normally an intent lock, even if the data partitions are locked in share or exclusive. This allows for increased concurrency. However, if non-intent locks are required at the data partition level and the plan indicates that all data partitions may be accessed, then a non-intent might be chosen at the table level to prevent deadlocks between data partition locks from concurrent transactions.

Locking for SQL LOCK TABLE statements

For partitioned tables, the only lock acquired for the LOCK TABLE statement is at the table level; no data partition locks are acquired. This ensures no row locking to the table for subsequent DML statements as well as avoiding deadlocks at the row, block, or data partition level. Using LOCK TABLE IN EXCLUSIVE MODE can also be used to guarantee exclusive access when updating indexes, which is useful in limiting the growth of type 2 indexes during a large update.

Effect of the LOCKSIZE TABLE parameter of the ALTER TABLE statement

The ALTER TABLE statement has an option for setting LOCKSIZE TABLE, which ensures that the table is locked share or exclusive with no intent locks. For a partitioned table this lock strategy is applied to both the table lock and to the data partition locks for any data partitions accessed.

Row and block lock escalation

For partitioned tables, the unit of escalation of row and block locks is to the data partition level. This again means that a table is more accessible to other transactions even if a data partition is escalated to share, exclusive, or super exclusive, leaving other non-escalated data partitions are unaffected. The transaction might continue to row lock on other data partitions after escalation for a given data partition. The notification log message for escalations includes the data partition escalated as well as the partitioned table's name. Therefore, exclusive

access to an index cannot be ensured by lock escalation. Either the statement must use an exclusive table level lock, an explicit LOCK TABLE IN EXCLUSIVE MODE statement must be issued, or the table must use the LOCKSIZE TABLE attribute. The overall table lock for the access method is chosen by the optimizer, and depends upon data partition elimination. A large update to a table might choose an exclusive table lock if there is no data partition elimination occurring.

Interpreting lock information

The following example from the SNAPLOCK administrative view can help you interpret lock information returned for a partitioned table.

Example 1:

This SNAPLOCK administrative view was captured during an offline index reorganization.

```
SELECT SUBSTR(TABNAME, 1, 15) TABNAME, TAB_FILE_ID, SUBSTR(TBSP_NAME, 1, 15) TBSP_NAME, DATA_PARTITION_ID, LOCK_OBJECT_TYPE,
LOCK_MODE, LOCK_ESCALATION FROM SYSIBMADM.SNAPLOCK where TABNAME like 'TP1' and LOCK_OBJECT_TYPE like 'TABLE_%'
ORDER BY TABNAME, DATA_PARTITION_ID, LOCK_OBJECT_TYPE, TAB_FILE_ID, LOCK_MODE

TABNAME         TAB_FILE_ID          TBSP_NAME       DATA_PARTITION_ID LOCK_OBJECT_TYPE   LOCK_MODE  LOCK_ESCALATION
--------------- -------------------- --------------- ----------------- ------------------ ---------- ---------------
TP1                           32768 -                              -1 TABLE_LOCK         Z                       0
TP1                               4 USERSPACE1                     0 TABLE_PART_LOCK    Z                       0
TP1                               5 USERSPACE1                     1 TABLE_PART_LOCK    Z                       0
TP1                               6 USERSPACE1                     2 TABLE_PART_LOCK    Z                       0
TP1                               7 USERSPACE1                     3 TABLE_PART_LOCK    Z                       0
TP1                               8 USERSPACE1                     4 TABLE_PART_LOCK    Z                       0
TP1                               9 USERSPACE1                     5 TABLE_PART_LOCK    Z                       0
TP1                              10 USERSPACE1                     6 TABLE_PART_LOCK    Z                       0
TP1                              11 USERSPACE1                     7 TABLE_PART_LOCK    Z                       0
TP1                              12 USERSPACE1                     8 TABLE_PART_LOCK    Z                       0
TP1                              13 USERSPACE1                     9 TABLE_PART_LOCK    Z                       0
TP1                              14 USERSPACE1                    10 TABLE_PART_LOCK    Z                       0
TP1                              15 USERSPACE1                    11 TABLE_PART_LOCK    Z                       0
TP1                               4 USERSPACE1                     - TABLE_LOCK         Z                       0
TP1                               5 USERSPACE1                     - TABLE_LOCK         Z                       0
TP1                               6 USERSPACE1                     - TABLE_LOCK         Z                       0
TP1                               7 USERSPACE1                     - TABLE_LOCK         Z                       0
TP1                               8 USERSPACE1                     - TABLE_LOCK         Z                       0
TP1                               9 USERSPACE1                     - TABLE_LOCK         Z                       0
TP1                              10 USERSPACE1                     - TABLE_LOCK         Z                       0
TP1                              11 USERSPACE1                     - TABLE_LOCK         Z                       0
TP1                              12 USERSPACE1                     - TABLE_LOCK         Z                       0
TP1                              13 USERSPACE1                     - TABLE_LOCK         Z                       0
TP1                              14 USERSPACE1                     - TABLE_LOCK         Z                       0
TP1                              15 USERSPACE1                     - TABLE_LOCK         Z                       0
TP1                              16 USERSPACE1                     - TABLE_LOCK         Z                       0

  26 record(s) selected.
```

In this example, a lock object of type TABLE_LOCK and a DATA_PARTITION_ID of -1 are used to control access to, and concurrency on, the partitioned table TP1. The lock objects of type TABLE_PART_LOCK are used to control most access and concurrency to each data partition.

There are additional lock objects of type TABLE_LOCK captured in this output (TAB_FILE_ID 4 through 16) which do not have a value for DATA_PARTITION_ID. A lock of this type, where an object with a TAB_FILE_ID and a TBSP_NAME correspond to a data partition or index on the partitioned table, might be used to control concurrency with the online backup utility.

Chapter 24. Agent management

Agents in a partitioned database

For partitioned database environments and environments with intra-partition parallelism enabled, each database partition (that is, each database server or node) has its own pool of agents from which subagents are drawn. Because of this pool, subagents do not have to be created and destroyed each time one is needed or is finished its work. The subagents can remain as associated agents in the pool and be used by the database manager for new requests from the application they are associated with or from new applications.

For partitioned database environments and environments with intra-partition parallelism enabled, the impact to performance and memory costs within the system is strongly related to how your agent pool is tuned:

- The database manager configuration parameter for agent pool size (*num_poolagents*) affects the total number of both agents and subagents that can be kept associated with applications on a database partition, which is also called a node. If the pool size is too small and the pool is full, a subagent disassociates itself from the application it is working on and terminates. Because subagents must be constantly created and re-associated to applications, performance suffers.

  By default *num_poolagents* is set to AUTOMATIC with a value of 100. When this parameter is set to AUTOMATIC, the database manager automatically manages the number of idle agents to pool.

  In addition, if the value of *num_poolagents* is too small, one application may fill the pool with associated subagents. Then when another application requires a new subagent and has no subagents in its associated agent pool, it will recycle inactive subagents from the agent pools of other applications. This behavior ensures that resources are fully utilized.
- Weigh concerns about having too few agents against the resource costs of allowing too many agents to be active at any given time.

  For example, if the value of *num_poolagents* is too large, associated subagents may sit unused in the pool for long periods of time, using database manager resources that are not available for other tasks.

  Note: When the connection concentrator is enabled, the number of agents specified by *num_poolagents* does not necessarily reflect the exact number of agents that may sit idle in the pool at any one time. The number of agents may be exceeded temporarily to handle occurrences of higher workload activity.

Other asynchronous processes and threads

In addition to the database agents, other asynchronous database-manager activities run as their own process or thread including:

- Database I/O servers or I/O prefetchers
- Database asynchronous page cleaners
- Database loggers
- Database deadlock detectors
- Communication and IPC listeners

© Copyright IBM Corp. 1993, 2009

- Table space container rebalancers.

Chapter 25. Optimizing access plans

Index access and cluster ratios

When it chooses an access plan, the optimizer estimates the number of I/Os required to fetch required pages from disk to the buffer pool. This estimate includes a prediction of buffer-pool usage, since additional I/Os are not required to read rows in a page that is already in the buffer pool.

For index scans, information from the system catalog tables (SYSCAT.INDEXES) helps the optimizer estimate I/O cost of reading data pages into the buffer pool. It uses information from the following columns in the SYSCAT.INDEXES table:

- CLUSTERRATIO information indicates the degree to which the table data is clustered in relation to this index. The higher the number, the better rows are ordered in index key sequence. If table rows are in close to index-key sequence, rows can be read from a data page while the page is in the buffer. If the value of this column is -1, the optimizer uses PAGE_FETCH_PAIRS and CLUSTERFACTOR information if it is available.
- PAGE_FETCH_PAIRS contains pairs of numbers that model the number of I/Os required to read the data pages into buffer pools of various sizes together with CLUSTERFACTOR information. Data is collected for these columns only if you execute RUNSTATS on the index with the DETAILED clause.

If index clustering statistics are not available, the optimizer uses default values, which assume poor clustering of the data to the index.

The degree to which the data is clustered with respect to the index can have a significant impact on performance and you should try to keep one of the indexes on the table close to 100 percent clustered.

In general, only one index can be one hundred percent clustered, except in those cases where the keys are a superset of the keys of the clustering index or where there is de facto correlation between the key columns of the two indexes.

When you reorganize an table, you can specify an index that will be used to cluster the rows and attempt to preserve this characteristic during insert processing. Because updates and inserts may make the table less well clustered in relation to the index, you might need to periodically reorganize the table. To reduce the frequency of reorganization on a table that has frequent changes due to INSERTs, UPDATEs, and DELETES, use the PCTFREE parameter when you alter a table. This allows for additional inserts to be clustered with the existing data.

Table and index management for MDC tables

Table and index organization for multi-dimensional clustering (MDC) tables is based on the same logical structures as standard table organization. Like standard tables, MDC tables are organized into pages that contain rows of data, divided into columns, and the rows on each page are identified by row IDs (RIDs). In addition, however, the pages of MDC tables are grouped into extent-sized blocks. For example, in the illustration below, which shows a table with an extent size of four, the first four pages, numbered 0 through 3, are the first block in the table. The next set of pages, numbered 4 through 7, are the second block in the table.

© Copyright IBM Corp. 1993, 2009

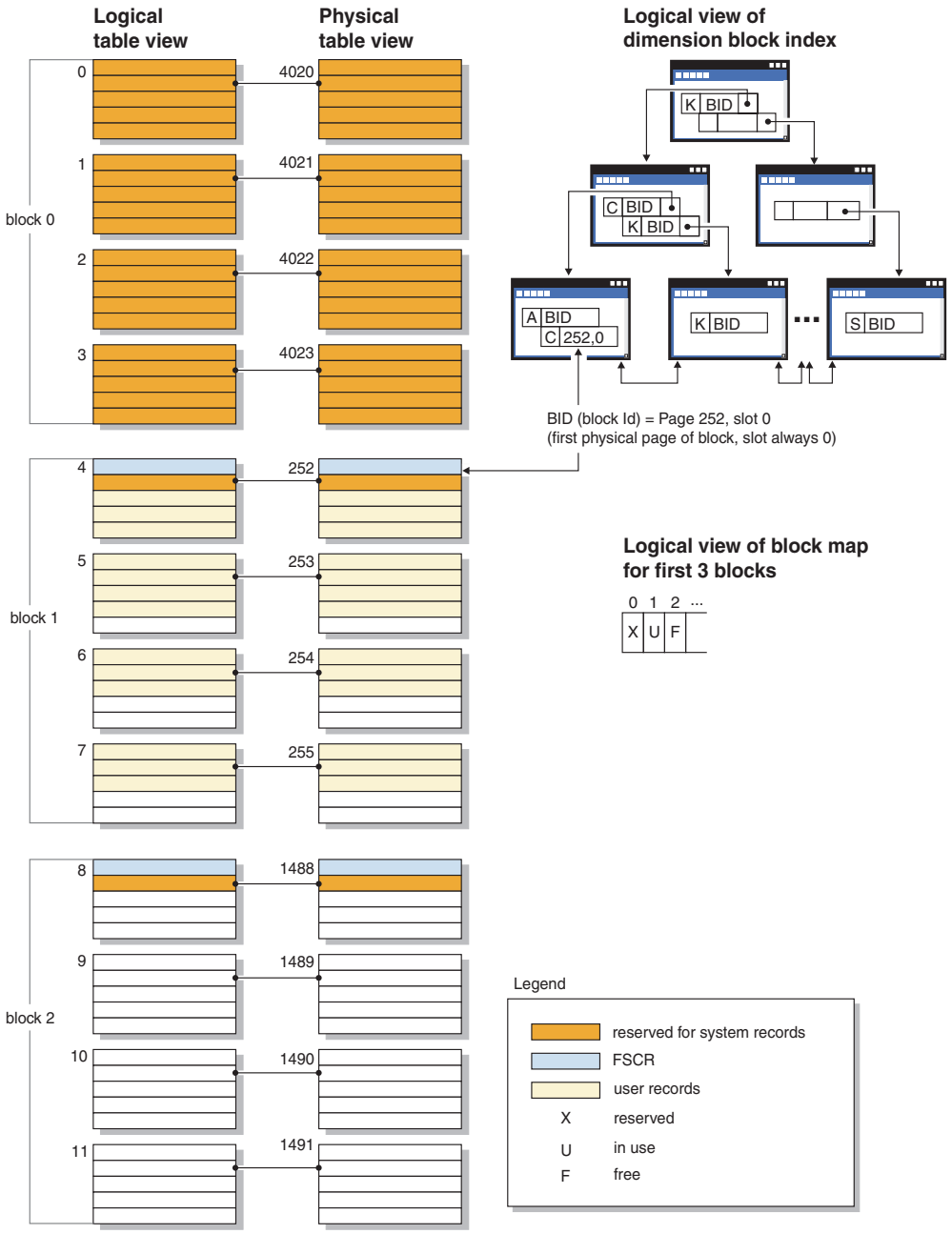


Figure 53. Logical table, record, and index structure for MDC tables

The first block contains special internal records that are used by DB2 to manage the table, including the free-space control record (FSCR). In subsequent blocks, the first page contains the FSCR. An FSCR maps the free space for new records that exists on each of the pages in the block. This available free space is used when inserting records into the table.

As the name implies, MDC tables cluster data on more than one dimension. Each dimension is determined by a column or set of columns that you specify in the ORGANIZE BY DIMENSIONS clause of the CREATE TABLE statement. When you create an MDC table, the following two kinds of indexes are created automatically:

- A dimension-block index, which contains pointers to each occupied block for a single dimension.

- A composite block index, which contains all dimension key columns. The composite block index is used to maintain clustering during insert and update activity.

The optimizer considers access plans which utilize dimension-block indexes when it determines the most efficient access plan for a particular query. When queries have predicates on dimension values, the optimizer can use the dimension block index to identify, and fetch from, the extents that contain these values. Because extents are physically contiguous pages on disk, this results in more efficient performance and minimizes I/O.

In addition, you can create specific RID indexes if analysis of data access plans indicates that such indexes would improve query performance.

Along with the dimension block indexes and the composite block index, MDC tables maintain a block map that contains a bitmap that indicates the availability status of each block. The following attributes are coded in the bitmap list:

- X (reserved): the first block contains only system information for the table.
- U (in use): this block is used and associated with a dimension block index
- L (loaded): this block has been loaded by a current load operation
- C (check constraint): this block is set by the load operation to specify incremental constraint checking during the load.
- T (refresh table): this block is set by the load operation to specify that AST maintenance is required.
- F (free): If no other attribute is set, the block is considered free.

Because each block has an entry in the block map file, the file grows as the table grows. This file is stored as a separate object. In an SMS table space it is a new file type. In a DMS table space, it has a new object descriptor in the object table.

Clustering

Over time, updates might cause rows on data pages to change location lowering the degree of *clustering* that exists between an index and the data pages.

Reorganizing a table with respect to a chosen index re-clusters the data. A clustered index is most useful for columns that have range predicates because it allows better sequential access of data in the base table. This results in fewer page fetches, since like values are on the same data page.

In general, only one of the indexes in a table can have a high degree of clustering.

To check the degree of clustering for an index, double-click on its node to display the Index Statistics window. The cluster ratio or cluster factor values are shown in this window. If the value is low, consider reorganizing the table's data.

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 execution 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 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. A range of pages or rows is assigned to a subagent. A subagent scans its assigned range and is assigned another range when it has completed its 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 the parallel table scan with subagents being assigned a range of records. A subagent is assigned a new range when it has complete its work on the current range.

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 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, then 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 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 completed, each subagent reads the entire sort. If the number of rows is small, this sort may be used to rebalance the data stream.
- **Shared sort**

This sort is the same as a replicated sort, except the subagents open a parallel scan on the sorted result to distribute the data among the subagents in a way similar to the 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 in parallel by subagents. 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 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 in parallel by subagents. 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 re-evaluated independently by each subagent. A merged join can be parallelized if its inner and outer streams are value-partitioned for partitioned sorts.

Examples of db2expln and dynexpln output

The examples shown here can help you understand the layout and format of the output from `db2expln` and `dynexpln`. These examples were run against the SAMPLE database that is provided with DB2, unless shown otherwise. A brief discussion is included with each example. Significant differences from one example to the next have been shown in **bold**.

Example one: no parallelism

This example is simply requesting a list of all employee names, their jobs, department name and location, and the project names on which they are working. The essence of this access plan is that hash joins are used to join the relevant data

from each of the specified tables. Since no indexes are available, the access plan does a relation scan of each table as it is joined.

```
******************** PACKAGE ************************************

Package Name = "DOOLE"."EXAMPLE"  Version = ""

 Prep Date = 2002/01/04
 Prep Time = 14:05:00

 Bind Timestamp = 2002-01-04-14.05.00.415403

 Isolation Level          = Cursor Stability
 Blocking                 = Block Unambiguous Cursors
 Query Optimization Class = 5

 Partition Parallel       = No
 Intra-Partition Parallel = No

 SQL Path                 = "SYSIBM", "SYSFUN", "SYSPROC", "DOOLE"

-------------------- SECTION ---------------------------------------
Section = 1


SQL Statement:
  DECLARE EMPCUR CURSOR
  FOR
     SELECT e.lastname, e.job, d.deptname, d.location, p.projname
     FROM employee AS e, department AS d, project AS p
     WHERE e.workdept = d.deptno AND e.workdept = p.deptno


Estimated Cost        = 120.518692
Estimated Cardinality = 221.535980

(    6) Access Table Name = DOOLE.EMPLOYEE  ID = 2,5
         |  #Columns = 3
         |  Relation Scan
         |  |  Prefetch: Eligible
         |  Lock Intents
         |  |  Table: Intent Share
         |  |  Row  : Next Key Share
(    6) |  Process Build Table for Hash Join
(    2) Hash Join
         |  Estimated Build Size: 7111
         |  Estimated Probe Size: 9457
(    5)  |  Access Table Name = DOOLE.PROJECT  ID = 2,7
         |  |  #Columns = 2
         |  |  Relation Scan
         |  |  |  Prefetch: Eligible
         |  |  Lock Intents
         |  |  |  Table: Intent Share
         |  |  |  Row  : Next Key Share
(    5)  |  |  Process Build Table for Hash Join
(    3)  |  Hash Join
         |  |  Estimated Build Size: 5737
         |  |  Estimated Probe Size: 6421
(    4)  |  |  Access Table Name = DOOLE.DEPARTMENT  ID = 2,4
         |  |  |  #Columns = 3
         |  |  |  Relation Scan
         |  |  |  |  Prefetch: Eligible
         |  |  |  Lock Intents
         |  |  |  |  Table: Intent Share
         |  |  |  |  Row  : Next Key Share
(    4)  |  |  |  Process Probe Table for Hash Join
(    1) Return Data to Application
```

```
        |  #Columns = 5

End of section


Optimizer Plan:

              RETURN
              (   1)
                |
              HSJOIN
              (   2)
             /      \
        HSJOIN        TBSCAN
        (   3)        (   6)
       /      \         |
   TBSCAN    TBSCAN   Table:
   (   4)    (   5)   DOOLE
     |         |      EMPLOYEE
 Table:      Table:
 DOOLE       DOOLE
 DEPARTMENT  PROJECT
```

The first part of the plan accesses the DEPARTMENT and PROJECT tables and uses a hash join to join them. The result of this join is joined to the EMPLOYEE table. The resulting rows are returned to the application.

Example two: single-partition plan with intra-partition parallelism

This example shows the same SQL statement as the first example, but this query has been compiled for a four-way SMP machine.

```
******************** PACKAGE ************************************

Package Name = "DOOLE"."EXAMPLE"  Version = ""

 Prep Date = 2002/01/04
 Prep Time = 14:12:38

 Bind Timestamp = 2002-01-04-14.12.38.732627

 Isolation Level          = Cursor Stability
 Blocking                 = Block Unambiguous Cursors
 Query Optimization Class = 5

 Partition Parallel       = No
 Intra-Partition Parallel = Yes (Bind Degree = 4)

 SQL Path                 = "SYSIBM", "SYSFUN", "SYSPROC", "DOOLE"

-------------------- SECTION ---------------------------------------
Section = 1


SQL Statement:
  DECLARE EMPCUR CURSOR
  FOR
     SELECT e.lastname, e.job, d.deptname, d.location, p.projname
     FROM employee AS e, department AS d, project AS p
     WHERE e.workdept = d.deptno AND e.workdept = p.deptno

Intra-Partition Parallelism Degree = 4

Estimated Cost        = 133.934692
Estimated Cardinality = 221.535980
```

```
(    2) Process Using 4 Subagents
(    7) |  Access Table Name = DOOLE.EMPLOYEE  ID = 2,5
        |  |  #Columns = 3
        |  |  Parallel Scan
        |  |  Relation Scan
        |  |  |  Prefetch: Eligible
        |  |  Lock Intents
        |  |  |  Table: Intent Share
        |  |  |  Row  : Next Key Share
(    7) |  |  Process Build Table for Hash Join
(    3) |  Hash Join
        |  |  Estimated Build Size: 7111
        |  |  Estimated Probe Size: 9457
(    6) |  |  Access Table Name = DOOLE.PROJECT  ID = 2,7
        |  |  |  #Columns = 2
        |  |  |  Parallel Scan
        |  |  |  Relation Scan
        |  |  |  |  Prefetch: Eligible
        |  |  |  Lock Intents
        |  |  |  |  Table: Intent Share
        |  |  |  |  Row  : Next Key Share
(    6) |  |  |  Process Build Table for Hash Join
(    4) |  |  Hash Join
        |  |  |  Estimated Build Size: 5737
        |  |  |  Estimated Probe Size: 6421
(    5) |  |  |  Access Table Name = DOOLE.DEPARTMENT  ID = 2,4
        |  |  |  |  #Columns = 3
        |  |  |  |  Parallel Scan
        |  |  |  |  Relation Scan
        |  |  |  |  |  Prefetch: Eligible
        |  |  |  |  Lock Intents
        |  |  |  |  |  Table: Intent Share
        |  |  |  |  |  Row  : Next Key Share
(    5) |  |  |  |  Process Probe Table for Hash Join
(    2) |  Insert Into Asynchronous Local Table Queue  ID = q1
(    2) Access Local Table Queue  ID = q1  #Columns = 5
(    1) Return Data to Application
        |  #Columns = 5

End of section


Optimizer Plan:

              RETURN
              (   1)
                |
               LTQ
              (   2)
                |
              HSJOIN
              (   3)
             /      \
       HSJOIN         TBSCAN
       (   4)         (   7)
      /      \          |
   TBSCAN    TBSCAN   Table:
   (   5)    (   6)   DOOLE
     |         |      EMPLOYEE
 Table:      Table:
 DOOLE       DOOLE
 DEPARTMENT  PROJECT
```

This plan is almost identical to the plan in the first example. The main differences are the creation of four subagents when the plan first starts and the table queue at the end of the plan to gather the results of each of subagent's work before returning them to the application.

Example three: multipartition plan with inter-partition parallelism

This example shows the same SQL statement as the first example, but this query has been compiled on a partitioned database made up of three database partitions.

```
******************** PACKAGE ************************************

Package Name = "DOOLE"."EXAMPLE"  Version = ""

 Prep Date = 2002/01/04
 Prep Time = 14:54:57

 Bind Timestamp = 2002-01-04-14.54.57.033666

 Isolation Level          = Cursor Stability
 Blocking                 = Block Unambiguous Cursors
 Query Optimization Class = 5

 Partition Parallel       = Yes
 Intra-Partition Parallel = No

 SQL Path                 = "SYSIBM", "SYSFUN", "SYSPROC", "DOOLE"

-------------------- SECTION ---------------------------------------
Section = 1


SQL Statement:
  DECLARE EMPCUR CURSOR
  FOR
     SELECT e.lastname, e.job, d.deptname, d.location, p.projname
     FROM employee AS e, department AS d, project AS p
     WHERE e.workdept = d.deptno AND e.workdept = p.deptno


Estimated Cost        = 118.483406
Estimated Cardinality = 474.720032

        Coordinator Subsection:
(-----)    Distribute Subsection #2
           |  Broadcast to Node List
           |  |  Nodes = 10, 33, 55
(-----)    Distribute Subsection #3
           |  Broadcast to Node List
           |  |  Nodes = 10, 33, 55
(-----)    Distribute Subsection #1
           |  Broadcast to Node List
           |  |  Nodes = 10, 33, 55
(    2)    Access Table Queue  ID = q1  #Columns = 5
(    1)    Return Data to Application
           |  #Columns = 5

        Subsection #1:
(    8)    Access Table Queue  ID = q2  #Columns = 2
(    3)    Hash Join
           |  Estimated Build Size: 5737
           |  Estimated Probe Size: 8015
(    6)    |  Access Table Queue  ID = q3  #Columns = 3
(    4)    |  Hash Join
           |  |  Estimated Build Size: 5333
```

```
            |  |  Estimated Probe Size: 6421
(    5)     |  |  Access Table Name = DOOLE.DEPARTMENT  ID = 2,4
            |  |  |  #Columns = 3
            |  |  |  Relation Scan
            |  |  |  |  Prefetch: Eligible
            |  |  |  Lock Intents
            |  |  |  |  Table: Intent Share
            |  |  |  |  Row  : Next Key Share
(    5)     |  |  |  Process Probe Table for Hash Join
(    2)     Insert Into Asynchronous Table Queue  ID = q1
            |  Broadcast to Coordinator Node
            |  Rows Can Overflow to Temporary Table

         Subsection #2:
(    9)     Access Table Name = DOOLE.PROJECT  ID = 2,7
            |  #Columns = 2
            |  Relation Scan
            |  |  Prefetch: Eligible
            |  Lock Intents
            |  |  Table: Intent Share
            |  |  Row  : Next Key Share
(    9)     |  Insert Into Asynchronous Table Queue  ID = q2
            |  |  Hash to Specific Node
            |  |  Rows Can Overflow to Temporary Tables
(    8)     Insert Into Asynchronous Table Queue Completion  ID = q2

         Subsection #3:
(    7)     Access Table Name = DOOLE.EMPLOYEE  ID = 2,5
            |  #Columns = 3
            |  Relation Scan
            |  |  Prefetch: Eligible
            |  Lock Intents
            |  |  Table: Intent Share
            |  |  Row  : Next Key Share
(    7)     |  Insert Into Asynchronous Table Queue  ID = q3
            |  |  Hash to Specific Node
            |  |  Rows Can Overflow to Temporary Tables
(    6)     Insert Into Asynchronous Table Queue Completion  ID = q3

End of section


Optimizer Plan:

                RETURN
                (   1)
                  |
                 BTQ
                (   2)
                  |
                HSJOIN
                (   3)
               /      \
          HSJOIN          DTQ
          (   4)         (   8)
         /      \          |
    TBSCAN      DTQ     TBSCAN
    (   5)     (   6)   (   9)
      |          |        |
  Table:       TBSCAN   Table:
  DOOLE        (   7)   DOOLE
  DEPARTMENT     |      PROJECT
              Table:
              DOOLE
              EMPLOYEE
```

This plan has all the same pieces as the plan in the first example, but the section has been broken into four subsections. The subsections have the following tasks:

- **Coordinator Subsection**. This subsection coordinates the other subsections. In this plan, it causes the other subsections to be distributed and then uses a table queue to gather the results to be returned to the application.
- **Subsection #1**. This subsection scans table queue q2 and uses a hash join to join it with the data from table queue q3. A second hash join then adds in the data from the DEPARTMENT table. The joined rows are then sent to the coordinator subsection using table queue q1.
- **Subsection #2**. This subsection scans the PROJECT table and hashes to a specific node with the results. These results are read by Subsection #1.
- **Subsection #3**. This subsection scans the EMPLOYEE table and hashes to a specific node with the results. These results are read by Subsection #1.

Example four: multipartition plan with inter-partition and intra-partition parallelism

This example shows the same SQL statement as the first example, but this query has been compiled on a partitioned database made up of three database partitions, each of which is on a four-way SMP machine.

```
******************** PACKAGE ************************************

Package Name = "DOOLE"."EXAMPLE"  Version = ""

 Prep Date = 2002/01/04
 Prep Time = 14:58:35

 Bind Timestamp = 2002-01-04-14.58.35.169555

 Isolation Level          = Cursor Stability
 Blocking                 = Block Unambiguous Cursors
 Query Optimization Class = 5

 Partition Parallel       = Yes
 Intra-Partition Parallel = Yes (Bind Degree = 4)

 SQL Path                 = "SYSIBM", "SYSFUN", "SYSPROC", "DOOLE"

-------------------- SECTION ---------------------------------------
Section = 1


SQL Statement:
  DECLARE EMPCUR CURSOR
  FOR
     SELECT e.lastname, e.job, d.deptname, d.location, p.projname
     FROM employee AS e, department AS d, project AS p
     WHERE e.workdept = d.deptno AND e.workdept = p.deptno

Intra-Partition Parallelism Degree = 4

Estimated Cost        = 145.198898
Estimated Cardinality = 474.720032

        Coordinator Subsection:
(-----)    Distribute Subsection #2
           |  Broadcast to Node List
           |  |  Nodes = 10, 33, 55
(-----)    Distribute Subsection #3
           |  Broadcast to Node List
           |  |  Nodes = 10, 33, 55
(-----)    Distribute Subsection #1
```

```
            |  Broadcast to Node List
            |  |  Nodes = 10, 33, 55
(    2)     Access Table Queue  ID = q1  #Columns = 5
(    1)     Return Data to Application
            |  #Columns = 5

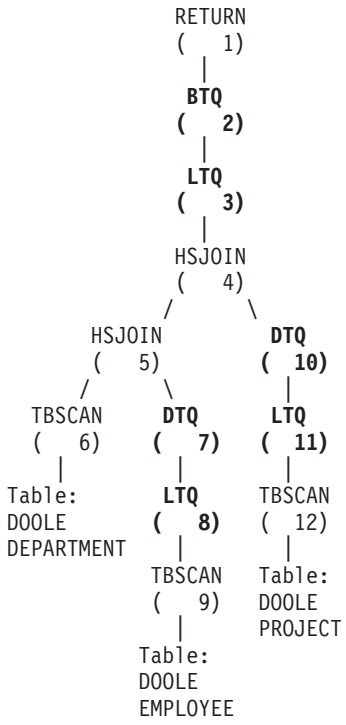
        Subsection #1:
(    3)     Process Using 4 Subagents
(   10)     |  Access Table Queue  ID = q3  #Columns = 2
(    4)     |  Hash Join
            |  |  Estimated Build Size: 5737
            |  |  Estimated Probe Size: 8015
(    7)     |  |  Access Table Queue  ID = q5  #Columns = 3
(    5)     |  |  Hash Join
            |  |  |  Estimated Build Size: 5333
            |  |  |  Estimated Probe Size: 6421
(    6)     |  |  |  Access Table Name = DOOLE.DEPARTMENT  ID = 2,4
            |  |  |  |  #Columns = 3
            |  |  |  |  Parallel Scan
            |  |  |  |  Relation Scan
            |  |  |  |  |  Prefetch: Eligible
            |  |  |  |  Lock Intents
            |  |  |  |  |  Table: Intent Share
            |  |  |  |  |  Row  : Next Key Share
(    6)     |  |  |  |  Process Probe Table for Hash Join
(    3)     |  Insert Into Asynchronous Local Table Queue  ID = q2
(    3)     Access Local Table Queue  ID = q2  #Columns = 5
(    2)     Insert Into Asynchronous Table Queue  ID = q1
            |  Broadcast to Coordinator Node
            |  Rows Can Overflow to Temporary Table

        Subsection #2:
(   11)     Process Using 4 Subagents
(   12)     |  Access Table Name = DOOLE.PROJECT  ID = 2,7
            |  |  #Columns = 2
            |  |  Parallel Scan
            |  |  Relation Scan
            |  |  |  Prefetch: Eligible
            |  |  Lock Intents
            |  |  |  Table: Intent Share
            |  |  |  Row  : Next Key Share
(   11)     |  Insert Into Asynchronous Local Table Queue  ID = q4
(   11)     Access Local Table Queue  ID = q4  #Columns = 2
(   10)     Insert Into Asynchronous Table Queue  ID = q3
            |  Hash to Specific Node
            |  Rows Can Overflow to Temporary Tables

        Subsection #3:
(    8)     Process Using 4 Subagents
(    9)     |  Access Table Name = DOOLE.EMPLOYEE  ID = 2,5
            |  |  #Columns = 3
            |  |  Parallel Scan
            |  |  Relation Scan
            |  |  |  Prefetch: Eligible
            |  |  Lock Intents
            |  |  |  Table: Intent Share
            |  |  |  Row  : Next Key Share
(    8)     |  Insert Into Asynchronous Local Table Queue  ID = q6
(    8)     Access Local Table Queue  ID = q6  #Columns = 3
(    7)     Insert Into Asynchronous Table Queue  ID = q5
            |  Hash to Specific Node
            |  Rows Can Overflow to Temporary Tables

End of section


Optimizer Plan:
```



This plan is similar to that in the third example, except that multiple subagents execute each subsection. Also, at the end of each subsection, a local table queue gathers the results from all of the subagents before the qualifying rows are inserted into the second table queue to be hashed to a specific node.

Joins

A *join* is the process of combining information from two or more tables based on some common domain of information. Rows from one table are paired with rows from another table when information in the corresponding rows match on the joining criterion.

For example, consider the following two tables:

Table1		Table2	
PROJ	**PROJ_ID**	**PROJ_ID**	**NAME**
A	1	1	Sam
B	2	3	Joe
C	3	4	Mary
D	4	1	Sue
		2	Mike

To join Table1 and Table2 where the ID columns have the same values, use the following SQL statement:

```
SELECT PROJ, x.PROJ_ID, NAME
  FROM TABLE1 x, TABLE2 y
  WHERE x.PROJ_ID = y.PROJ_ID
```

This query yields the following set of result rows:

PROJ	PROJ_ID	NAME
A	1	Sam
A	1	Sue
B	2	Mike
C	3	Joe
D	4	Mary

Depending on the existence of a join predicate, as well as various costs involved as determined by table and index statistics, the optimizer chooses one of the following join methods:

- Nested-loop join
- Merge join
- Hash join

When two tables are joined, one table is selected as the outer table and the other as the inner. The outer table is accessed first and is scanned only once. Whether the inner table is scanned multiple times depends on the type of join and the indexes that are present. Even if a query joins more than two tables, the optimizer joins only two tables at a time. If necessary, temporary tables are created to hold intermediate results.

You can provide explicit join operators, such as INNER or LEFT OUTER JOIN to determine how tables are used in the join. Before you alter a query in this way, however, you should allow the optimizer to determine how to join the tables. Then analyze query performance to decide whether to add join operators.

Database database partition group impact on query optimization

In partitioned database environments, the optimizer recognizes collocation of tables and uses this collocation when it determines the best access plan for a query. If tables are frequently involved in join queries, they should be divided among database partitions in a partitioned database environment so that the rows from each table being joined are located on the same database partition. During the join operation, the collocation of the data from both joined tables prevents moving data from one database partition to another. Place both tables in the same database partition group to ensure that the data from the tables is collocated.

In a partitioned database environment, depending on the size of the table, spreading data over more database partitions reduces the estimated time (or cost) to execute a query. The number of tables, the size of the tables, the location of the data in those tables, and the type of query, such as whether a join is required, all affect the cost of the query.

Join strategies in partitioned databases

In some ways, join strategies are different in a partitioned database environment than in a non-partitioned database environment. Additional techniques can be applied to standard join methods to improve performance.

One consideration for those tables involved in frequent joins in a partitioned database environment is that of table collocation. Table collocation provides the means in a partitioned database environment to locate data from one table with

the data from another table at the same database partition based on the same distribution key. Once collocated, data to be joined can participate in a query without having to be moved to another database partition as part of the query activity. Only the answer set for the join is moved to the coordinator node.

Table Queues

The descriptions of join techniques in a partitioned database environment use the following terminology:

- Table queue (sometimes referred to as *TQ*)

  A mechanism for transferring rows between database partitions, or between processors in a single partition database.
- Directed table queue (sometimes referred to as *DTQ*)

  A table queue in which rows are hashed to one of the receiving database partitions.
- Broadcast table queue (sometimes referred to as *BTQ*)

  A table queue in which rows are sent to all of the receiving database partitions, but are not hashed.

A table queue is used in the following circumstances:

- To pass table data from one database partition to another when using inter-partition parallelism
- To pass table data within a database partition when using intra-partition parallelism
- To pass table data within a database partition when using a single partition database.

Each table queue is passes the data in a single direction. The compiler decides where table queues are required, and includes them in the plan. When the plan is executed, the connections between the database partitions initiate the table queues. The table queues close as processes end.

There are several types of table queues:

- *Asynchronous table queues.* These table queues are known as asynchronous because they read rows in advance of any FETCH being issued by the application. When the FETCH is issued, the row is retrieved from the table queue.

  Asynchronous table queues are used when you specify the FOR FETCH ONLY clause on the SELECT statement. If you are only fetching rows, the asynchronous table queue is faster.
- *Synchronous table queues.* These table queues are known as synchronous because they read one row for each FETCH that is issued by the application. At each database partition, the cursor is positioned on the next row to be read from that database partition.

  Synchronous table queues are used when you do not specify the FOR FETCH ONLY clause on the SELECT statement. In a partitioned database environment, if you are updating rows, the database manager will use the synchronous table queues.
- *Merging table queues.*

  These table queues preserve order.
- *Non-merging table queues.*

These table queues are also known as “regular” table queues. They do not preserve order.

- *Listener table queues (sometimes referred to as LTQ)*

  These table queues are use with correlated subqueries. Correlation values are passed down to the subquery and the results are passed back up to the parent query block using this type of table queue.

Join methods in partitioned database environments

The following figures illustrate join methods in a partitioned database environment.

Note: In the diagrams q1, q2, and q3 refer to table queues in the examples. The tables that are referenced are divided across two database partitions for the purpose of these scenarios. The arrows indicate the direction in which the table queues are sent. The coordinator node is database partition 0.

Collocated Joins

A collocated join occurs locally on the database partition where the data resides. The database partition sends the data to the other database partitions after the join is complete. For the optimizer to consider a collocated join, the joined tables must be collocated, and all pairs of the corresponding distribution key must participate in the equality join predicates.

The following figure provides an example.

Note: Replicated materialized query tables enhance the likelihood of collocated joins.

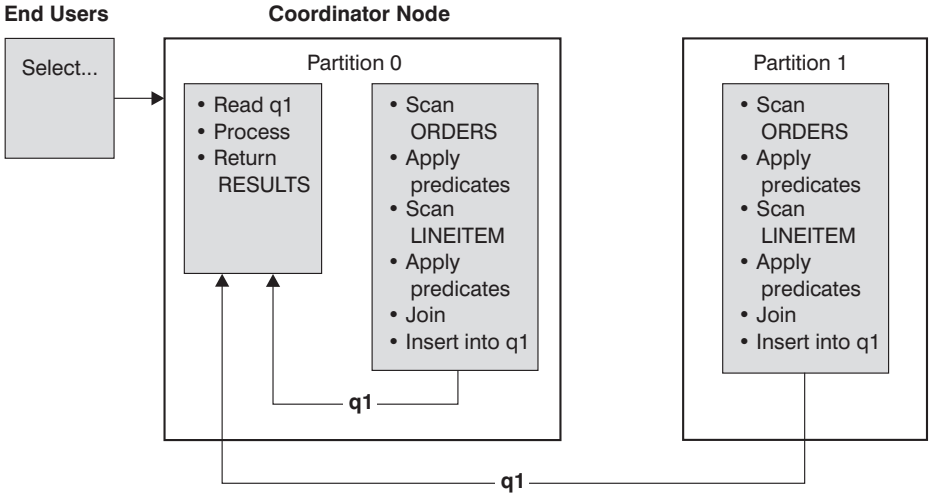


Figure 54. Collocated Join Example

Broadcast Outer-Table Joins

Broadcast outer-table joins are a parallel join strategy that can be used if there are no equality join predicates between the joined tables. It can also be used in other situations in which it is the most cost-effective join method. For example, a broadcast outer-table join might occur when there is one very large table and one very small table, neither of which is split on the join predicate columns. Instead of splitting both tables, it might be cheaper to broadcast the smaller table to the larger table. The following figures provide an example.

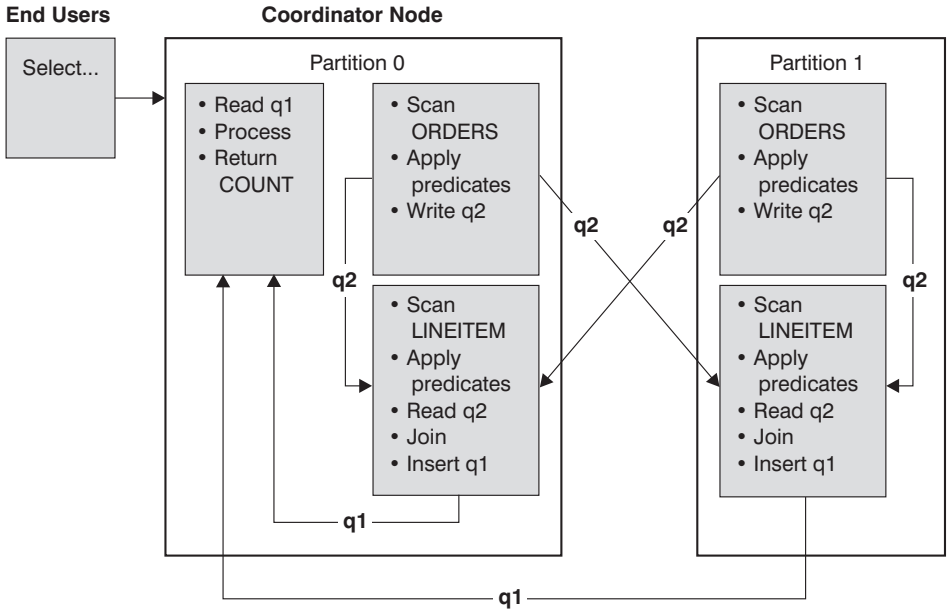
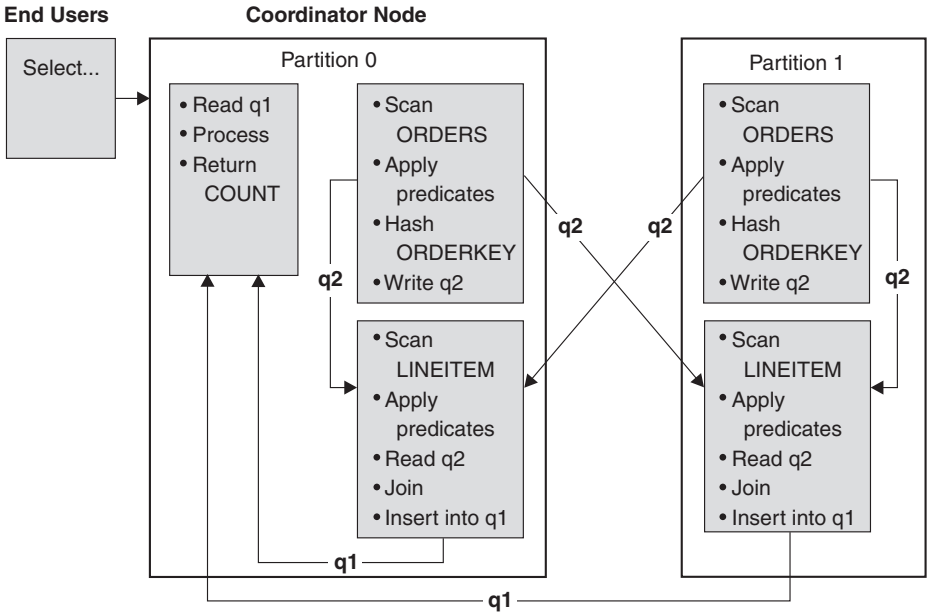


Figure 55. Broadcast Outer-Table Join Example

Directed Outer-Table Joins

In the directed outer-table join strategy, each row of the outer table is sent to one portion of the inner table, based on the splitting attributes of the inner table. The join occurs on this database partition. The following figure provides an example.

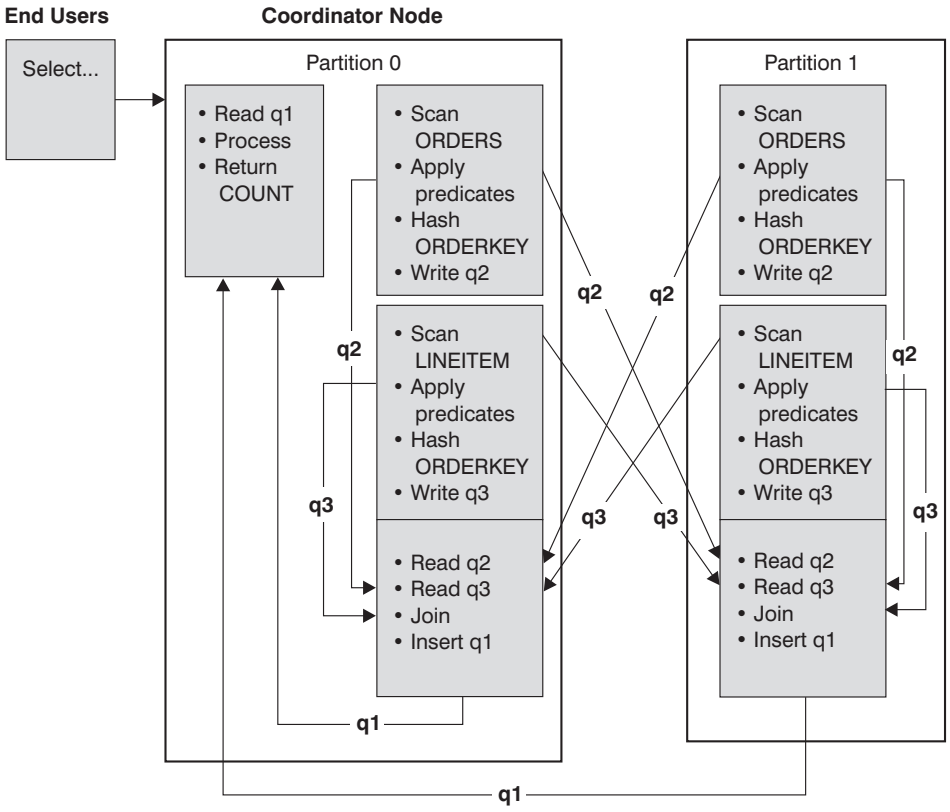


The LINEITEM table is partitioned on the ORDERKEY column.
The ORDERS table is partitioned on a different column.
The ORDERS table is hashed and sent to the correct LINEITEM table database partition.
In this example, the join predicate is assumed to be:
ORDERS.ORDERKEY = LINEITEM.ORDERKEY.

Figure 56. Directed Outer-Table Join Example

Directed Inner-Table and Outer-Table Joins

In the directed inner-table and outer-table join strategy, rows of both the outer and inner tables are directed to a set of database partitions, based on the values of the joining columns. The join occurs on these database partitions. The following figure provides an example. An example is shown in the following figure.

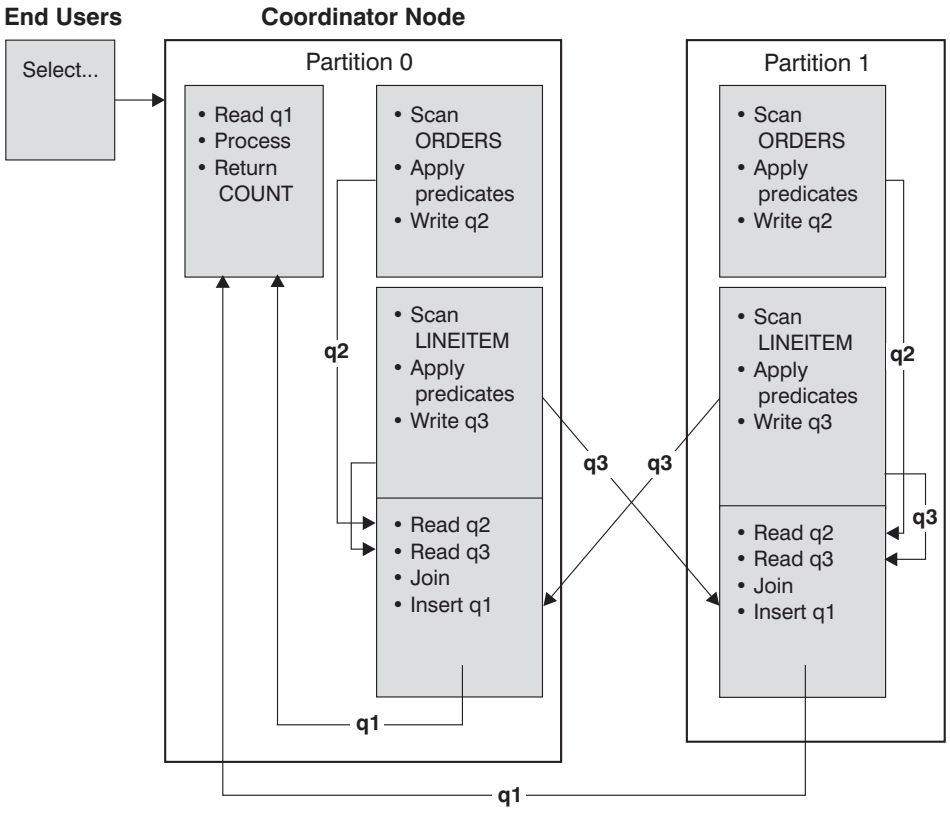


Neither table is partitioned on the ORDERKEY column.
Both tables are hashed and are sent to new database partitions where they are joined.
Both table queue q2 and q3 are directed.
In this example, the join predicate is assumed to be:
ORDERS.ORDERKEY = LINEITEM.ORDERKEY

Figure 57. Directed Inner-Table and Outer-Table Join Example

Broadcast Inner-Table Joins

In the broadcast inner-table join strategy, the inner table is broadcast to all the database partitions of the outer join table. The following figure provides an example.

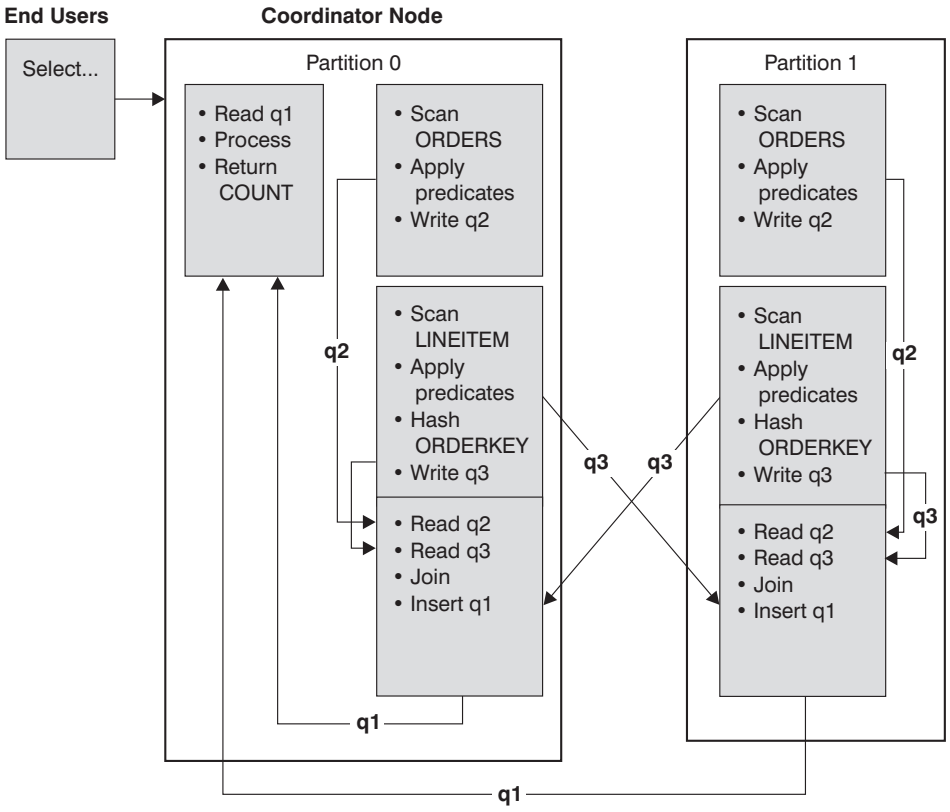


The LINEITEM table is sent to all database partitions that have the ORDERS table. Table queue q3 is broadcast to all database partitions of the outer table.

Figure 58. Broadcast Inner-Table Join Example

Directed Inner-Table Joins

With the directed inner-table join strategy, each row of the inner table is sent to one database partition of the outer join table, based on the splitting attributes of the outer table. The join occurs on this database partition. The following figure provides an example.



The ORDERS table is partitioned on the ORDERKEY column.
The LINEITEM table is partitioned on a different column.
The LINEITEM table is hashed and sent to the correct ORDERS table database partition.
In this example, the join predicate is assumed to be:
ORDERS.ORDERKEY = LINEITEM.ORDERKEY.

Figure 59. Directed Inner-Table Join Example

Replicated materialized query tables in partitioned database environments

Replicated materialized query tables improve performance of frequently executed joins in a partitioned database environment by allowing the database to manage precomputed values of the table data.

Consider an example of a query and a replicated materialized table. The following assumptions are made:

- The SALES table is in the multipartition table space REGIONTABLESPACE, and is split on the REGION column.
- The EMPLOYEE and DEPARTMENT tables are in a single-partition database partition group.

Create a replicated materialized query table based on the information in the EMPLOYEE table.

```
CREATE TABLE R_EMPLOYEE
  AS (
      SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT
      FROM   EMPLOYEE
```

```
    )
DATA INITIALLY DEFERRED REFRESH IMMEDIATE
IN REGIONTABLESPACE
REPLICATED;
```

To update the content of the replicated materialized query table, run the following statement:

```
REFRESH TABLE R_EMPLOYEE;
```

Note: After using the REFRESH statement, you should run RUNSTATS on the replicated table as you would any other table.

The following example calculates sales by employee, the total for the department, and the grand total:

```
SELECT d.mgrno, e.empno, SUM(s.sales)
FROM   department AS d, employee AS e, sales AS s
WHERE  s.sales_person = e.lastname
   AND e.workdept = d.deptno
GROUP BY ROLLUP(d.mgrno, e.empno)
ORDER BY d.mgrno, e.empno;
```

Instead of using the EMPLOYEE table, which is on only one database partition, the database manager uses the R_EMPLOYEE table, which is replicated on each of the database partitions where the SALES tables is stored. The performance enhancement occurs because the employee information does not have to be moved across the network to each database partition to calculate the join.

Replicated materialized query tables in collocated joins

Replicated materialized query tables can also assist in the collocation of joins. For example, if a star schema contains a large fact table spread across twenty nodes, the joins between the fact table and the dimension tables are most efficient if these tables are collocated. If all of the tables are in the same database partition group, at most one dimension table is partitioned correctly for a collocated join. The other dimension tables cannot be used in a collocated join because the join columns on the fact table do not correspond to the distribution key of the fact table.

Consider a table called FACT (C1, C2, C3, ...) split on C1; and a table called DIM1 (C1, dim1a, dim1b, ...) split on C1; and a table called DIM2 (C2, dim2a, dim2b, ...) split on C2; and so on.

In this case, you see that the join between FACT and DIM1 is perfect because the predicate DIM1.C1 = FACT.C1 is collocated. Both of these tables are split on column C1.

However, the join between DIM2 with the predicate WHERE DIM2.C2 = FACT.C2 cannot be collocated because FACT is split on column C1 and not on column C2. In this case, you might replicate DIM2 in the database partition group of the fact table so that the join occurs locally on each database partition.

Note: The replicated materialized query tables discussion here is related to intra-database replication. Inter-database replication is concerned with subscriptions, control tables, and data located in different databases and on different operating systems.

When you create a replicated materialized query table, the source table can be a single-node table or a multi-node table in a database partition group. In most

cases, the replicated table is small and can be placed in a single-node database partition group. You can limit the data to be replicated by specifying only a subset of the columns from the table or by specifying the number of rows through the predicates used, or by using both methods. The data capture option is not required for replicated materialized query tables to function.

A replicated materialized query table can also be created in a multi-node database partition group so that copies of the source table are created on all of the database partitions. Joins between a large fact table and the dimension tables are more likely to occur locally in this environment that if you broadcast the source table to all database partitions.

Indexes on replicated tables are not created automatically. You can create indexes that are different from those on the source table. However, to prevent constraint violations that are not present on the source tables, you cannot create unique indexes or put constraints on the replicated tables. Constraints are disallowed even if the same constraint occurs on the source table.

Replicated tables can be referenced directly in a query, but you cannot use the NODENUMBER() predicate with a replicated table to see the table data on a particular partition.

Use the EXPLAIN facility to see if a replicated materialized query table was used by the access plan for a query. Whether the access plan chosen by the optimizer uses the replicated materialized query table depends on the information that needs to be joined. The optimizer might not use the replicated materialized query table if the optimizer determines that it would be cheaper to broadcast the original source table to the other database partitions in the database partition group.

Lesson 4. Improving an access plan in a partitioned database environment

You will learn how the access plan and related windows for the basic query change when you perform various tuning activities.

Using a series of examples, accompanied by illustrations, you will learn how the estimated total cost for the access plan of even a simple query can be improved by using the **runstats** command and adding appropriate indexes.

As you gain experience with Visual Explain, you will discover other ways to tune queries.

Working with access plan graphs

Using the four sample explain snapshots as examples, you will learn how tuning is an important part of database performance.

The queries associated with the explain snapshots are number 1 - 4. Each query uses the same SQL or XQuery statement (described in Lesson 1):

```
SELECT S.ID,SNAME,O.DEPTNAME,SALARY+COMM
FROM ORG O, STAFF S
WHERE
 O.DEPTNUMB = S.DEPT AND
 S.JOB <> 'Mgr' AND
```

```
 S.SALARY+S.COMM > ALL ( SELECT ST.SALARY*.9
                         FROM STAFF ST
                         WHERE ST.JOB='Mgr' )
ORDER BY S.NAME
```

But each iteration of the query uses more tuning technics than the previous execution. For example, Query 1 has had no performance tuning, while Query 4 has had the most. The differences in the queries are described below:

Query 1
Running a query with no indexes and no statistics
Query 2
Collecting current statistics for the tables and indexes in a query
Query 3
Creating indexes on columns used to join tables in a query
Query 4
Creating additional indexes on table columns

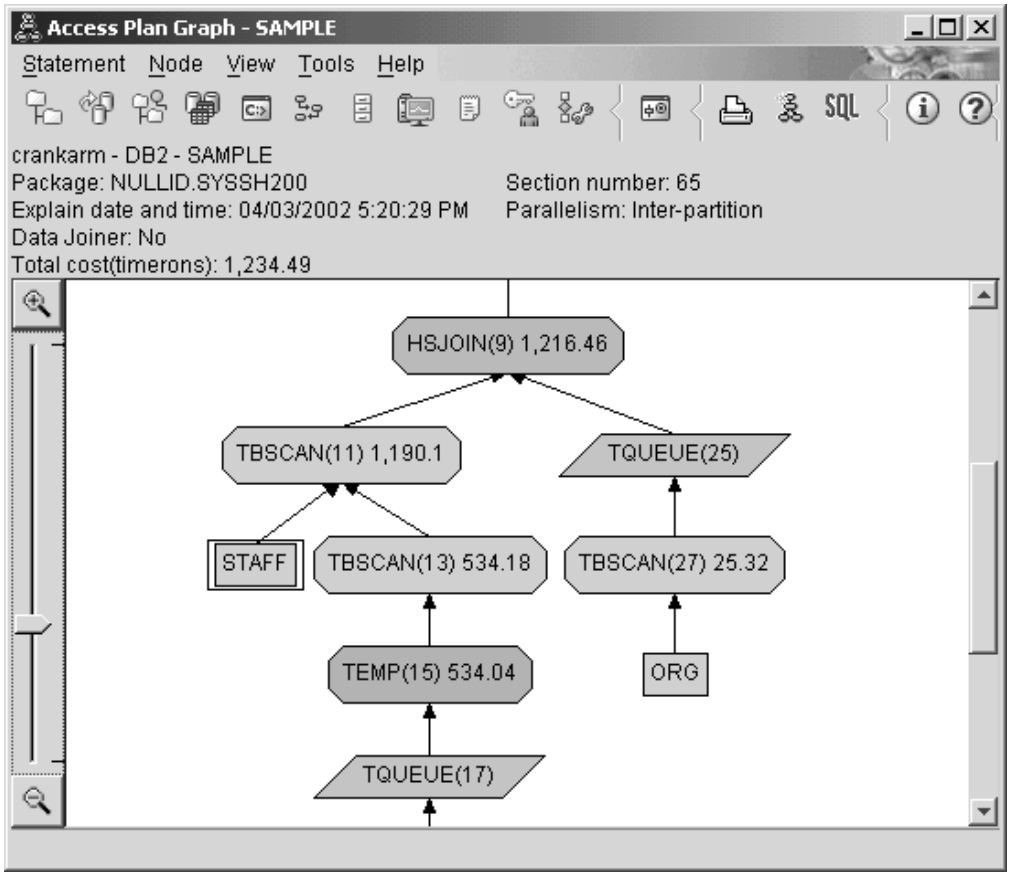
These examples were produced on an RS/6000 SP machine with 7 physical nodes using inter-partition parallelism.

Running a query with no indexes and no statistics in a partitioned database environment

In this example, the access plan was created for the SQL query with no indexes and no statistics.

To view the access plan graph for this query (Query 1):

1. In the Control Center, expand the object tree until you find the SAMPLE database.
2. Right-click the database and select **Show explained statements history** from the pop-up menu.
   The Explained Statements History window opens.
3. Double-click the entry identified as Query Number 1 (you might need to scroll to the right to find the **Query Number** column).
   The Access Plan Graph window for the statement opens.



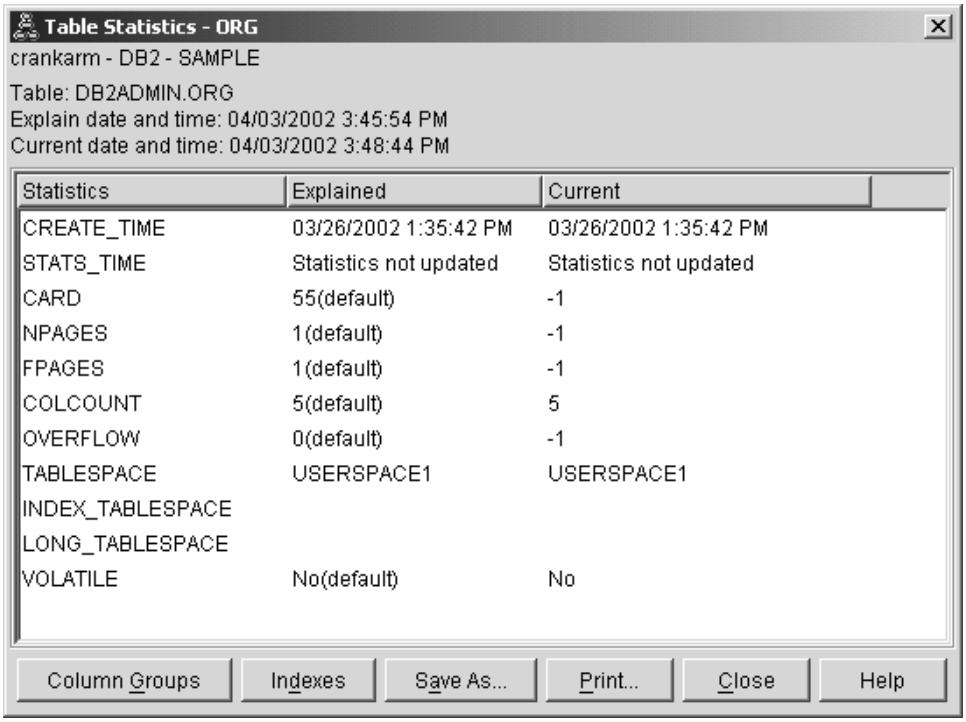
Answering the following questions will help you understand how to improve the query:

1. Do current statistics exist for each table in the query?

   To check if current statistics exist for each table in the query, double-click each table node in the access plan graph. In the corresponding Table Statistics window that opens, the **STATS_TIME** row under the **Explained** column contains the words "Statistics not updated" indicating that no statistics had been collected at the time when the snapshot was created.

   If current statistics do not exist, the optimizer uses default statistics, which might differ from the actual statistics. Default statistics are identified by the word "default" under the **Explained** column in the Table Statistics window.

   According to the information in the Table Statistics window for the ORG table, the optimizer used default statistics (as indicated next to the explained values). Default statistics were used because actual statistics were not available when the snapshot was created (as indicated in the **STATS_TIME** row).



2. Does this access plan use the most effective methods of accessing data?

   This access plan contains table scans, not index scans. Table scans are shown as octagons and are labeled TBSCAN operator. If Index scans had been used they would appear as diamonds and be labeled IXSCAN. The use of an index that was created for a table is more cost-effective than a table scan if small amounts of data are being extracted.

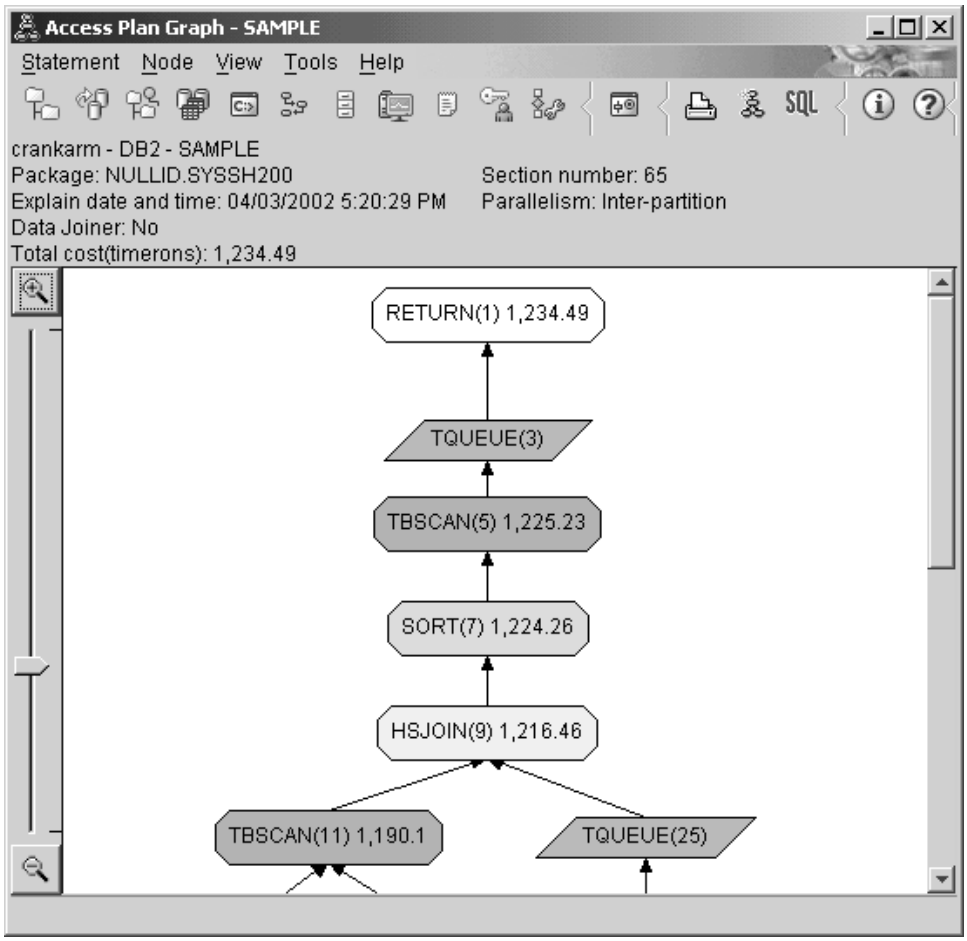
3. How effective is this plan?

   You can determine the effectiveness of an access plan only if it is based on actual statistics. Since the optimizer used default statistics in the access plan, you cannot determine how effective the plan is.

   In general, you should make note of the total estimated Cost for the access plan for later comparison with revised access plans. The cost listed in each node is cumulative, from the first steps of your query up to and including the node.

   Note: For partitioned databases, this is the cumulative cost for the node that uses the most resources.

   In the Access Plan Graph window, the total cost is approximately 1,234 timerons, shown in **RETURN (1)** at the top of the graph. The total estimated cost is also shown in the top area of the window.



What's Next

Moving on to query 2.

Query 2 looks at an access plan for the basic query after runstats has been run. Using the runstats command provides the optimizer with current statistics on all tables accessed by the query.

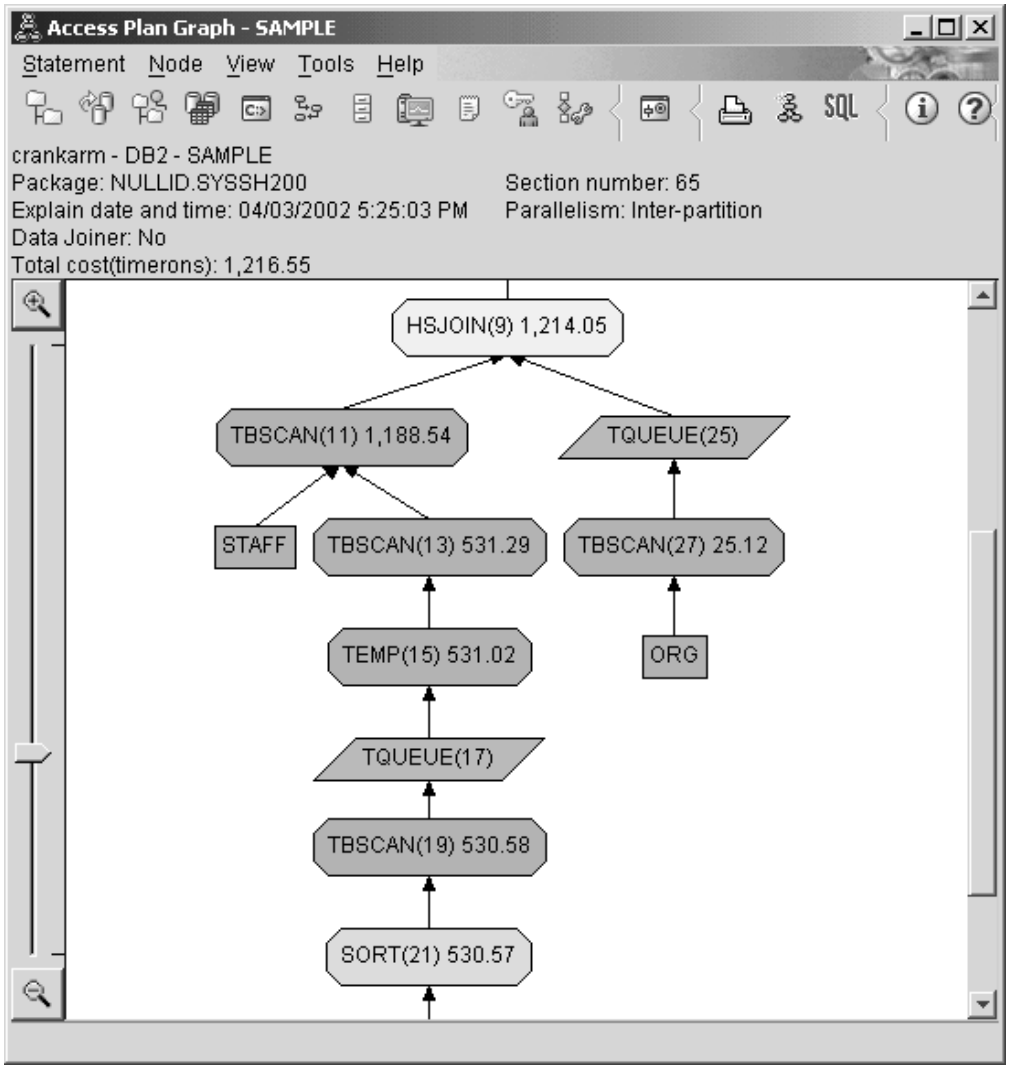
Collecting current statistics for the tables and indexes using runstats in a partitioned database environment

This example builds on the access plan described in Query 1 by collecting current statistics with the runstats command.

It is highly recommended that you use the runstats command to collect current statistics on tables and indexes, especially if significant update activity has occurred or new indexes have been created since the last time the runstats command was executed. This provides the optimizer with the most accurate information with which to determine the best access plan. If current statistics are not available, the optimizer can choose an inefficient access plan based on inaccurate default statistics.

Be sure to use runstats *after* making your table updates; otherwise, the table might appear to the optimizer to be empty. This problem is evident if cardinality on the Operator Details window equals zero. In this case, complete your table updates, rerun the runstats command, and recreate the explain snapshot for affected tables.

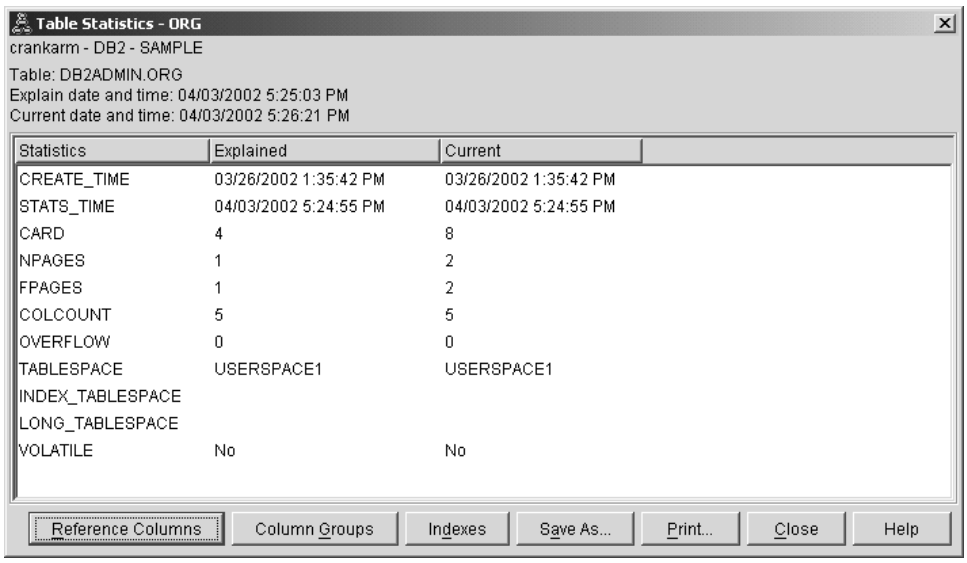
To view the access plan graph for this query (Query 2): in the Explain Statements History window, double-click the entry identified as Query Number 2. The Access Plan Graph window for this execution of the statement opens.



Answering the following questions will help you understand how to improve the query.

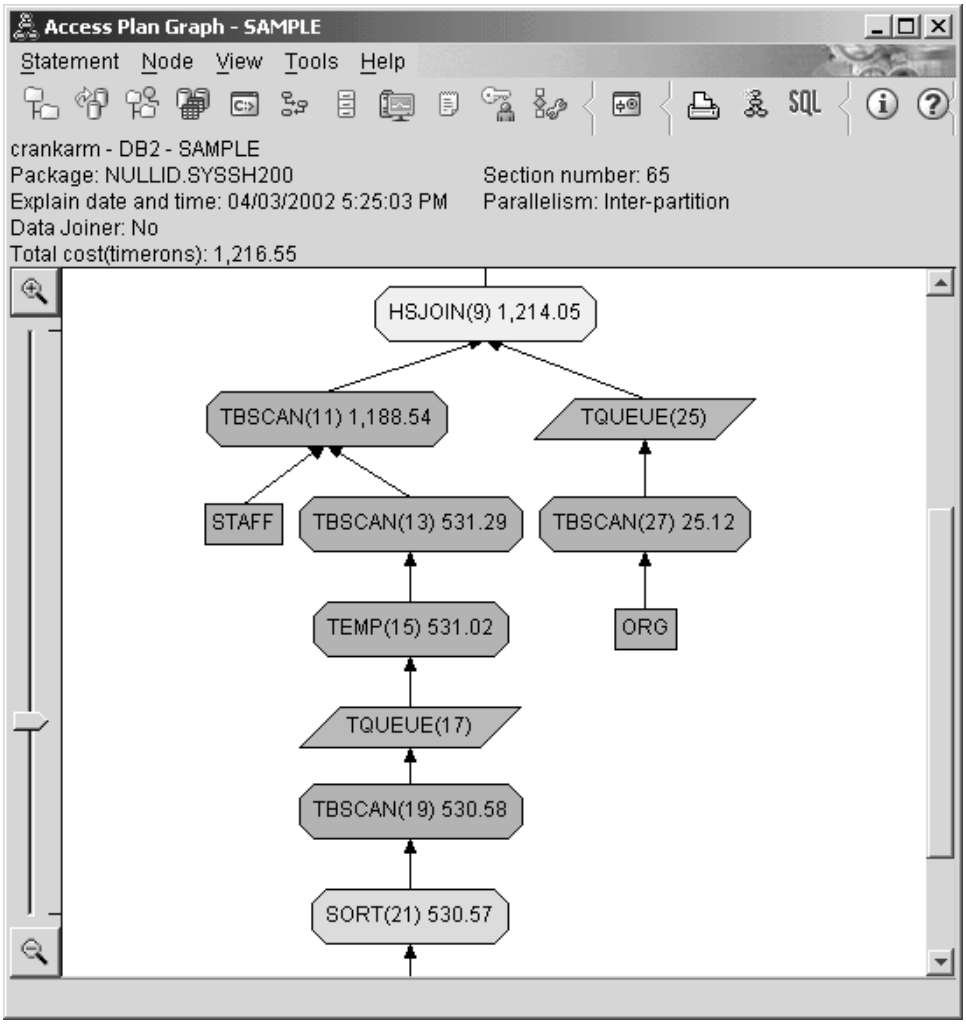
1. Do current statistics exist for each table in the query?

   The Table Statistics window for the ORG table shows that the optimizer used actual statistics (the STATS_TIME value is the actual time that the statistics were collected). The accuracy of the statistics depends on whether there were significant changes to the contents of the tables since the runstats command was run.

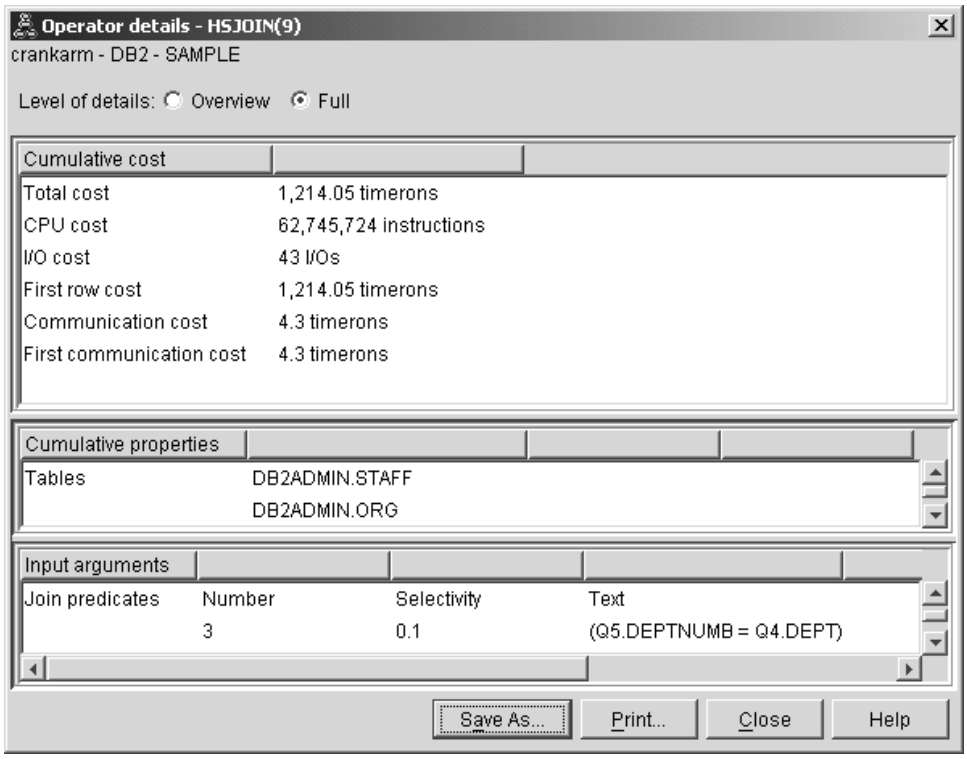


2. Does this access plan use the most effective methods of accessing data?

   Like Query 1, the access plan in Query 2 uses table scans (TBSCAN operator) not index scans (IXSCAN). Even though current statistics exist, an index scan was not done because there are no indexes on the columns that were used by the query. One way to improve the query would be to provide the optimizer with indexes on columns that are used to join tables (that is, on columns that are used in join Predicates). In this example, this is the first merge scan join: HSJOIN (9).



In the Operator Details window for the HSJOIN (9) operator, look at the **Join predicates** section under **Input arguments**. The columns used in this join operation are listed under the **Text** column. in this example, these columns are DEPTNUMB and DEPT.



3. How effective is this access plan?

   Access plans based on up-to-date statistics always produce a realistic estimated cost (measured in timerons). Because the estimated cost in Query 1 was based on default statistics, the cost of the two access plan graphs cannot be compared to determine which one is more effective. Whether the cost is higher or lower is not relevant. You must compare the cost of access plans that are based on actual statistics to get a valid measurement of effectiveness.

What's Next

Moving on to query 3.

Query 3 looks at the effects of adding indexes on the DEPTNUMB and DEPT columns. Adding indexes on the columns that are used in join predicates can improve performance.

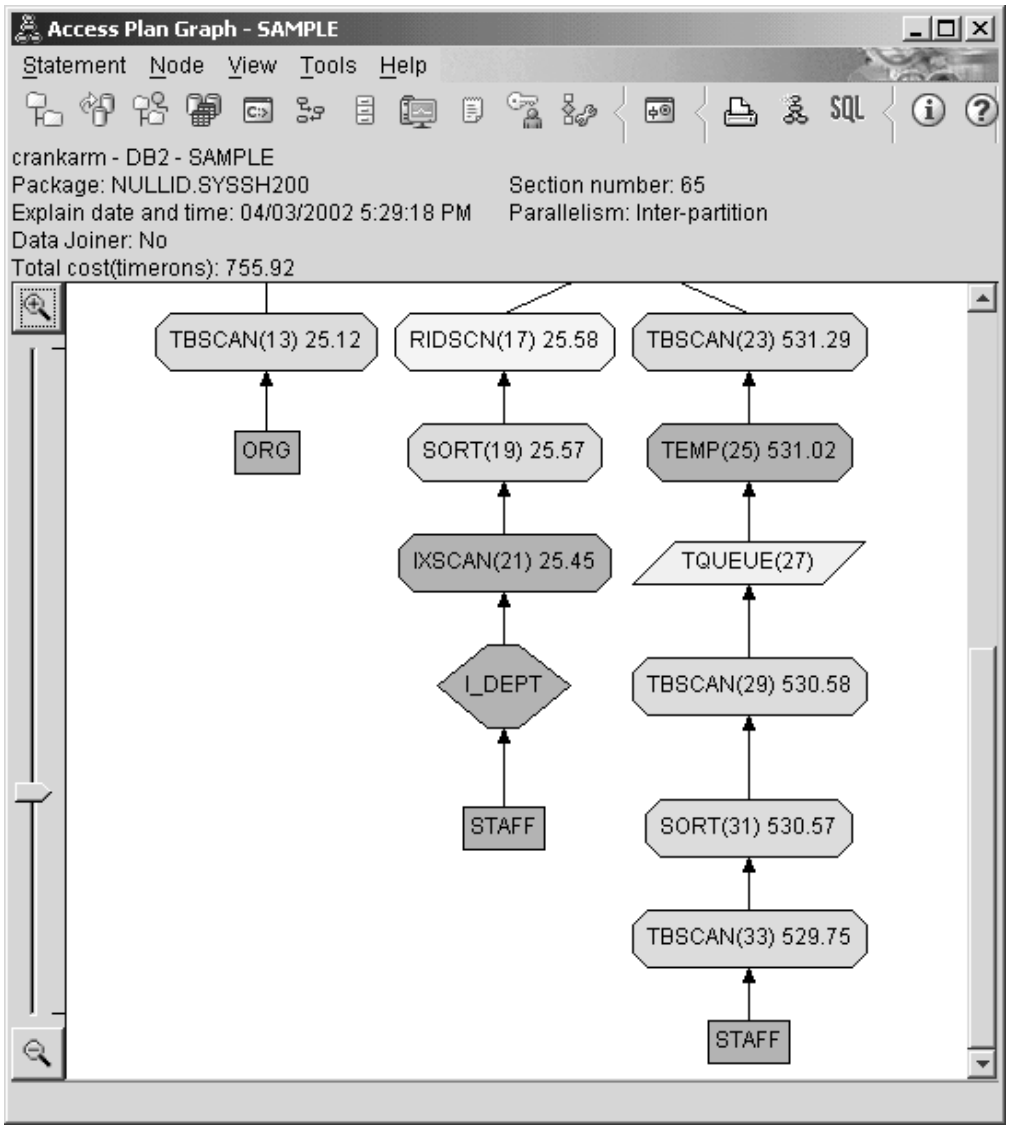
Creating indexes on columns used to join tables in a query in a partitioned database environment

This example builds on the access plan described in Query 2 by creating indexes on the DEPT column on the STAFF table and on the DEPTNUMB column on the ORG table.

Note: Recommended indexes can be created using the Design Advisor.

To view the access plan graph for this query (Query 3): in the Explained Statements History window, double-click the entry identified as Query Number 3. The Access Plan Graph window for this execution of the statement opens.

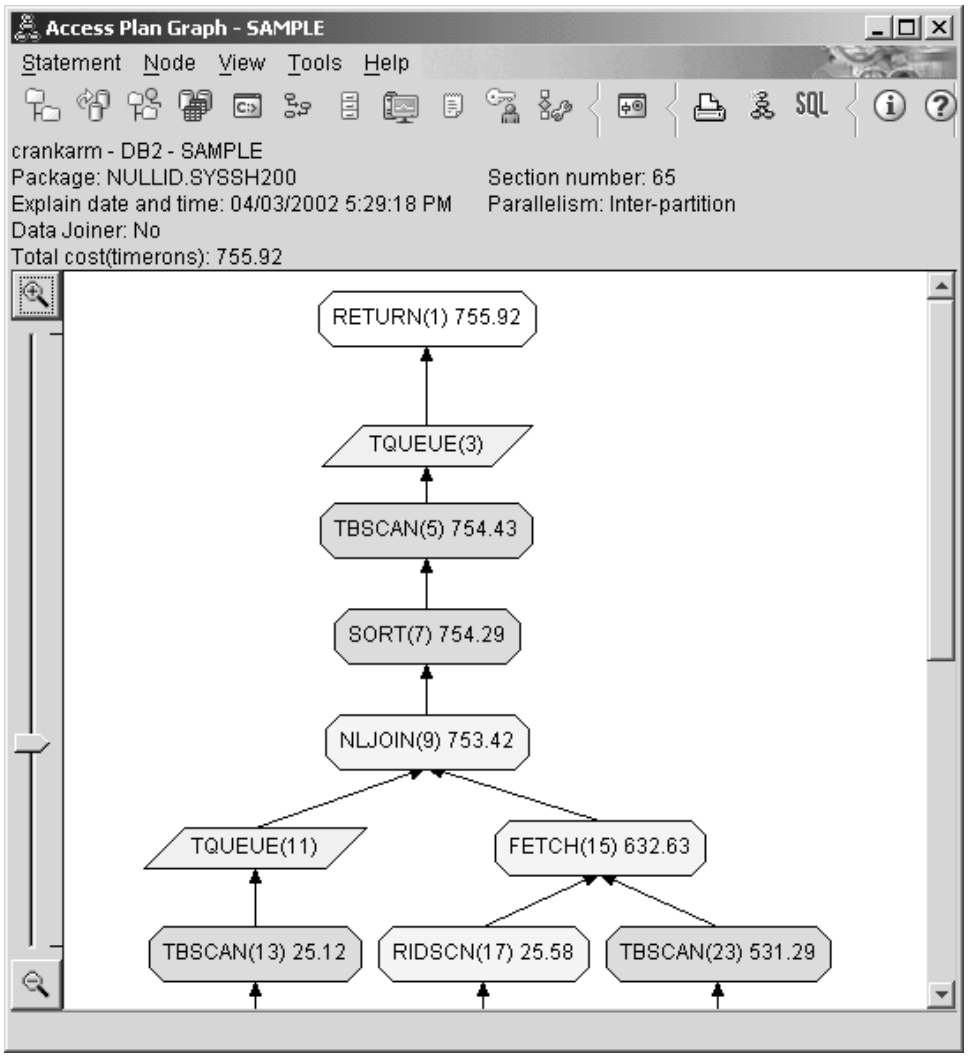
Note: Even though an index was created for DEPTNUM, the optimizer did not use it.



Answering the following questions will help you understand how to improve the query.

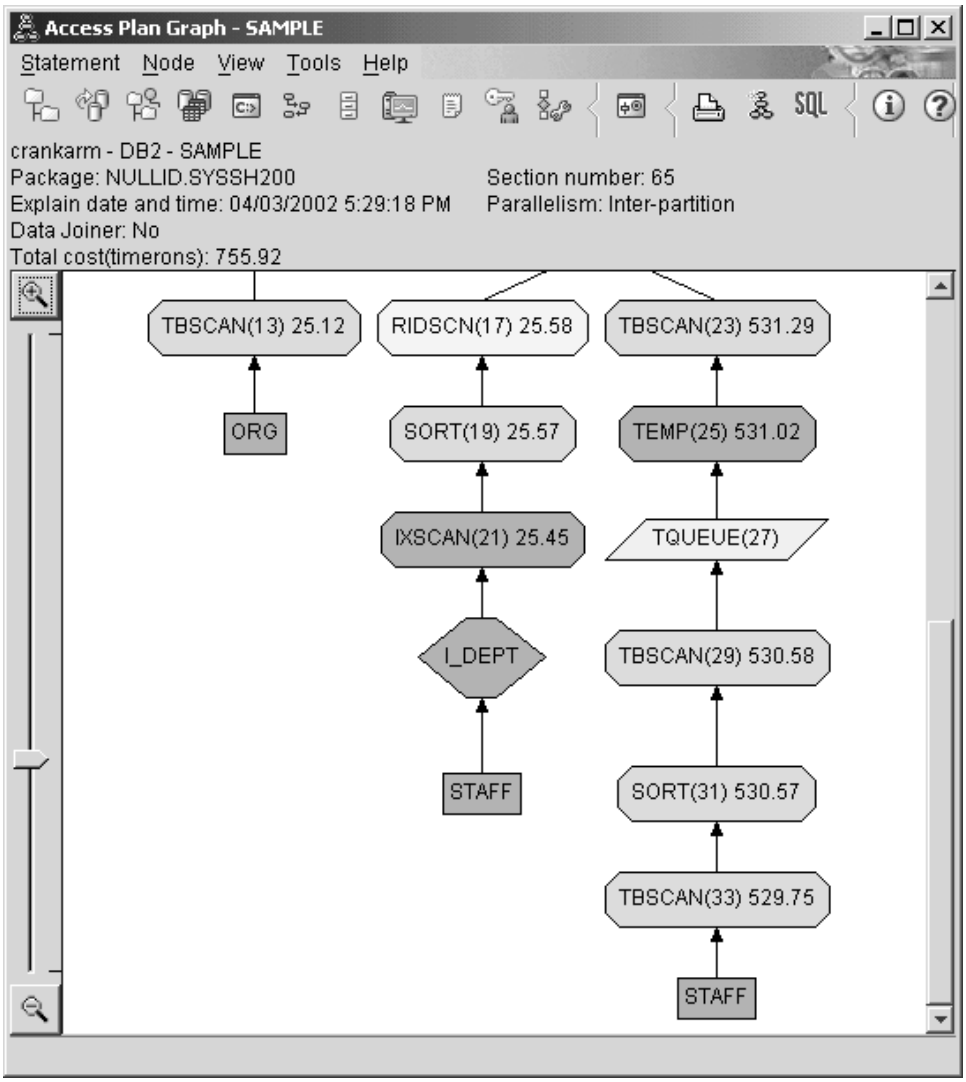
1. What has changed in the access plan with indexes?

   A new diamond-shaped node, **I_DEPT**, has been added just above the STAFF table. This node represents the index that was created on DEPT, and it shows that the optimizer used an index scan instead of a table scan to determine which rows to retrieve.

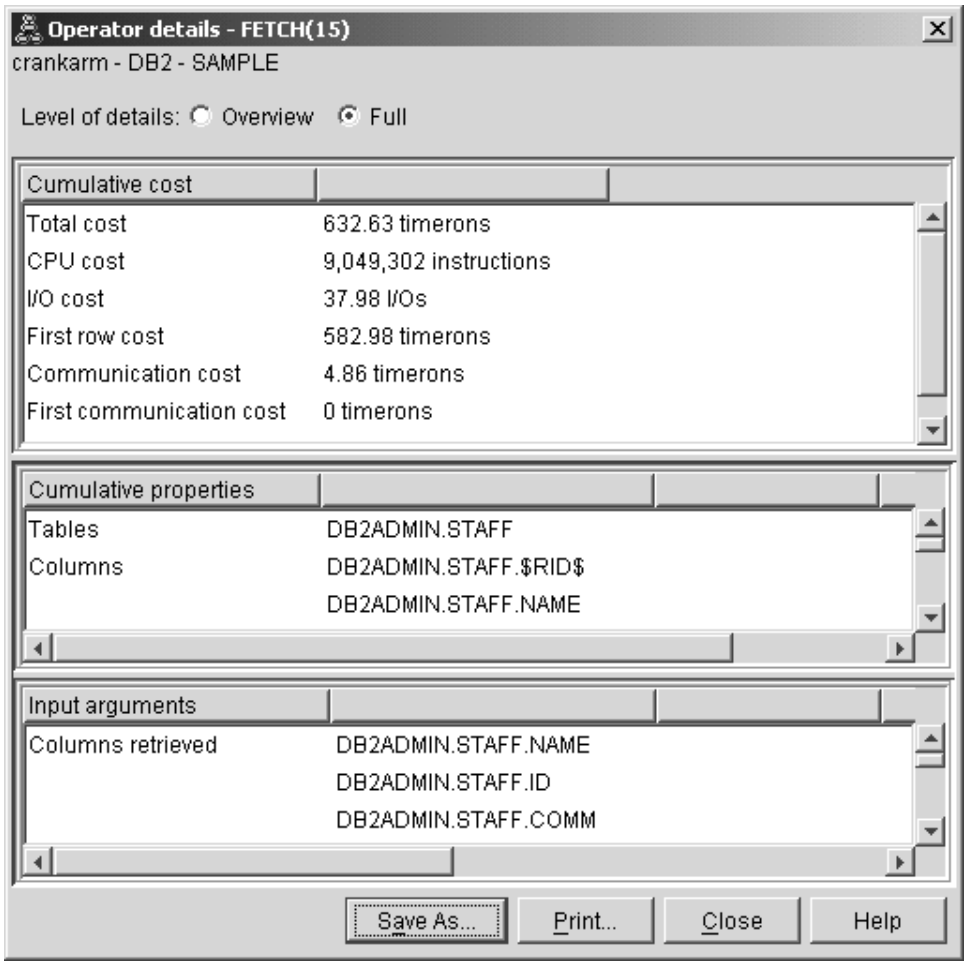


2. Does this access plan use the most effective methods of accessing data?

   The access plan for this query shows the effect of creating indexes on the DEPTNUMB column of the ORG table, resulting in FETCH (15) and IXSCAN (21) and on the DEPT column of the STAFF table. Query 2 did not have this index; therefore, a table scan was used in that example.



The Operator Details window for the FETCH (15) operator shows the columns being used in this operation.



The combination of index and fetch are calculated to be less costly than the full table scans used in the previous access plans.

3. How effective is this access plan?

   This access plan is more effective than the one from the previous example. The cumulative cost has been reduced from approximately 1,214 timerons in Query 2 to approximately 755 timerons in Query 3.

What's Next

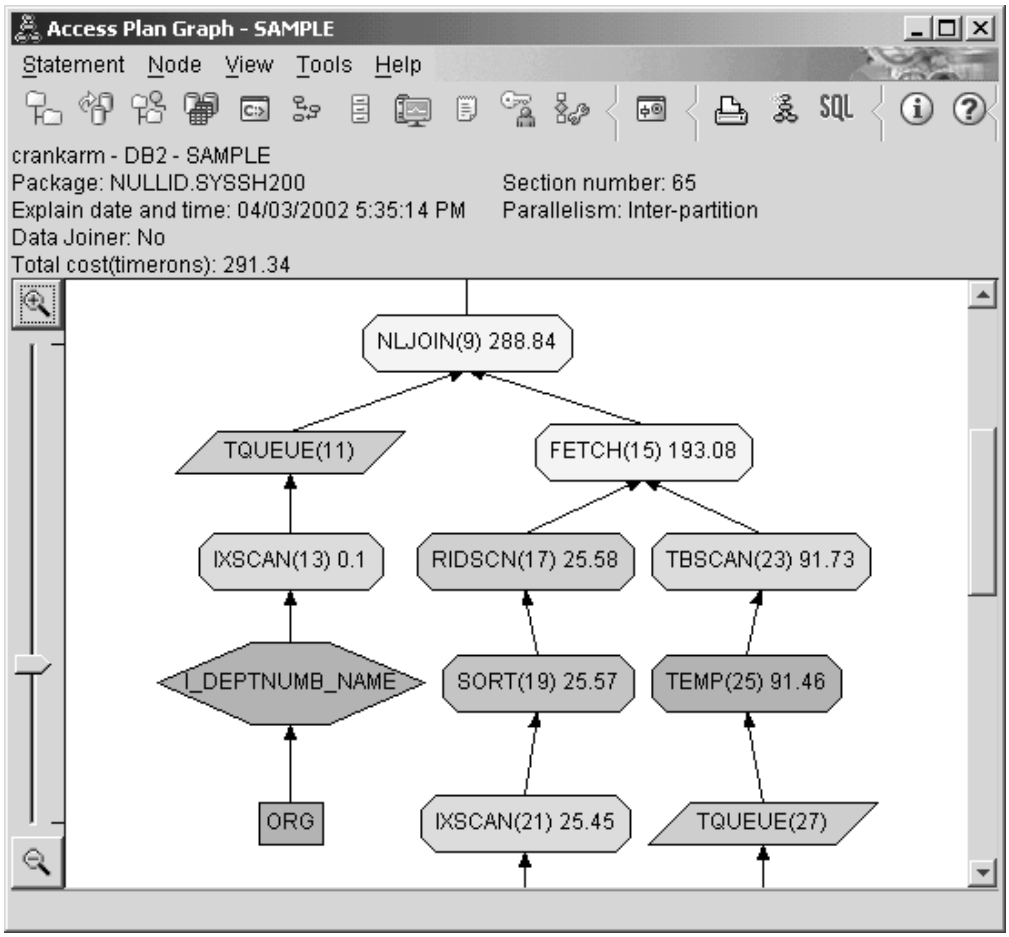
Moving on to query 4.

Query 4 reduces the fetch and index scan to a single index scan without a fetch. Creating additional indexes can reduce the estimated cost for the access plan.

Creating additional indexes on table columns in a partitioned database environment

This example builds on the access plan described in Query 3 by creating an index on the JOB column in the STAFF table, and adding DEPTNAME to the existing index in the ORG table. (Adding a separate index could cause an additional access.)

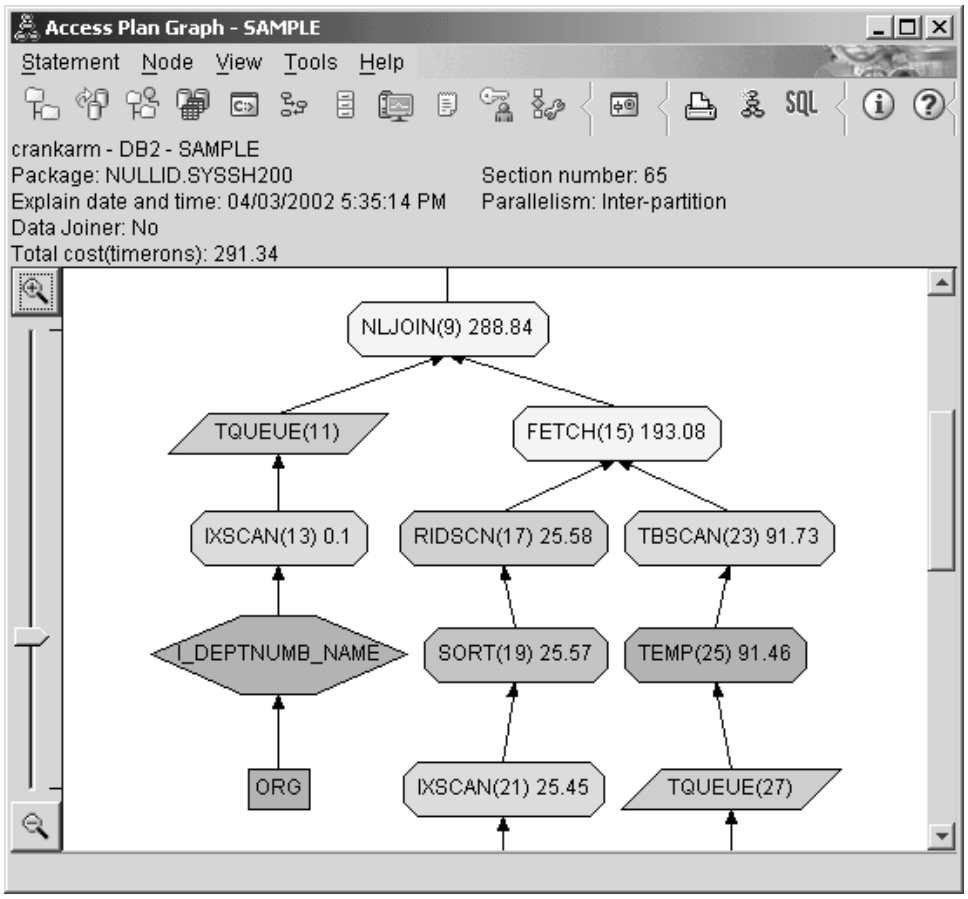
To view the access plan graph for this query (Query 4): in the Explained Statements History window, double-click the entry identified as Query Number 4. The Access Plan Graph window for this execution of the statement opens.



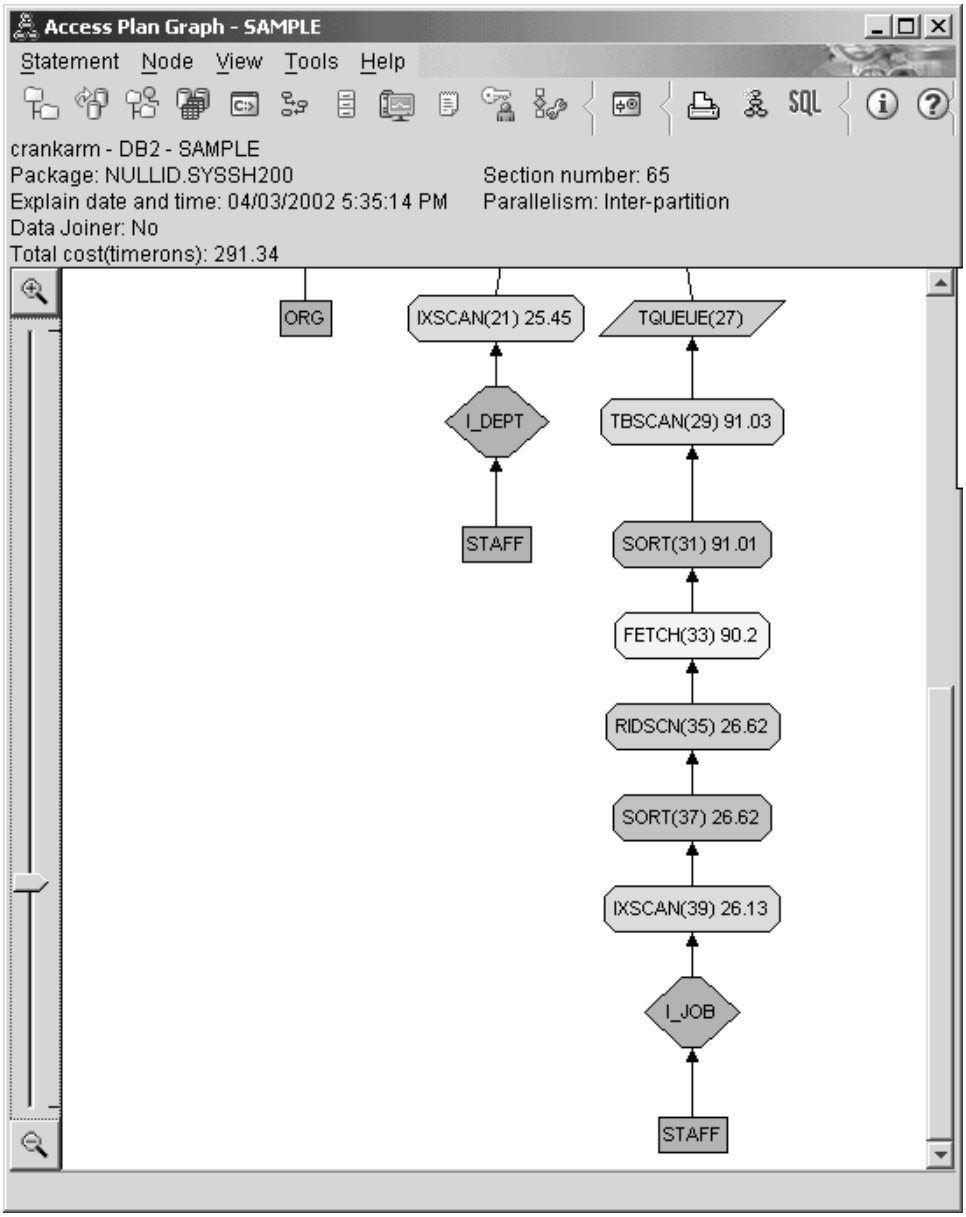
Answering the following questions will help you understand how to improve the query.

1. What changes in this process plan as a result of creating additional indexes?

   In the middle portion of the access plan graph, notice that for the ORG table, the previous table scan has been changed to an index scan, IXSCAN (7). Adding the DEPTNAME column to the index on the ORG table has allowed the optimizer to refine the access involving the table scan.



In the bottom portion of the access plan graph, note that for the STAFF table the previous index scan and fetch have been changed to an index scan only IXSCAN (39). Creating the JOB index on the STAFF table has allowed the optimizer to eliminate the extra access involving the fetch.



2. How effective is this access plan?

   This access plan is more cost effective than the one from the previous example. The cumulative cost has been reduced from approximately 753 timerons in Query 3 to approximately 288 timerons in Query 4.

What's Next

Improving the performance of your own SQL or XQuery statements.

Refer to the *DB2 Information Center* to find detailed information on additional steps that you can take to improve performance. You can then return to Visual Explain to access the impact of your actions.

Chapter 26. 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 so as to balance the usage of storage space, improve database system performance, or to meet other system requirements.

Data redistribution can be performed using one of the following interfaces:

- REDISTRIBUTE DATABASE PARTITION GROUP command
- ADMIN_CMD system-defined procedure
- STEPWISE REDISTRIBUTE_DBPG system-defined procedure
- sqludrdt 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 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 or can be explicitly defined using the CREATE TABLE or ALTER TABLE statements. By default during data redistribution, for each table within a specified nodegroup, table data is divided and redistributed evenly among the database partitions, however 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.

Restrictions on data redistribution

Restrictions on data redistribution are important to note prior to proceeding with data redistribution or when troubleshooting problems related to data redistribution.

Note: This topic contains references to some keyword updates that are only valid when DB2 9.5 Fix Pack 1 is installed. If DB2 9.5 Fix Pack 1 is not yet available or installed, refer to the DB2 9 version of this topic available in the DB2 9 Information Center at:http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp

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:

© Copyright IBM Corp. 1993, 2009

  - Starting another redistribution operation on the 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 option 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 option is specified:
  - The data changes made as part of the data redistribution are not rollforward recoverable.
  - If the database is otherwise recoverable, the table space is put into the BACKUP PENDING state after accessing the first table within the partition. 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 option 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.
- For range-partitioned tables, movement of data between ranges of a data partitioned table is restricted. Data redistribution however is supported for the movement of data between database partitions.
- For partitioned tables, redistribution of data 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 utility with the ABORT or CONTINUE option and an appropriate table list so that redistribution of the table is either completed or aborted.

Determining the current data distribution within a nodegroup and for a table

Determining the current data distribution for a nodegroup or table can be helpful in determining if data redistribution is required and can be used to create a custom distribution map that can be used to specify how data should be distributed.

If a new database partition is added to a database partition group, or an existing database partition is dropped from a database partition group, then data redistribution should be performed in order 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. Please note that, in some cases, an unequal distribution of data can be desirable. For example, if some database partitions reside on a particularly powerful machine, then it may be beneficial for those database partitions to contain larger volumes of data than other partitions.

To get information about the current distribution of data among database partitions in a 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.

If the results of the query show that the distribution of data among database partitions is not as desired, then run the following query to get the distribution of data across hash partitions:

```
SELECT PARTITION(column_name), COUNT(*) FROM table_name
      GROUP BY PARTITION(column_name)
      ORDER BY PARTITION(column_name) DESC
```

The output of this query can easily be used to construct the distribution file needed when the USING DISTFILE option of the REDISTRIBUTE DATABASE PARTITION GROUP command is specified (please refer to the Command Reference section for the REDISTRIBUTE DATABASE PARTITION GROUP command for a description of the format of the distribution file).

When the USING DISTFILE option is specified, the REDISTRIBUTE DATABASE PARTITION GROUP command will use the information in the file to generate a new partition map for the database partition group that results in a uniform distribution of data among database partitions.

If a uniform distribution is not desired, then the user can construct his or her own target partition map for the redistribution operation, which can be specified using the USING TARGETMAP option of the REDISTRIBUTE DATABASE PARTITION GROUP command.

After doing this investigation you will know if your data is uniformly distributed or not or if data redistribution is required. If the data requires redistribution, you can plan to do this during a system maintenance opportunity using one of the supported interfaces.

Redistributing data across database partitions using the REDISTRIBUTE DATABASE PARTITION GROUP command

Redistributing data can be successfully performed using the REDISTRIBUTE DATABASE PARTITION GROUP command. This is the recommended interface for performing data redistribution.

Note: This topic contains references to some keyword updates that are only valid when DB2 9.5 Fix Pack 1 is installed. If DB2 9.5 Fix Pack 1 is not yet available or installed, refer to the DB2 9 version of this topic available in the DB2 9 Information Center at: http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.udb.admin.doc/doc/t0005017.htm

Restrictions

- See: "Restrictions on data redistribution" on page 339

To redistribute data across database partitions in a database partition group using the REDISTRIBUTE DATABASE PARTITION GROUP command:

1. Perform a backup of the database. See the BACKUP command.
2. Connect to the database partition that contains the system catalog tables. See the CONNECT command.
3. Issue the REDISTRIBUTE DATABASE PARTITION GROUP command.

   Note: In previous versions of DB2, 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 evenly distributed and is to remain evenly distributed. UNIFORM is the default when no distribution-type is specified so it is also valid to omit this option if no other distribution type has been specified.

   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 should be used and the name of file that contains the target redistribution map.

For details, refer to the REDISTRIBUTE DATABASE PARTITION GROUP command-line utility information.

4. Allow the command to run uninterrupted. When the command completes, 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 option of the REDISTRIBUTE DATABASE PARTITION GROUP command is used.
   - Recreate any replicated materialized query tables dropped before redistribution.
   - If the STATISTICS NONE option of the REDISTRIBUTE DATABASE PARTITION GROUP command was specified or the NOT ROLLFORWARD RECOVERABLE option was omitted (both of which mean that the statistics were not collected during data redistribution) and there are tables in the database partition group possessing a statistics profile, execute the RUNSTATS command now to collect data distribution statistics for the SQL compiler and optimizer to use when it chooses data access plans for queries.
   - If the NOT ROLLFORWARD RECOVERABLE option 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*\

Data redistribution should have completed successfully 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 data in a database partition group

Use the Redistribute Data wizard to create an effective redistribution plan for your database partition group and redistribute your data. First you select your redistribution method and strategy, then you make more advanced choices.

To work with database partition groups, you must have *sysadm* or *dbadm* authority.

To redistribute data in your database partition group:

1. Open the Redistribute Data wizard: From the Control Center, expand the object tree until you find the **Database Partition Groups** folder. Any existing database partition groups are displayed in the contents pane on the right side of the window. Right-click on the database partition group that you want to work with and select **Redistribute** from the pop-up menu. The Redistribute Data wizard opens.

   You can also open the Redistribute Data wizard from the *Adding database partitions using the Add Partitions launchpad* or the *Dropping database partitions from the instance using the Drop Partitions launchpad*.
2. Complete each of the applicable wizard pages. Click the wizard overview link on the first page for more information. The **Finish** push button is enabled when you specify enough information for the wizard to redistribute your data.

Log space requirements for data redistribution

To successfully perform a data redistribution operation adequate log file space must be allocated before beginning the data redistribution operation to ensure that data redistribution is not interrupted.

Note: This topic contains references to some keyword updates that are only valid when DB2 9.5 Fix Pack 1 is installed. If DB2 9.5 Fix Pack 1 is not yet available or installed, refer to the DB2 9 version of this topic available in the DB2 9 Information Center at:http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp

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 REDISTRIBUTE DATABASE PARTITION GROUP command is used and the NOT ROLLFORWARD RECOVERABLE option is **not** used, or redistribution is perfomed from any other supported interface where the data redistribution is not rollforward 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 non-uniform 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 512 GB, then the data redistribution must be done in steps. The active log limit has been increased to 1024 GB in Version 9.5 Fix Pack 3. Use the ″makepmap″ utility to generate a series of target distribution maps, one for each step. You might also set the logsecond database configuration parameter to -1 to avoid most log space problems.

When the REDISTRIBUTE DATABASE PARTITION GROUP command is used and the NOT ROLLFORWARD RECOVERABLE option is used, or redistribution is performed from any other supported interface where the data redistribution is not rollforward recoverable:

- Log records are not created when rows are moved as part of data redistribution. This significantly reduces log file space requirements, however when this option is used when a rollforward recovery of the database is performed the redistribute operation log record cannot be rolled forward and any tables processed as part of the rollforward operation will remain in an UNAVAILABLE state. Please refer to the Command Reference for a discussion of the consequences of using the NOT ROLLFORWARD RECOVERABLE option.
- If the database partition group undergoing data redistribution does contain 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 (i.e., data being sent, received or both). Regardless of the presence of LF/LOB data, on receiving partitions there is one type of log record that is written for which the number of such log records does depend on the amount of data moving: extent allocation log records. However, the total space required for these log records is small, and is never more than a tiny fraction of the total user data that is moving.

Redistribution event log file

During data redistribution event logging is performed. Event information is logged to an event log file which can later be used to perform error recovery.

Note: This topic contains references to some keyword updates that are only valid when DB2 9.5 Fix Pack 1 is installed. If DB2 9.5 Fix Pack 1 is not yet available or installed, refer to the DB2 9 version of this topic available in the DB2 9 Information Center at:http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp

When data redistribution is performed, information about each table which is processed is logged in a single redistribute event log file.

The event log file name is formatted like: database-name.database-partition-group-name.timestamp.log. The log files are located as follows:

- The homeinst/sqllib/redist directory on Linux® and UNIX® based 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 an event log file name:

```
DB819.NG1.2007062419415651.log
```

This event log file is for a redistribute operation on a database named DB819 with a database partition group named NG1 that was created on June 24, 2007 at 7:41 PM local time.

The three main uses of the event log file are as follows:

- To provide general information about the redistribute operation, such as the old and new distribution maps.
- Provide users with information that will help them keep track of 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.

For more information about redistribute log file entries and how to recover from errors during data redistribution, see:

Redistributing database partition groups using the STEPWISE_REDISTRIBUTE_DBPG procedure

Redistributing data can be performed using the STEPWISE_REDISTRIBUTE_DBPG system-defined procedure.

Redistributing database partition groups can be done using the STEPWISE_REDISTRIBUTE_DBPG system-defined procedure and other system-defined procedures.

The following steps outline what must be done and an example that demonstrates these steps follows:

1. Analyze the database partition group regarding log space availability and data skew using *ANALYZE_LOG_SPACE procedure - Retrieve log space analysis information*.

   The analyze_log_space function 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 - Generate a data distribution file*.

   The generate_distfile function 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 *STEPWISE_REDISTRIBUTE_DBPG procedure - Redistribute part of a database partition group*.
4. Create a data distribution file for a given table using the *GET_SWRD_SETTINGS procedure - Retrieve redistribute information* and *SET_SWRD_SETTINGS procedure - Create or change redistribute registry*.

   The get_swrd_settings function reads the existing redistribute registry records for the given database partition group.

   The set_swrd_settings function 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 *STEPWISE_REDISTRIBUTE_DBPG procedure - Redistribute part of database partition group*.

   The stepwise_redistribute_dbpg function redistributes part of the database partition group according to the input and the setting file.

Usage 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="$HOME/sqllib/function/$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)"
```

Chapter 27. Configuring self-tuning memory

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 in 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. Once tuning decisions are made on the tuning partition, the memory adjustments are distributed to all other database partitions to ensure that all database partitions maintain similar configurations.

The single tuning partition model necessitates that the feature be used only on database partitions that have similar memory requirements. The following are guidelines to use when determining whether to enable self tuning memory on your partitioned database.

Cases where self tuning is recommended in partitioned databases

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 on identical hardware, including an even distribution of multiple logical nodes to multiple physical nodes
- Perfect or near-perfect distribution of data
- Workload running on the database partitions is distributed evenly across database partitions. This means that no one database partition has elevated memory requirements for one or more heaps.

In such an environment, it is desirable to have all database partitions configured equally, and self tuning memory will properly configure the system.

Cases where self tuning is recommended in partitioned databases with care

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 a set of database partitions for data, and a much smaller set of coordinator partitions and a catalog partitions. In such environments, it might be beneficial to configure the coordinator partitions and catalog partitions differently than the database partitions that contain your data.

In this environment, it is still possible to benefit from the self tuning memory feature with some minor setup. Since the database partitions containing the data comprise the bulk of the database partitions, self tuning should be enabled on all of these database partitions and one of these database partitions should be specified as the tuning partition. Additionally, since the catalog and coordinator partitions might be configured differently, self tuning memory should be disabled

© Copyright IBM Corp. 1993, 2009

on these partitions. To disable self tuning on the catalog and coordinator partitions, update the *self_tuning_mem* database configuration parameter on these partitions to OFF.

Cases where self tuning is not recommended in partitioned databases

In environments where the memory requirements of each database partition are different or when different database partitions are running on dramatically different hardware, it is advisable to disable the self tuning memory feature. This can be done by setting the *self_tuning_mem* database configuration parameter to OFF on all partitions.

Comparing memory requirements of different database partitions

The best way to determine if 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 partitions (differing by no more than 20%), then the partitions can be considered similar.

Collect the following data by issuing the command `get snapshot for database on <dbname>`.

```
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

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
```

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 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 based on a number of characteristics, such as the number of database partitions in the partition group and the number of buffer pools defined.

- To determine which database partition is currently specified as the tuning partition, use the following ADMIN_CMD:

  ```
  CALL SYSPROC.ADMIN_CMD( 'get stmm tuning dbpartitionnum' )
  ```

- To change the tuning partition, use the following ADMIN_CMD:

  ```
  CALL SYSPROC.ADMIN_CMD( 'update stmm tuning dbpartitionnum <db_partition_num>')
  ```

  When you issue this command, the tuning partition will be updated asynchronously or at the next database startup.

- To have the memory tuner automatically re-select the tuning partition, enter -1 for the <db_partition_num> value.

Starting the memory tuner on DPF systems

The memory tuner will only be started in a DPF environment if the database is activated by an explicit ACTIVATE DATABASE command because self-tuning requires all partitions to be active before it can properly tune the memory on a multi-partition system.

Disabling self-tuning for a given database partition

- To disable self-tuning for a subset of database partitions, set the *self_tuning_mem* configuration parameter to OFF for the database partitions you want to leave untuned.
-

  To disable self-tuning for a subset of the memory consumers controlled by configuration parameters on a particular database partition, set the value of the relevant configuration parameter or buffer pool size to MANUAL or a specific value on that database partition. However, it is recommended that self-tuning configuration parameter values be consistent across all running partitions.

- To disable tuning for a particular buffer pool on a database partition, issue an ALTER BUFFER POOL command specifying a size value and a value for the PARTITIONNUM parameter for the partition where self-tuning is to be disabled.

  An ALTER BUFFERPOOL statement that specifies the size on a particular database partition using the PARTITIONNUM clause will create an exception entry for the given buffer pool in the SYSCAT.SYSBUFFERPOOLNODES catalog, or update the exception entry if one already exists. When an exception entry exists for a buffer pool in this catalog, that buffer pool will not participate in self-tuning when the default buffer pool size is set to AUTOMATIC. To remove an exception entry so that a buffer pool can be re-enabled for self-tuning:

  1. Disable 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 with the PARTITIONNUM clause specified to set the size of the buffer pool on this database partition to the default buffer pool size.
3. Enable self-tuning by issuing another ALTER BUFFERPOOL statement setting the size to AUTOMATIC.

Enabling self-tuning memory in non-uniform environments

Ideally, your data should be distributed evenly across all of your database partitions and the workload 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 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 configuration in the remaining partitions should be configured manually.

Chapter 28. DB2 configuration parameters and variables

Configuring databases across multiple partitions

The database manager provides a single view of all database configuration elements across multiple partitions. This means that you can update or reset a database configuration across all database partitions without invoking the db2_all command against each database partition.

You can update a database configuration across partitions by issuing only one SQL statement or only one administration command from any partition on which the database resides. By default, the method of updating or resetting a database configuration is *on all database partitions*.

For backward compatibility of command scripts and applications, you have three options:

- Use the db2set command to set the **DB2_UPDDBCFG_SINGLE_DBPARTITION** registry variable to TRUE, as follows:

  ```
  DB2_UPDDBCFG_SINGLE_DBPARTITION=TRUE
  ```

  Note: Setting the registry variable does not apply to UPDATE DATABASE CONFIGURATION or RESET DATABASE CONFIGURATION requests that you make using the ADMIN_CMD procedure.
- Use the **DBPARTITIONNUM** parameter with either the UPDATE DATABASE CONFIGURATION or the RESET DATABASE CONFIGURATION command or with the ADMIN_CMD procedure. For example, to update the database configurations on all database partitions, call the ADMIN_CMD procedure as follows:

  ```
  CALL SYSPROC.ADMIN_CMD
    ('UPDATE DB CFG USING sortheap 1000')
  ```

  To update a single database partition, call the ADMIN_CMD procedure as follows:

  ```
  CALL SYSPROC.ADMIN_CMD
    ('UPDATE DB CFG DBPARTITIONNUM 10 USING sortheap 1000')
  ```
- Use the **DBPARTITIONNUM** parameter with the db2CfgSet API. The flags in the **db2Cfg** structure indicate whether the value for the database configuration is to be applied to a single database partition. If you set a flag, you must also provide the **DBPARTITIONNUM** value, for example:

  ```
  #define db2CfgSingleDbpartition          256
  ```

  If you do not set the db2CfgSingleDbpartition value, the value for the database configuration applies to all database partitions unless you set the **DB2_UPDDBCFG_SINGLE_DBPARTITION** registry variable to TRUE or you set *versionNumber* to anything that is less than the version number for Version 9.5, for the db2CfgSet API that sets the database manager or database configuration parameters.

When migrating your databases to Version 9.5, existing database configuration parameters, as a general rule, retain their values after migration. However, new parameters are added using their default values and some existing parameters are set to their new Version 9.5 default values. Refer to the ″DB2 server behavior changes″ topic in the *Migration Guide* for details about the changes to existing

© Copyright IBM Corp. 1993, 2009

database configuration parameters. Any subsequent update or reset database configuration requests for the migrated databases will use the Version 9.5 method of updating or resetting configuration requests.

For existing update or reset command scripts, the same rules mentioned previously apply: you can use the pre-Version 9.5 method, you can modify your scripts to include the **DBPARTITIONNUM** option of the UPDATE DATABASE CONFIGURATION or RESET DATABASE CONFIGURATION command, or you can set the **DB2_UPDDBCFG_SINGLE_DBPARTITION** registry variable.

For existing applications that call the db2CfgSet API, you must use the Version 9.5 method. If you want the pre-Version 9.5 method, you can set the **DB2_UPDDBCFG_SINGLE_DBPARTITION** registry variable or modify your applications to call the API with the Version 9.5 version number, including the new db2CfgSingleDbpartition flag and the new **dbpartitionnum** field to update or reset database configurations for a specific database partition.

Note: If you find that database configuration values are inconsistent, you can update or reset each database partition individually.

Partitioned database environment variables

DB2CHGPWD_EEE

- Operating system: DB2 ESE on AIX, Linux, and Windows
- Default=NULL, Values: YES or NO
- This variable specifies whether you allow other users to change passwords on AIX or Windows ESE systems. You must ensure that the passwords for all database partitions or nodes are maintained centrally using either a Windows domain controller on Windows, or LDAP on AIX. If not maintained centrally, passwords may not be consistent across all database partitions or nodes. This could result in a password being changed only at the database partition to which the user connects to make the change.

DB2_FCM_SETTINGS

- Operating system: Linux
- Default=NULL, Values: FCM_MAXIMIZE_SET_SIZE:[YES | TRUE | NO | FALSE]. The default value for FCM_MAXIMIZE_SET_SIZE is NO.
- Starting from Version 9.5 fix pack 4, you can set the **DB2_FCM_SETTINGS** registry variable with the FCM_MAXIMIZE_SET_SIZE token to preallocate a default 2 GB space for the fast communication manager (FCM) buffer. The token needs have a value of eitherYES or TRUE to enable this feature.

DB2_NUM_FAILOVER_NODES

- Operating system: All
- Default=2, Values: 0 to the required number of database partitions
- On each machine where a given database exists, set **DB2_NUM_FAILOVER_NODES** to specify the total number of database partitions in that database.

  In a DB2 database high availability solution, if a database server fails, the database partitions on the failed machine can be restarted on another machine. The fast communication manager (FCM) uses

DB2_NUM_FAILOVER_NODES to calculate how much memory to reserve on each machine to facilitate this failover.

For example, consider the following configuration:

– Machine A has two database partitions: 1 and 2.

– Machine B has two database partitions: 3 and 4.

– **DB2_NUM_FAILOVER_NODES** is set to 4 on both A and B.

At DB2START, FCM will reserve enough memory on both A and B to manage up to four database partitions so that if one machine fails, the two database partitions on the failed machine can be restarted on the other machine. If machine A fails, database partitions 1 and 2 can be restarted on machine B. If machine B fails, database partitions 3 and 4 can be restarted on machine A.

DB2_PARTITIONEDLOAD_DEFAULT

- Operating system: All supported ESE platforms
- Default=YES, Values: YES or NO
- The **DB2_PARTITIONEDLOAD_DEFAULT** registry variable lets users change the default behavior of the load utility in an ESE environment when no ESE-specific load options are specified. The default value is YES, which specifies that in an ESE environment if you do not specify ESE-specific load options, loading is attempted on all database partitions on which the target table is defined. When the value is NO, loading is attempted only on the database partition to which the load utility is currently connected.

  Note: This variable is deprecated and may be removed in a later release. The LOAD command has various options that can be used to achieve the same behavior. You can achieve the same results as the NO setting for this variable by specifying the following with the LOAD command: `PARTITIONED DB CONFIG MODE LOAD_ONLY OUTPUT_DBPARTNUMS x`, where x is the partition number of the partition into which you want to load data.

DB2PORTRANGE

- Operating system: Windows
- Values: nnnn:nnnn
- This value is set to the TCP/IP port range used by FCM so that any additional database partitions created on another machine will also have the same port range.

Partitioned database environment configuration parameters

Communications

conn_elapse - Connection elapse time

This parameter specifies the number of seconds within which a TCP/IP connection is to be established between two database partition servers.

Configuration type
Database manager

Applies to
Partitioned database server with local and remote clients

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]
10 [0–100]

Unit of measure
Seconds

If the attempt to connect succeeds within the time specified by this parameter, communications are established. If it fails, another attempt is made to establish communications. If the connection is attempted the number of times specified by the *max_connretries* parameter and always times out, an error is issued.

fcm_num_buffers - Number of FCM buffers

This parameter specifies the number of 4 KB buffers that are used for internal communications (messages) both among and within database servers.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]

32-bit platforms
Automatic [128 - 65 300]

64-bit platforms
Automatic [128 - 524 288]

- Database server with local and remote clients: the default is 1024
- Database server with local clients: the default is 512
- Partitioned database server with local and remote clients: the default is 4096

On single-partition database systems, this parameter is used only if intra-partition parallelism is enabled by changing the *intra_parallel* parameter from its default value of NO to YES.

It is possible to set both an initial value and the AUTOMATIC attribute.

When set to AUTOMATIC, FCM monitors resource usage and can either increase or decrease resources, if they are not used within 30 minutes. The amount by which resources can be increased or decreased depends on the platform, in particular, that on Linux it can only be increased 25% above the starting value. If the database manager cannot allocate the number of resources specified when an instance is started, it scales back the configuration values incrementally until it can start the instance.

If you have multiple logical nodes on the same machine, you might find it necessary to increase the value of this parameter. You might also find it necessary to increase the value of this parameter if you run out of message buffers because of the number of users on the system, the number of database partition servers on the system, or the complexity of the applications.

If you are using multiple logical nodes, one pool of *fcm_num_buffers* buffers is shared by all the multiple logical nodes on the same machine. The size of the pool will be determined by multiplying the *fcm_num_buffers* value times the number of logical nodes on that physical machine. Re-examine the value you are using; consider how many FCM buffers in total will be allocated on the machine (or machines) where the multiple logical nodes reside.

fcm_num_channels - Number of FCM channels

This parameter specifies the number of FCM channels for each database partition.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients
- Satellite database server with local clients

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]

UNIX 32-bit platforms
Automatic, with starting values of 256, 512, 2 048 [128 - 120 000]

UNIX 64-bit platforms
Automatic, with starting values of 256, 512, 2 048 [128 - 524 288]

Windows 32-bit
Automatic, with a starting value 10 000 [128 - 120 000]

Windows 64-bit
Automatic, with starting values of 256, 512, 2 048 [128 - 524 288]

- For database server with local and remote clients, the starting value is 512.
- For database server with local clients, the starting value is 256.
- For partitioned database environment servers with local and remote clients, the starting value is 2 048.

On non-partitioned database environments, the *intra_parallel* parameter must be active before *fcm_num_channels* can be used.

An FCM channel represents a logical communication end point between EDUs running in the DB2 engine. Both control flows (request and reply) and data flows (table queue data) rely on channels to transfer data between partitions.

When set to AUTOMATIC, FCM monitors channel usage, incrementally allocating and releasing resources as requirements change.

max_connretries - Node connection retries

This parameter specifies the maximum number of times an attempt will be made to establish a TCP/IP connection between two database partition servers.

Configuration type
Database manager

Applies to
Partitioned database server with local and remote clients

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]
5 [0–100]

If the attempt to establish communication between two database partition servers fails (for example, the value specified by the *conn_elapse* parameter is reached), *max_connretries* specifies the number of connection retries that can be made to a database partition server. If the value specified for this parameter is exceeded, an error is returned.

max_time_diff - Maximum time difference among nodes

This parameter specifies the maximum time difference, in minutes, that is permitted among the database partition servers listed in the node configuration file.

Configuration type
Database manager

Applies to
Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
60 [1 - 1 440]

Unit of measure
Minutes

Each database partition server has its own system clock. If two or more database partition servers are associated with a transaction, and their clocks are not synchronized to within the time specified by this parameter, the transaction is rejected and an SQLCODE is returned. (The transaction is rejected only if data modification is associated with it.)

DB2 uses *Coordinated Universal Time* (UTC), so different time zones are not a consideration when you set this parameter. The Coordinated Universal Time is the same as Greenwich Mean Time.

start_stop_time - Start and stop timeout

This parameter specifies the time, in minutes, within which all database partition servers must respond to a DB2START or a DB2STOP command. It is also used as the timeout value during an ADD DBPARTITIONNUM operation.

Configuration type
Database manager

Applies to
Database server with local and remote clients

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]
10 [1 - 1 440]

Unit of measure
Minutes

Database partition servers that do not respond to a DB2START command within the specified time send a message to the `db2start` error log in the `log` subdirectory of the `sqllib` subdirectory of the home directory for the instance. You should issue a DB2STOP on these nodes before restarting them.

Database partition servers that do not respond to a DB2STOP command within the specified time send a message to the `db2stop` error log in the `log` subdirectory of the `sqllib` subdirectory of the home directory for the instance. You can either issue `db2stop` for each database partition server that does not respond, or for all of them. (Those that are already stopped will return stating that they are stopped.)

If a `db2start` or `db2stop` operation in a multi-partition database is not completed within the value specified by the *start_stop_time* database manager configuration parameter, the database partitions that have timed out will be killed internally. Environments with many database partitions with a low value for *start_stop_time* might experience this behavior. To resolve this behavior, increase the value of *start_stop_time*.

When adding a new database partition using one of the DB2START, START DATABASE MANAGER, or ADD DBPARTITIONNUM commands, the add database partition operation must determine whether or not each database in the instance is enabled for automatic storage. This is done by communicating with the catalog partition for each database. If automatic storage is enabled, the storage path definitions are retrieved as part of that communication. Likewise, if system temporary table spaces are to be created with the database partitions, the operation might have to communicate with another database partition server to retrieve the table space definitions for the database partitions that reside on that server. These factors should be considered when determining the value of the *start_stop_time* parameter.

Parallel processing

intra_parallel - Enable intra-partition parallelism

This parameter specifies whether the database manager can use intra-partition parallelism.

Configuration type
Database manager

Applies to

- Database server with local and remote clients

- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
NO (0) [SYSTEM (-1), NO (0), YES (1)]

A value of -1 causes the parameter value to be set to "YES" or "NO" based on the hardware on which the database manager is running.

Some of the operations that can take advantage of parallel performance improvements when this parameter is ″YES″ include database queries and index creation.

Note:

- Parallel index creation does not use this configuration parameter.
- If you change this parameter value, packages might be rebound to the database, and some performance degradation might occur.

max_querydegree - Maximum query degree of parallelism

This parameter specifies the maximum degree of intra-partition parallelism that is used for any SQL statement executing on this instance of the database manager. An SQL statement will not use more than this number of parallel operations within a database partition when the statement is executed.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable Online

Propagation class
Statement boundary

Default [range]
-1 (ANY) [ANY, 1 - 32 767] (ANY means system-determined)

The *intra_parallel* configuration parameter must be set to "YES" to enable the database partition to use intra-partition parallelism for SQL statements. The *intra_parallel* parameter is no longer required for parallel index creation.

The default value for this configuration parameter is -1. This value means that the system uses the degree of parallelism determined by the optimizer; otherwise, the user-specified value is used.

Note: The degree of parallelism for an SQL statement can be specified at statement compilation time using the CURRENT DEGREE special register or the DEGREE bind option.

The maximum query degree of parallelism for an active application can be modified using the SET RUNTIME DEGREE command. The actual runtime degree used is the lower of:

- *max_querydegree* configuration parameter
- Application runtime degree
- SQL statement compilation degree

This configuration parameter applies to queries only.

Part 5. Administrative APIs, commands, SQL statements

© Copyright IBM Corp. 1993, 2009

Chapter 29. Administrative APIs

sqleaddn - Add a database partition server to the partitioned database environment

Adds a new database partition server to the partitioned database environment. This API creates database partitions for all databases currently defined in the instance on the new database partition server. The user can specify the source database partition server for any system temporary table spaces to be created with the databases, or specify that no system temporary table spaces are to be created. The API must be issued from the database partition server that is being added, and can only be issued on a database partition server.

Scope

This API only affects the database partition server on which it is executed.

Authorization

One of the following:

- sysadm
- sysctrl

Required connection

None

API include file

```
sqlenv.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  sqleaddn (
        void * pAddNodeOptions,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgaddn (
        unsigned short addnOptionsLen,
        struct sqlca * pSqlca,
        void * pAddNodeOptions);
```

sqleaddn API parameters

pAddNodeOptions
Input. A pointer to the optional sqle_addn_options structure. This structure is used to specify the source database partition server, if any, of the system temporary table space definitions for all database partitions created during the add node operation. If not specified (that is, a NULL pointer is specified), the system temporary table space definitions will be the same as those for the catalog partition.

pSqlca
Output. A pointer to the sqlca structure.

© Copyright IBM Corp. 1993, 2009

sqlgaddn API-specific parameters

addnOptionsLen
Input. A 2-byte unsigned integer representing the length of the optional sqle_addn_options structure in bytes.

Usage notes

Before adding a new database partition server, ensure that there is sufficient storage for the containers that must be created for all existing databases on the system.

The add node operation creates an empty database partition on the new database partition server for every database that exists in the instance. The configuration parameters for the new database partitions are set to the default value.

If an add node operation fails while creating a database partition locally, it enters a clean-up phase, in which it locally drops all databases that have been created. This means that the database partitions are removed only from the database partition server being added (that is, the local database partition server). Existing database partitions remain unaffected on all other database partition servers. If this fails, no further clean up is done, and an error is returned.

The database partitions on the new database partition server cannot be used to contain user data until after the ALTER DATABASE PARTITION GROUP statement has been used to add the database partition server to a database partition group.

This API will fail if a create database or a drop database operation is in progress. The API can be called again once the operation has completed.

This API will fail if at any time in a database in the system a user table with an XML column had been, successfully or not, created or an XSR object had been, successfully or not, registered.

To determine whether or not a database is enabled for automatic storage, the sqleaddn API has to communicate with the catalog partition for each of the databases in the instance. If automatic storage is enabled then the storage path definitions are retrieved as part of that communication. Likewise, if system temporary table spaces are to be created with the database partitions, the sqleaddn API may have to communicate with another database partition server in the partitioned database environment in order to retrieve the table space definitions. The start_stop_time database manager configuration parameter is used to specify the time, in minutes, by which the other database partition server must respond with the automatic storage and table space definitions. If this time is exceeded, the API fails. Increase the value of start_stop_time, and call the API again.

REXX API syntax

This API can be called from REXX through the SQLDB2 interface.

sqlecran - Create a database on a database partition server

Creates a database only on the database partition server that calls the API. This API is not intended for general use. For example, it should be used with db2Restore if the database partition at a database partition server was damaged and must be recreated. Improper use of this API can cause inconsistencies in the system, so it should only be used with caution.

Note: If this API is used to recreate a database partition that was dropped (because it was damaged), the database at this database partition server will be in the restore-pending state. After recreating the database partition, the database must immediately be restored on this database partition server.

Scope

This API only affects the database partition server on which it is called.

Authorization

One of the following:

- sysadm
- sysctrl

Required connection

Instance. To create a database at another database partition server, it is necessary to first attach to that database partition server. A database connection is temporarily established by this API during processing.

API include file

```
sqlenv.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  sqlecran (
        char * pDbName,
        void * pReserved,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgcran (
        unsigned short reservedLen,
        unsigned short dbNameLen,
        struct sqlca * pSqlca,
        void * pReserved,
        char * pDbName);
```

sqlecran API parameters

pDbName
Input. A string containing the name of the database to be created. Must not be NULL.

pReserved
Input. A spare pointer that is set to null or points to zero. Reserved for future use.

pSqlca
Output. A pointer to the sqlca structure.

sqlgcran API-specific parameters

reservedLen
Input. Reserved for the length of pReserved.

dbNameLen
Input. A 2-byte unsigned integer representing the length of the database name in bytes.

Usage notes

When the database is successfully created, it is placed in restore-pending state. The database must be restored on this database partition server before it can be used.

REXX API syntax

This API can be called from REXX through the SQLDB2 interface.

sqledpan - Drop a database on a database partition server

Drops a database at a specified database partition server. Can only be run in a partitioned database environment.

Scope

This API only affects the database partition server on which it is called.

Authorization

One of the following:

- sysadm
- sysctrl

Required connection

None. An instance attachment is established for the duration of the call.

API include file

```
sqlenv.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  sqledpan (
        char * pDbAlias,
        void * pReserved,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgdpan (
        unsigned short Reserved1,
        unsigned short DbAliasLen,
        struct sqlca * pSqlca,
        void * pReserved2,
        char * pDbAlias);
```

sqledpan API parameters

pDbAlias
Input. A string containing the alias of the database to be dropped. This name is used to reference the actual database name in the system database directory.

pReserved
Reserved. Should be NULL.

pSqlca
Output. A pointer to the sqlca structure.

sqlgdpan API-specific parameters

Reserved1
Reserved for future use.

DbAliasLen
Input. A 2-byte unsigned integer representing the length in bytes of the database alias.

pReserved2
A spare pointer that is set to null or points to zero. Reserved for future use.

Usage notes

Improper use of this API can cause inconsistencies in the system, so it should only be used with caution.

REXX API syntax

This API can be called from REXX through the SQLDB2 interface.

sqledrpn - Check whether a database partition server can be dropped

Verifies whether a database partition server is being used by a database. A message is returned, indicating whether the database partition server can be dropped.

Scope

This API only affects the database partition server on which it is issued.

Authorization

One of the following:

- sysadm
- sysctrl

API include file

`sqlenv.h`

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  sqledrpn (
        unsigned short Action,
        void * pReserved,
```

```
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgdrpn (
        unsigned short Reserved1,
        struct sqlca * pSqlca,
        void * pReserved2,
        unsigned short Action);
```

sqledrpn API parameters

Action
The action requested. The valid value is: SQL_DROPNODE_VERIFY

pReserved
Reserved. Should be NULL.

pSqlca
Output. A pointer to the sqlca structure.

sqlgdrpn API-specific parameters

Reserved1
Reserved for the length of pReserved2.

pReserved2
A spare pointer that is set to NULL or points to 0. Reserved for future use.

Usage notes

If a message is returned, indicating that the database partition server is not in use, use the db2stop command with DROP NODENUM to remove the entry for the database partition server from the db2nodes.cfg file, which removes the database partition server from the partitioned database environment.

If a message is returned, indicating that the database partition server is in use, the following actions should be taken:

1. The database partition server to be dropped will have a database partition on it for each database in the instance. If any of these database partitions contain data, redistribute the database partition groups that use these database partitions. Redistribute the database partition groups to move the data to database partitions that exist at database partition servers that are not being dropped.
2. After the database partition groups are redistributed, drop the database partition from every database partition group that uses it. To remove a database partition from a database partition group, you can use either the drop node option of the sqludrdt API or the ALTER DATABASE PARTITION GROUP statement.
3. Drop any event monitors that are defined on the database partition server.
4. Rerun sqledrpn to ensure that the database partition at the database partition server is no longer in use.

REXX API syntax

This API can be called from REXX through the SQLDB2 interface.

sqlugrpn - Get the database partition server number for a row

Returns the database partition number and the database partition server number based on the distribution key values. An application can use this information to determine on which database partition server a specific row of a table is stored.

The partitioning data structure, sqlupi, is the input for this API. The structure can be returned by the sqlugtpi API. Another input is the character representations of the corresponding distribution key values. The output is a database partition number generated by the distribution strategy and the corresponding database partition server number from the distribution map. If the distribution map information is not provided, only the database partition number is returned. This can be useful when analyzing data distribution.

The database manager does not need to be running when this API is called.

Scope

This API must be invoked from a database partition server in the db2nodes.cfg file. This API should not be invoked from a client, since it could result in erroneous database partitioning information being returned due to differences in codepage and endianess between the client and the server.

Authorization

None

API include file

```
sqlutil.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  sqlugrpn (
   unsigned short num_ptrs,
   unsigned char ** ptr_array,
   unsigned short * ptr_lens,
   unsigned short territory_ctrycode,
   unsigned short codepage,
   struct sqlupi * part_info,
   short * part_num,
   SQL_PDB_NODE_TYPE * node_num,
   unsigned short chklvl,
   struct sqlca * sqlca,
   short dataformat,
   void * pReserved1,
   void * pReserved2);

SQL_API_RC SQL_API_FN
  sqlggrpn (
   unsigned short num_ptrs,
   unsigned char ** ptr_array,
   unsigned short * ptr_lens,
   unsigned short territory_code,
   unsigned short codepage,
   struct sqlupi * part_info,
   short * part_num,
   SQL_PDB_NODE_TYPE * node_num,
   unsigned short chklvl,
```

```
struct sqlca * sqlca,
short dataformat,
void * pReserved1,
void * pReserved2);
```

sqlugrpn API parameters

num_ptrs
The number of pointers in ptr_array. The value must be the same as the one specified for the part_info parameter; that is, part_info->sqld.

ptr_array
An array of pointers that points to the character representations of the corresponding values of each part of the distribution key specified in part_info. If a null value is required, the corresponding pointer is set to null. For generated columns, this function does not generate values for the row. The user is responsible for providing a value that will lead to the correct partitioning of the row.

ptr_lens
An array of unsigned integers that contains the lengths of the character representations of the corresponding values of each part of the partitioning key specified in part_info.

territory_ctrycode
The country/region code of the target database. This value can also be obtained from the database configuration file using the GET DATABASE CONFIGURATION command.

codepage
The code page of the target database. This value can also be obtained from the database configuration file using the GET DATABASE CONFIGURATION command.

part_info
A pointer to the sqlupi structure.

part_num
A pointer to a 2-byte signed integer that is used to store the database partition number.

node_num
A pointer to an SQL_PDB_NODE_TYPE field used to store the node number. If the pointer is null, no node number is returned.

chklvl An unsigned integer that specifies the level of checking that is done on input parameters. If the value specified is zero, no checking is done. If any non-zero value is specified, all input parameters are checked.

sqlca Output. A pointer to the sqlca structure.

dataformat
Specifies the representation of distribution key values. Valid values are:

SQL_CHARSTRING_FORMAT
All distribution key values are represented by character strings. This is the default value.

SQL_IMPLIEDDECIMAL_FORMAT
The location of an implied decimal point is determined by the column definition. For example, if the column definition is DECIMAL(8,2), the value 12345 is processed as 123.45.

SQL_PACKEDDECIMAL_FORMAT
All decimal column distribution key values are in packed decimal format.

SQL_BINARYNUMERICS_FORMAT
All numeric distribution key values are in big-endian binary format.

pReserved1
Reserved for future use.

pReserved2
Reserved for future use.

Usage notes

Data types supported on the operating system are the same as those that can be defined as a distribution key.

Note: CHAR, VARCHAR, GRAPHIC, and VARGRAPHIC data types must be converted to the database code page before this API is called.

For numeric and datetime data types, the character representations must be at the code page of the respective system where the API is invoked.

If node_num is not null, the distribution map must be supplied; that is, pmaplen field in part_info parameter (part_info->pmaplen) is either 2 or 8192. Otherwise, SQLCODE -6038 is returned. The distribution key must be defined; that is, sqld field in part_info parameter (part_info->sqld) must be greater than zero. Otherwise, SQLCODE -2032 is returned.

If a null value is assigned to a non-nullable partitioning column, SQLCODE -6039 is returned.

All the leading blanks and trailing blanks of the input character string are stripped, except for the CHAR, VARCHAR, GRAPHIC, and VARGRAPHIC data types, where only trailing blanks are stripped.

Chapter 30. Commands

REDISTRIBUTE DATABASE PARTITION GROUP

Redistributes data across the database partitions in a database partition group. The target distribution of data can be uniform (default) or user specified to meet specific system requirements.

The REDISTRIBUTE DATABASE PARTITION GROUP command redistributes data across all partitions in a database partition group. This affects all objects present in the database partition group and cannot be restricted to one object alone.

This command can only be issued from the catalog database partition. Use the LIST DATABASE DIRECTORY command to determine which database partition is the catalog database partition for each database.

Scope

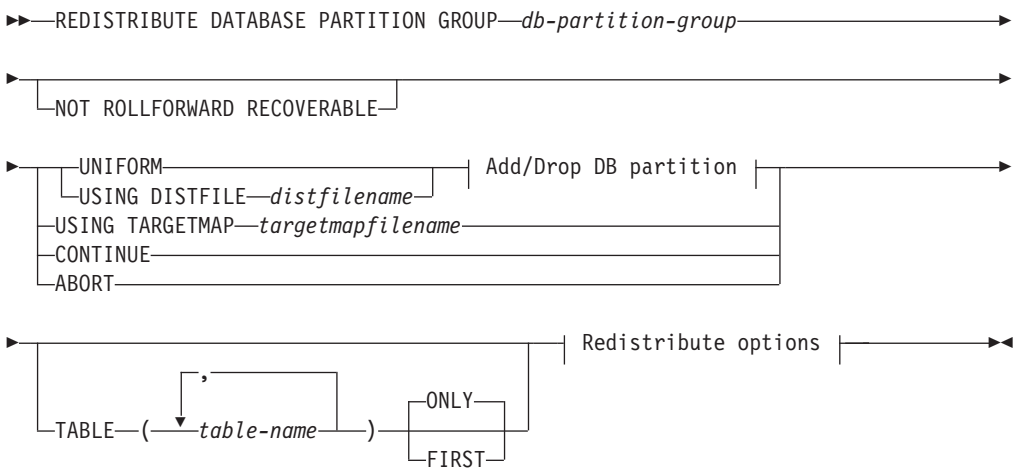
This command affects all database partitions in the database partition group.

Authorization

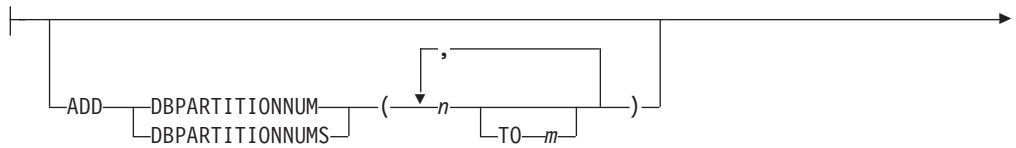
One of the following:

- SYSADM
- SYSCTRL
- DBADM

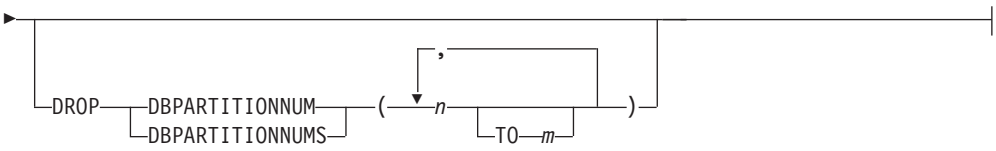
Command syntax



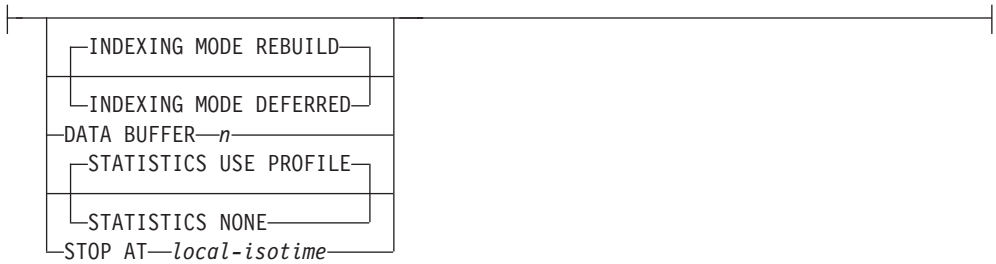
Add/Drop DB partition:



© Copyright IBM Corp. 1993, 2009



Redistribute options:



Command parameters

DATABASE PARTITION GROUP *db-partition-group*
The name of the database partition group. This one-part name identifies a database partition group described in the SYSCAT.DBPARTITIONGROUPS catalog table. The database partition group cannot currently be undergoing redistribution.

Note: Tables in the IBMCATGROUP and the IBMTEMPGROUP database partition groups cannot be redistributed.

NOT ROLLFORWARD RECOVERABLE
This option is only available when DB2 9.5 Fix Pack 1 is installed. When this option is used, the REDISTRIBUTE DATABASE PARTITION GROUP command 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 no longer required for each of the insert and delete operations. This means that you no longer need to manage large amounts of active log space and log archiving space in your system when performing data redistribution. This 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.

When this option is *not* used, extensive logging of all row movement is performed such that the database can be recovered later in the event of any interruptions, errors, or other business need.

UNIFORM
Specifies that the data is uniformly distributed across hash partitions (that is, every hash partition is assumed to have the same number of rows), but the same number of hash partitions do not map to each database partition. After redistribution, all database partitions in the database partition group have approximately the same number of hash partitions.

USING DISTFILE *distfilename*

If the distribution of distribution key values is skewed, use this option to achieve a uniform redistribution of data across the database partitions of a database partition group.

Use the *distfilename* to indicate the current distribution of data across the 4096 hash partitions.

Use row counts, byte volumes, or any other measure to indicate the amount of data represented by each hash partition. The utility reads the integer value associated with a partition as the weight of that partition. When a *distfilename* is specified, the utility generates a target distribution map that it uses to redistribute the data across the database partitions in the database partition group as uniformly as possible. After the redistribution, the weight of each database partition in the database partition group is approximately the same (the weight of a database partition is the sum of the weights of all hash partitions that map to that database partition).

For example, the input distribution file might contain entries as follows:

```
10223
1345
112000
0
100
...
```

In the example, hash partition 2 has a weight of 112000, and partition 3 (with a weight of 0) has no data mapping to it at all.

The *distfilename* should contain 4096 positive integer values in character format. The sum of the values should be less than or equal to 4294967295.

If the path for *distfilename* is not specified, the current directory is used.

USING TARGETMAP *targetmapfilename*

The file specified in *targetmapfilename* is used as the target distribution map. Data redistribution is done according to this file. If the path is not specified, the current directory is used.

If a database partition, included in the target map, is not in the database partition group, an error is returned. Issue ALTER DATABASE PARTITION GROUP ADD DBPARTITIONNUM statement before running REDISTRIBUTE DATABASE PARTITION GROUP command.

If a database partition, excluded from the target map, *is* in the database partition group, that database partition will not be included in the partitioning. Such a database partition can be dropped using ALTER DATABASE PARTITION GROUP DROP DBPARTITIONNUM statement either before or after the REDISTRIBUTE DATABASE PARTITION GROUP command.

CONTINUE

Continues a previously failed or stopped REDISTRIBUTE DATABASE PARTITION GROUP operation. If none occurred, an error is returned.

ABORT

Aborts a previously failed or stopped REDISTRIBUTE DATABASE PARTITION GROUP operation. If none occurred, an error is returned.

ADD

DBPARTITIONNUM *n*

TO *m*

n or *n* **TO** *m* specifies a list or lists of database partition numbers which are to be added into the database partition group. Any specified partition must not already be defined in the database partition group (SQLSTATE 42728). This is equivalent to executing the ALTER DATABASE PARTITION GROUP statement with ADD DBPARTITIONNUM clause specified.

DBPARTITIONNUMS *n*

TO *m*

n or *n* **TO** *m* specifies a list or lists of database partition numbers which are to be added into the database partition group. Any specified partition must not already be defined in the database partition group (SQLSTATE 42728). This is equivalent to executing the ALTER DATABASE PARTITION GROUP statement with ADD DBPARTITIONNUM clause specified.

Note: When a database partition is added using this option, 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. If this would result in a naming conflict among containers, which could happen if the new partitions are on the same physical machine as existing containers, this option should not be used. Instead, the ALTER DATABASE PARTITION GROUP statement should be used with the WITHOUT TABLESPACES option prior to issuing the REDISTRIBUTE DATABASE PARTITION GROUP command. Table space containers can then be created manually specifying appropriate names.

DROP

DBPARTITIONNUM *n*

TO *m*

n or *n* **TO** *m* specifies a list or lists of database partition numbers which are to be dropped from the database partition group. Any specified partition must already be defined in the database partition group (SQLSTATE 42729). This is equivalent to executing the ALTER DATABASE PARTITION GROUP statement with the DROP DBPARTITIONNUM clause specified.

DBPARTITIONNUMS *n*

TO *m*

n or *n* **TO** *m* specifies a list or lists of database partition numbers which are to be dropped from the database partition group. Any specified partition must already be defined in the database partition group (SQLSTATE 42729). This is equivalent to executing the ALTER DATABASE PARTITION GROUP statement with the DROP DBPARTITIONNUM clause specified.

TABLE *tablename*
Specifies a table order for redistribution processing.

ONLY If the table order is followed by the **ONLY** keyword (which is the

default), then, only the specified tables will be redistributed. The remaining tables can be later processed by subsequent REDISTRIBUTE CONTINUE commands. This is the default.

FIRST If the table order is followed by the **FIRST** keyword, then, the specified tables will be redistributed with the given order and the remaining tables in the database partition group will be redistributed with random order.

INDEXING MODE

This parameter specifies how indexes are maintained during redistribution when the NOT ROLLFORWARD RECOVERABLE option is specified. Valid values are:

REBUILD

Indexes will be rebuilt from scratch. Indexes do not have to be valid to use this option. As a result of using this option, index pages will be clustered together on disk.

DEFERRED

Redistribute will not attempt to maintain any indexes. Indexes will be marked as needing a refresh. The first access to such indexes may force a rebuild, or indexes may be rebuilt when the database is restarted.

Note: For non-MDC tables, if there are invalid indexes on the tables, the REDISTRIBUTE DATABASE PARTITION GROUP command automatically rebuilds them if you do not specify **INDEXING MODE DEFERRED**. For an MDC table, even if you specify **INDEXING MODE DEFERRED**, a composite index that is invalid is rebuilt before table redistribution begins because the utility needs the composite index to process an MDC table.

DATA BUFFER *n*

Specifies the number of 4 KB pages to use as buffered space for transferring data within the utility. If the value specified is lower than the minimum supported value, the minimum value is used and no warning is returned. This memory is allocated directly from the utility heap, whose size can be modified through the **util_heap_sz** database configuration parameter. If a value is not specified, an intelligent default is calculated by the utility at runtime at the beginning of processing each table. Specifically, the default is to use 50% of the memory available in the utility heap at the time redistribution of the table begins and to take into account various table properties as well.

STOP AT *local-isotime*

When this option is specified, before beginning data redistribution for each table, the *local-isotime* is compared with the current local timestamp. If the specified *local-isotime* is equal to or earlier than the current local timestamp, the utility stops with a warning message. Data redistribution processing of tables in progress at the stop time will complete without interruption. No new data redistribution processing of tables begins. The unprocessed tables can be redistributed using the **CONTINUE** option. This *local-isotime* value is specified as a time stamp, a 7-part character string that identifies a combined date and time. The format is *yyyy-mm-dd-hh.mm.ss.nnnnnn* (year, month, day, hour, minutes, seconds, microseconds) expressed in local time.

STATISTICS

This option specifies that the utility should collect statistics for the tables

that have a statistics profile. Specifying this option is more efficient than separately issuing the RUNSTATS command after the data redistribution is completed.

USE PROFILE
Statistics will be collected for the tables with a statistics profile. For tables without a statistics profile, nothing will be done. This is the default.

NONE
Statistics will not be collected for tables.

Examples: Redistribute steps

You may want to add or drop node from node group. Following is the step for adding new node to a node group and redistribute the data. Added database partition is not in the distribution map, but the containers for the table spaces in the database partition group have been created; the database partition is added to the distribution map when a redistribute database partition group operation has completed successfully.

1. Identify the node groups that will require redistribution. In this document we name the node group needed to be redistributed as **sampleNodegrp**
2. Identify objects that should be disabled or removed before redistribute .
   a. Replicate MQTs: This type of MQT is not supported as part of the REDISTRIBUTE utility. They need to be dropped before running redistribute and recreated afterward.

      ```
      SELECT tabschema, tabname FROM syscat.tables WHERE partition_mode = 'R'
      ```

   b. Write-to-table event monitors: Any auto start write to table event monitor , which it's table reside in the nodegroup to be redistributed should be disabled.

      ```
      SELECT distinct evmonname FROM syscat.eventtables E
      JOIN syscat.tables T on T.tabname = E.tabname AND T.tabschema = E.tabschema
      JOIN syscat.tablespaces S on S.tbspace = T.tbspace AND S.ngname = 'sampleNodegrp'
      ```

   c. Explain tables: It is recommended to create the explain tables in a single partition nodegroup. However, if they are defined in a nodegroup that requires redistribution, you may consider dropping them before the redistribute and redefining them once redistribute is complete, if the data generated to date does not need to be maintained.
   d. Table access mode and load state: Ensure that all tables in the node groups to be redistributed are in full access mode and have no load pending or load in progress state.

      ```
      SELECT distinct trim(T.creator) || \'.\' || trim(T.name) as name, T.access_mode, A.load_status
       FROM sysibm.systables T, sysibm.sysnodegroups N, sysibmadm.admintabinfo A
       WHERE T.pmap_id = N.pmap_id
        AND A.tabschema = T.creator
        AND A.tabname = T.name
        AND N.name = 'sampleNodegrp'
        AND (T.access_mode <> 'F' or A.load_status is not null)
      ```

   e. Statistics profiles: Table statistics can be updated as part of the redistribution process if a statistics profile is defined for the table. Having the REDISTRIBUTE utility update a table's statistics reduces I/O as all the data is scanned for the redistribute and no additional scan of the data is needed for RUNSTATS.

      ```
      RUNSTATS on table schema.table USE PROFILE runstats_profile SET PROFILE ONLY
      ```

3. Review the database configuration. **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 redistribute performance.

4. 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's important to update the database configuration on the new nodes before the REDISTRIBUTE command is issued to ensure that the configuration is balanced across the entire warehouse.

```
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')
```

5. Backup the database (or the table spaces in node groups that will be redistributed), before starting the redistribution process to ensure a recent recovery point.
6. Define the new data BCUs in DB2 by updating the db2nodes.cfg file and adding the new data BCU database partition specifications and define the new database partitions to DB2 using the ADD NODE WITHOUT TABLESPACES command.

   ```
   db2start nodenum x export DB2NODE=x
   db2 add node without tablespaces
   db2stop nodenum x
   ```

   Note: If it's not the first logical port on the data BCU, then execute a start and stop of the first logical port number before and after the above sequence of commands for subsequent logical ports.
7. Define system temporary table space containers on the newly defined database partitions.

   ```
   ALTER TABLESPACE tablespace_name ADD container_information ON dbpartitionnums (x to y)
   ```
8. Add the new logical database partitions to the database partition groups that span the data BCUs.

   ```
   ALTER DATABESE PARTITION GROUP partition_group_name ADD dbpartitionnums (x to y) WITHOUT TABLESPACES
   ```
9. Define permanent data table space containers on the newly defined database partitions.

   ```
   ALTER TABLESPACE tablespace_name ADD container_information ON dbpartitionnums (x to y)
   ```
10. Apply the database configuration settings retrieved in step 4 to the new database partitions (or issue a single UPDATE DB CFG command against all database partitions using the new DB2 9.5 single view of configuration support).
11. Capture the definition of and then drop any replicated MQTs existing in the database partition groups to be redistributed.

    ```
    db2look -d dbname -e -z schema -t replicated_MQT_table_names -o repMQTs.clp
    ```
12. Disable any write-to-table event monitors that exist in the database partition groups to be redistributed.

    ```
    SET EVENT MONITOR monitor_name STATE 0
    ```

13. Run the REDISTRIBUTE utility to redistribute uniformly across all database partitions. Following shows the simple redistribute command:

```
REDISTRIBUTE DATABASE PARTITION GROUP sampleNodegrp NOT ROLLFORWARD RECOVERABLE uniform;
```

User also should consider specifying a table list as input to the REDISTRIBUTE command to enforce the order that the tables will be processed. The REDISTRIBUTE 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 sampleNodegrp
  NOT ROLLFORWARD RECOVERABLE uniform TABLE (tab1, tab2,...) FIRST;
```

Consequences of using the NOT ROLLFORWARD RECOVERABLE option

When the REDISTRIBUTE DATABASE PARTITION GROUP command is issued and the NOT ROLLFORWARD RECOVERABLE option 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. It is important, however, for users to be aware that as a result of this minimal logging model, the REDISTRIBUTE DATABASE PARTITION GROUP command is *not* rollforward recoverable. This means that any operation that results in the database rolling forward through a redistribute operation results in all tables touched by the redistribution operation being left in the UNAVAILABLE state. 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, forcing the user to backup all redistributed table spaces at the end of a successful redistribute operation. With a backup taken after the redistribution operation, the user should not have a need to rollforward through the redistribute operation itself.

There is one very important consequence of the redistribute utility's lack of rollforward recoverability of which the user should be aware: If the user chooses 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 by the user, 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 rollforward recoverable. If it is necessary to restore the database from a backup taken prior to the redistribution operation, then it will not be possible to rollforward through the logs in order to replay the updates that were made during the redistribution operation without also rolling forward through the redistribute 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 prior to redistribute 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:

- The user avoids 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.
- 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, the user must decide whether such updates are appropriate or not for their scenario based on whether or not 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 prior to the redistribute operation.

Usage notes

- When the **NOT ROLLFORWARD RECOVERABLE** option is specified and the database is a recoverable database, the first time the utility accesses a table space, it is put into the BACKUP PENDING state. All the tables in that table space will become read-only until the table space is backed-up, which can only be done when all tables in the table space have finished being redistributed.
- When a redistribution operation is running, it produces an event log file containing general information about the redistribution operation and information such as the starting and ending time of each table processed. This event log file is written to:
  - The homeinst/sqllib/redist directory on Linux and UNIX operating systems, using the following format for subdirectories and file name: *database-name.database-partition-group-name.timestamp*.log.
  - The **DB2INSTPROF***instance*\redist directory on Windows operating systems (where **DB2INSTPROF** is the value of the **DB2INSTPROF** registry variable), using the following format for subdirectories and file name: *database-name.database-partition-group-name.timestamp*.log.
  - The time stamp value is the time when the command was issued.

  For more information about the redistribute event log, refer to the "Redistribution event log file" topic.
- This utility performs intermittent COMMITs during processing.
- All packages having a dependency on a table that has undergone redistribution are invalidated. It is recommended to explicitly rebind such packages after the redistribute database partition group operation has completed. Explicit rebinding eliminates the initial delay in the execution of the first SQL request for the invalid package. The redistribute message file contains a list of all the tables that have undergone redistribution.

- By default, the redistribute utility will update the statistics for those tables that have a statistics profile. For the tables without a statistics profile, it is recommended that you separately update the table and index statistics for these tables by calling the db2Runstats API or by issuing the RUNSTATS command after the redistribute operation has completed.
- Database partition groups containing replicated materialized query tables or tables defined with DATA CAPTURE CHANGES cannot be redistributed.
- Redistribution is not allowed if there are user temporary table spaces with existing declared temporary tables in the database partition group.
- Options such as **INDEXING MODE** are ignored on tables, on which they do not apply, without warning. For example, **INDEXING MODE** will be ignored on tables without indexes.
- Before starting a redistribute operation, ensure there are no tables in the Load Pending state. Table states can be checked by using the LOAD QUERY command.

Compatibilities

For compatibility with versions earlier than Version 8:

- The keyword **NODEGROUP** can be substituted for **DATABASE PARTITION GROUP**.

db2nchg - Change database partition server configuration

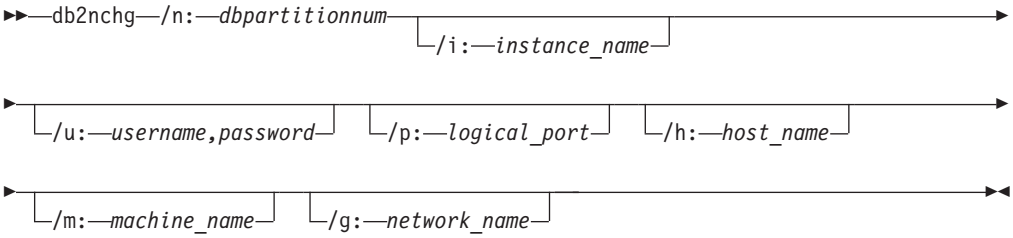
Modifies database partition server configuration. This includes moving the database partition server (node) from one machine to another; changing the TCP/IP host name of the machine; and selecting a different logical port number or a different network name for the database partition server (node). This command can only be used if the database partition server is stopped.

This command is available on Windows operating systems only.

Authorization

```
Local Administrator
```

Command syntax



Command parameters

/n:*dbpartitionnum*
Specifies the database partition number of the database partition server's configuration that is to be changed.

/i:*instance_name*
Specifies the instance in which this database partition server participates. If a parameter is not specified, the default is the current instance.

/u:*username,password*
Specifies the user name and password. If a parameter is not specified, the existing user name and password will apply.

/p:*logical_port*
Specifies the logical port for the database partition server. This parameter must be specified to move the database partition server to a different machine. If a parameter is not specified, the logical port number will remain unchanged.

/h:*host_name*
Specifies TCP/IP host name used by FCM for internal communications. If this parameter is not specified, the host name will remain the same.

/m:*machine_name*
Specifies the machine where the database partition server will reside. The database partition server can only be moved if there are no existing databases in the instance.

/g:*network_name*
Changes the network name for the database partition server. This parameter can be used to apply a specific IP address to the database partition server when there are multiple IP addresses on a machine. The network name or the IP address can be entered.

Examples

To change the logical port assigned to database partition 2, which participates in the instance TESTMPP, to logical port 3, enter the following command:

```
db2nchg /n:2 /i:TESTMPP /p:3
```

db2ncrt - Add database partition server to an instance

Adds a database partition server (node) to an instance.

This command is available on Windows operating systems only.

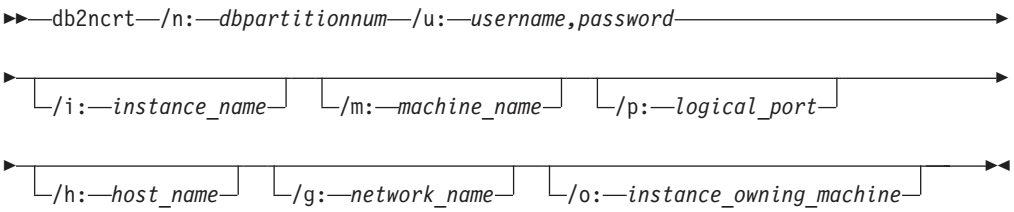
Scope

If a database partition server is added to a computer where an instance already exists, a database partition server is added as a logical database partition server to the computer. If a database partition server is added to a computer where an instance does not exist, the instance is added and the computer becomes a new physical database partition server. This command should not be used if there are databases in an instance. Instead, the START DATABASE MANAGER command should be issued with the ADD DBPARTITIONNUM option. This ensures that the database is correctly added to the new database partition server. It is also possible to add a database partition server to an instance in which a database has been created. The `db2nodes.cfg` file should not be edited since changing the file might cause inconsistencies in the partitioned database environment.

Authorization

`Local Administrator` authority on the computer where the new database partition server is added.

Command syntax



Command parameters

/n:*dbpartitionnum*
A unique database partition number which identifies the database partition server. The number entered can range from 1 to 999.

/u:*username,password*
Specifies the logon account name and password for DB2.

/i:*instance_name*
Specifies the instance name. If a parameter is not specified, the default is the current instance.

/m:*machine_name*
Specifies the computer name of the Windows workstation on which the database partition server resides. This parameter is required if a database partition server is added on a remote computer.

/p:*logical_port*
Specifies the logical port number used for the database partition server. If this parameter is not specified, the logical port number assigned will be 0. When creating a logical database partition server, this parameter must be specified and a logical port number that is not in use must be selected. Note the following restrictions:

- Every computer must have a database partition server that has a logical port 0.
- The port number cannot exceed the port range reserved for FCM communications in the `x:\winnt\system32\drivers\etc\` directory. For example, if a range of 4 ports is reserved for the current instance, then the maximum port number is 3. Port 0 is used for the default logical database partition server.

/h:*host_name*
Specifies the TCP/IP host name that is used by FCM for internal communications. This parameter is required when the database partition server is being added on a remote computer.

/g:*network_name*
Specifies the network name for the database partition server. If a parameter is not specified, the first IP address detected on the system will be used. This parameter can be used to apply a specific IP address to the database partition server when there are multiple IP addresses on a computer. The network name or the IP address can be entered.

/o:*instance_owning_machine*
Specifies the computer name of 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.

Examples

To add a new database partition server to the instance TESTMPP on the instance-owning computer SHAYER, where the new database partition server is known as database partition 2 and uses logical port 1, enter the following command:

```
db2ncrt /n:2 /u:QBPAULZ\paulz,g1reeky /i:TESTMPP /m:TEST /p:1 /o:SHAYER /h:TEST
```

db2ndrop - Drop database partition server from an instance

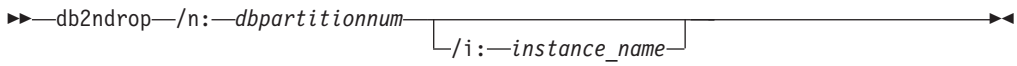
Drops a database partition server (node) from an instance that has no databases. If a database partition server is dropped, its database partition number can be reused for a new database partition server. This command can only be used if the database partition server is stopped.

This command is available on Windows operating systems only.

Authorization

`Local Administrator` authority on the machine where the database partition server is being dropped.

Command syntax



Command parameters

/n:*dbpartitionnum*
A unique database partition number which identifies the database partition server.

/i:*instance_name*
Specifies the instance name. If a parameter is not specified, the default is the current instance.

Examples

```
db2ndrop /n:2 /i=KMASCI
```

Usage notes

If the instance-owning database partition server (dbpartitionnum 0) is dropped from the instance, the instance becomes unusable. To drop the instance, use the db2idrop command.

This command should not be used if there are databases in this instance. Instead, the `db2stop drop nodenum` command should be used. This ensures that the database partition server is correctly removed from the partition database

environment. It is also possible to drop a database partition server in an instance where a database exists. The `db2nodes.cfg` file should not be edited since changing the file might cause inconsistencies in the partitioned database environment.

To drop a database partition server that is assigned to the logical port 0 from a machine that is running multiple logical database partition servers, all other database partition servers assigned to the other logical ports must be dropped first. Each database partition server must have a database partition server assigned to logical port 0.

Chapter 31. SQL language elements

Data types

Database partition-compatible data types

Database partition compatibility is defined between the base data types of corresponding columns of distribution keys. Database partition-compatible data types have the property that two variables, one of each type, with the same value, are mapped to the same distribution map index by the same database partitioning function.

Table 41 shows the compatibility of data types in database partitions.

Database partition compatibility has the following characteristics:

- Internal formats are used for DATE, TIME, and TIMESTAMP. 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. Different results might be produced for NULL values of non-compatible data types.
- Base data type of the UDT is used to analyze database 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 system-provided hashing function.
- When a locale-sensitive UCA-based collation is used, CHAR, VARCHAR, GRAPHIC, and VARGRAPHIC are compatible data types. When other collations are used, CHAR and VARCHAR are compatible types and GRAPHIC and VARGRAPHIC are compatible types, but CHAR and VARCHAR are not compatible types with GRAPHIC and VARGRAPHIC. CHAR or VARCHAR of different lengths are compatible data types.
- DECFLOAT values that are equal are treated identically even if their precision differs. DECFLOAT values that are numerically equal are treated identically even if they have a different number of significant digits.

Table 41. Database Partition Compatibilities

Operands	Binary Integer	Decimal Number	Floating-point	Decimal Floating-point	Character String	Graphic String	Date	Time	Time-stamp	Distinct Type	Structured Type
Binary Integer	Yes	No	No	No	No	No	No	No	No	[1]	No

© Copyright IBM Corp. 1993, 2009

Table 41. Database Partition Compatibilities (continued)

Operands	Binary Integer	Decimal Number	Floating-point	Decimal Floating-point	Character String	Graphic String	Date	Time	Time-stamp	Distinct Type	Structured Type
Decimal Number	No	Yes	No	No	No	No	No	No	No	[1]	No
Floating-point	No	No	Yes	No	No	No	No	No	No	[1]	No
Decimal Floating-point	No	No	No	Yes	No	No	No	No	No	[1]	No
Character String[4]	No	No	No	No	Yes[2]	2, 3	No	No	No	[1]	No
Graphic String[4]	No	No	No	No	2, 3	Yes[2]	No	No	No	[1]	No
Date	No	No	No	No	No	No	Yes	No	No	[1]	No
Time	No	No	No	No	No	No	No	Yes	No	[1]	No
Timestamp	No	No	No	No	No	No	No	No	Yes	[1]	No
Distinct Type	[1]	[1]	[1]	[1]	[1]	[1]	[1]	[1]	[1]	[1]	No
Structured Type[4]	No	No	No	No	No	No	No	No	No	No	No

Note:

[1] A user-defined distinct type (UDT) value is database partition compatible with the source type of the UDT or any other UDT with a database partition compatible source type.

[2] Character and graphic string types are compatible when they have compatible collations.

[3] Character and graphic string types are compatible when a locale-sensitive UCA-based collation is in effect. Otherwise, they are not compatible types.

[4] User-defined structured types and data types LONG VARCHAR, LONG VARGRAPHIC, CLOB, DBCLOB, and BLOB are not applicable for database partition compatibility, because they are not supported in distribution keys.

Special registers

CURRENT DBPARTITIONNUM

The CURRENT DBPARTITIONNUM special register specifies an INTEGER value that identifies the coordinator node number for the statement. For statements issued from an application, the coordinator is the database partition to which the application connects. For statements issued from a routine, the coordinator is the database partition from which the routine is invoked.

When used in an SQL statement inside a routine, CURRENT DBPARTITIONNUM is never inherited from the invoking statement.

CURRENT DBPARTITIONNUM returns 0 if the database instance is not defined to support database partitioning. (In other words, if there is no db2nodes.cfg file. For partitioned databases, the db2nodes.cfg file exists and contains database partition definitions.)

CURRENT DBPARTITIONNUM can be changed through the CONNECT statement, but only under certain conditions.

For compatibility with versions earlier than Version 8, the keyword NODE can be substituted for DBPARTITIONNUM.

Example: Set the host variable APPL_NODE (integer) to the number of the database partition to which the application is connected.

```
VALUES CURRENT DBPARTITIONNUM
  INTO :APPL_NODE
```

Chapter 32. SQL functions

DATAPARTITIONNUM

The schema is SYSIBM.

The DATAPARTITIONNUM function returns the sequence number (SYSDATAPARTITIONS.SEQNO) of the data partition in which the row resides. Data partitions are sorted by range, and sequence numbers start at 0. For example, the DATAPARTITIONNUM function returns 0 for a row that resides in the data partition with the lowest range.

The argument must be the qualified or unqualified name of any column in the table. Because row-level information is returned, the result is the same regardless of which column is specified. The column can have any data type.

If *column-name* references a column in a view, the expression for the column in the view must reference a column of the underlying base table, and the view must be deletable. A nested or common table expression follows the same rules as a view.

The data type of the result is INTEGER and is never null.

This function cannot be used as a source function when creating a user-defined function. Because the function accepts any data type as an argument, it is not necessary to create additional signatures to support user-defined distinct types.

The DATAPARTITIONNUM function cannot be used within check constraints or in the definition of generated columns (SQLSTATE 42881). The DATAPARTITIONNUM function cannot be used in a materialized query table (MQT) definition (SQLSTATE 428EC).

Example:

- ```
  SELECT DATAPARTITIONNUM (EMPNO)
  FROM EMPLOYEE
  ```

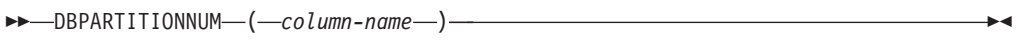
  To convert a sequence number that is returned by DATAPARTITIONNUM (for example, 0) to a data partition name that can be used in other SQL statements (such as, for example, ALTER TABLE...DETACH PARTITION), you can query the SYSCAT.DATAPARTITIONS catalog view. Include the SEQNO obtained from DATAPARTITIONNUM in the WHERE clause, as shown in the following example.

  ```
  SELECT DATAPARTITIONNAME
  FROM SYSCAT.DATAPARTITIONS
  WHERE TABNAME = 'EMPLOYEE' AND SEQNO = 0
  ```

  results in the value 'PART0'.

© Copyright IBM Corp. 1993, 2009
  ```

DBPARTITIONNUM



The schema is SYSIBM.

The DBPARTITIONNUM function returns the database partition number for a row. For example, if used in a SELECT clause, it returns the database partition number for each row in the result set.

The argument must be the qualified or unqualified name of any column in the table. Because row-level information is returned, the result is the same regardless of which column is specified. The column can have any data type.

If *column-name* references a column in a view, the expression for the column in the view must reference a column of the underlying base table, and the view must be deletable. A nested or common table expression follows the same rules as a view.

The specific row (and table) for which the database partition number is returned by the DBPARTITIONNUM function is determined from the context of the SQL statement that uses the function.

The database partition number returned on transition variables and tables is derived from the current transition values of the distribution key columns. For example, in a before insert trigger, the function returns the projected database partition number, given the current values of the new transition variables. However, the values of the distribution key columns might be modified by a subsequent before insert trigger. Thus, the final database partition number of the row when it is inserted into the database might differ from the projected value.

The data type of the result is INTEGER and is never null. If there is no `db2nodes.cfg` file, the result is 0.

This function cannot be used as a source function when creating a user-defined function. Because the function accepts any data type as an argument, it is not necessary to create additional signatures to support user-defined distinct types.

The DBPARTITIONNUM function cannot be used on replicated tables, within check constraints, or in the definition of generated columns (SQLSTATE 42881).

For compatibility with previous versions of DB2, NODENUMBER can be specified in place of DBPARTITIONNUM.

Examples:

- Count the number of instances in which the row for a given employee in the EMPLOYEE table is on a different database partition than the description of the employee's department in the DEPARTMENT table.

```
SELECT COUNT(*) FROM DEPARTMENT D, EMPLOYEE E
  WHERE D.DEPTNO=E.WORKDEPT
  AND DBPARTITIONNUM(E.LASTNAME) <> DBPARTITIONNUM(D.DEPTNO)
```

- Join the EMPLOYEE and DEPARTMENT tables so that the rows of the two tables are on the same database partition.

```
SELECT * FROM DEPARTMENT D, EMPLOYEE E
  WHERE DBPARTITIONNUM(E.LASTNAME) = DBPARTITIONNUM(D.DEPTNO)
```

- Using a before trigger on the EMPLOYEE table, log the employee number and the projected database partition number of any new row in the EMPLOYEE table in a table named EMPINSERTLOG1.

```
CREATE TRIGGER EMPINSLOGTRIG1
BEFORE INSERT ON EMPLOYEE
REFERENCING NEW AW NEWTABLE
FOR EACH ROW
INSERT INTO EMPINSERTLOG1
VALUES(NEWTABLE.EMPNO, DBPARTITIONNUM
(NEWTABLE.EMPNO))
```

Chapter 33. SQL statements

ALTER DATABASE PARTITION GROUP

The ALTER DATABASE PARTITION GROUP statement is used to:

- add one or more database partitions to a database partition group
- drop one or more database partitions from a database partition group.

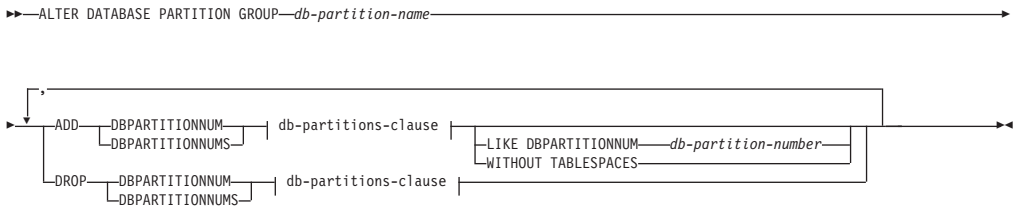
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

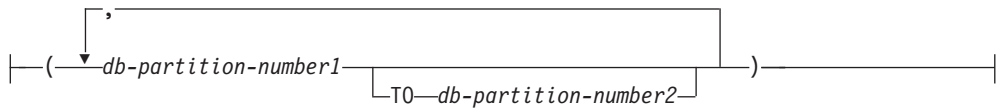
Authorization

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

Syntax



db-partitions-clause:



Description

db-partition-name
Names the database partition group. This is a one-part name. It is an SQL identifier (either ordinary or delimited). It must be a database partition group described in the catalog. IBMCATGROUP and IBMTEMPGROUP cannot be specified (SQLSTATE 42832).

ADD DBPARTITIONNUM
Specifies the specific database partition or partitions to add to the database partition group. DBPARTITIONNUMS is a synonym for DBPARTITIONNUM. Any specified database partition must not already be defined in the database partition group (SQLSTATE 42728).

DROP DBPARTITIONNUM
Specifies the specific database partition or partitions to drop from the database

© Copyright IBM Corp. 1993, 2009

partition group. DBPARTITIONNUMS is a synonym for DBPARTITIONNUM. Any specified database partition must already be defined in the database partition group (SQLSTATE 42729).

db-partitions-clause
Specifies the database partition or partitions to be added or dropped.

db-partition-number1
Specify a specific database partition number.

TO *db-partition-number2*
Specify a range of database partition numbers. The value of *db-partition-number2* must be greater than or equal to the value of *db-partition-number1* (SQLSTATE 428A9).

LIKE DBPARTITIONNUM *db-partition-number*
Specifies that the containers for the existing table spaces in the database partition group will be the same as the containers on the specified *db-partition-number*. The specified database partition must be a partition that existed in the database partition group prior to this statement, and that is not included in a DROP DBPARTITIONNUM clause of the same statement.

For table spaces that are defined to use automatic storage (that is, table spaces that were created with the MANAGED BY AUTOMATIC STORAGE clause of the CREATE TABLESPACE statement, or for which no MANAGED BY clause was specified at all), the containers will not necessarily match those from the specified partition. Instead, containers will automatically be assigned by the database manager based on the storage paths that are associated with the database, and this might or might not result in the same containers being used. The size of each table space is based on the initial size that was specified when the table space was created, and might not match the current size of the table space on the specified partition.

WITHOUT TABLESPACES
Specifies that the containers for existing table spaces in the database partition group are not created on the newly added database partition or partitions. The ALTER TABLESPACE statement using the *db-partitions-clause* must be used to define containers for use with the table spaces that are defined on this database partition group. If this option is not specified, the default containers are specified on newly added database partitions for each table space defined on the database partition group.

This option is ignored for table spaces that are defined to use automatic storage (that is, table spaces that were created with the MANAGED BY AUTOMATIC STORAGE clause of the CREATE TABLESPACE statement, or for which no MANAGED BY clause was specified at all). There is no way to defer container creation for these table spaces. Containers will automatically be assigned by the database manager based on the storage paths that are associated with the database. The size of each table space will be based on the initial size that was specified when the table space was created.

Rules

- Each database partition specified by number must be defined in the `db2nodes.cfg` file (SQLSTATE 42729).
- Each *db-partition-number* listed in the *db-partitions-clause* must be for a unique database partition (SQLSTATE 42728).
- A valid database partition number is between 0 and 999 inclusive (SQLSTATE 42729).

- A database partition cannot appear in both the ADD and DROP clauses (SQLSTATE 42728).
- There must be at least one database partition remaining in the database partition group. The last database partition cannot be dropped from a database partition group (SQLSTATE 428C0).
- If neither the LIKE DBPARTITIONNUM clause nor the WITHOUT TABLESPACES clause is specified when adding a database partition, the default is to use the lowest database partition number of the existing database partitions in the database partition group (say it is *2*) and proceed as if LIKE DBPARTITIONNUM *2* had been specified. For an existing database partition to be used as the default, it must have containers defined for all the table spaces in the database partition group (column IN_USE of SYSCAT.DBPARTITIONGROUPDEF is not 'T').

Notes

- When a database partition is added to a database partition group, a catalog entry is made for the database partition (see SYSCAT.DBPARTITIONGROUPDEF). The distribution map is changed immediately to include the new database partition, along with an indicator (IN_USE) that the database partition is in the distribution map if either:
  - no table spaces are defined in the database partition group or
  - no tables are defined in the table spaces defined in the database partition group and the WITHOUT TABLESPACES clause was not specified.

  The distribution map is not changed and the indicator (IN_USE) is set to indicate that the database partition is not included in the distribution map if either:
  - Tables exist in table spaces in the database partition group or
  - Table spaces exist in the database partition group and the WITHOUT TABLESPACES clause was specified (unless all of the table spaces are defined to use automatic storage, in which case the WITHOUT TABLESPACES clause is ignored)

  To change the distribution map, the REDISTRIBUTE DATABASE PARTITION GROUP command must be used. This redistributes any data, changes the distribution map, and changes the indicator. Table space containers need to be added before attempting to redistribute data if the WITHOUT TABLESPACES clause was specified.
- When a database partition is dropped from a database partition group, the catalog entry for the database partition (see SYSCAT.DBPARTITIONGROUPDEF) is updated. If there are no tables defined in the table spaces defined in the database partition group, the distribution map is changed immediately to exclude the dropped database partition and the entry for the database partition in the database partition group is dropped. If tables exist, the distribution map is not changed and the indicator (IN_USE) is set to indicate that the database partition is waiting to be dropped. The REDISTRIBUTE DATABASE PARTITION GROUP command must be used to redistribute the data and drop the entry for the database partition from the database partition group.
- ***Compatibilities***
  - For compatibility with previous versions of DB2:
    - NODE can be specified in place of DBPARTITIONNUM
    - NODES can be specified in place of DBPARTITIONNUMS
    - NODEGROUP can be specified in place of DATABASE PARTITION GROUP

Example

Assume that you have a six-partition database that has the following database partitions: 0, 1, 2, 5, 7, and 8. Two database partitions (3 and 6) are added to the system.

- Assume that you want to add database partitions 3 and 6 to a database partition group called MAXGROUP, and have table space containers like those on database partition 2. The statement is as follows:

```
ALTER DATABASE PARTITION GROUP MAXGROUP
  ADD DBPARTITIONNUMS (3,6)LIKE DBPARTITIONNUM 2
```

- Assume that you want to drop database partition 1 and add database partition 6 to database partition group MEDGROUP. You will define the table space containers separately for database partition 6 using ALTER TABLESPACE. The statement is as follows:

```
ALTER DATABASE PARTITION GROUP MEDGROUP
  ADD DBPARTITIONNUM(6)WITHOUT TABLESPACES
  DROP DBPARTITIONNUM(1)
```

CREATE DATABASE PARTITION GROUP

The CREATE DATABASE PARTITION GROUP statement defines a new database partition group within the database, assigns database partitions to the database partition group, and records the database partition group definition in the system catalog.

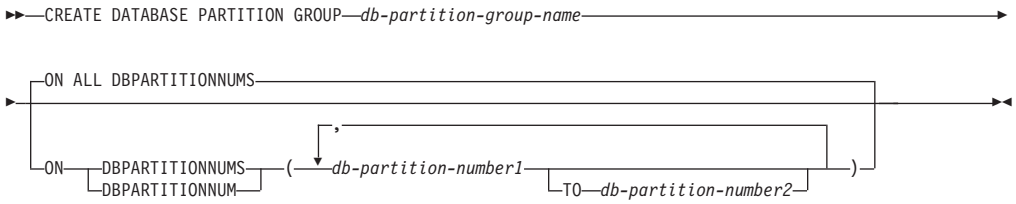
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include SYSCTRL or SYSADM authority.

Syntax



Description

db-partition-group-name

Names the database partition group. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The *db-partition-group-name* must not identify a database partition group that already exists in the catalog (SQLSTATE 42710). The *db-partition-group-name* must not begin with the characters 'SYS' or 'IBM' (SQLSTATE 42939).

ON ALL DBPARTITIONNUMS
Specifies that the database partition group is defined over all database partitions defined to the database (`db2nodes.cfg` file) at the time the database partition group is created.

If a database partition is added to the database system, the ALTER DATABASE PARTITION GROUP statement should be issued to include this new database partition in a database partition group (including IBMDEFAULTGROUP). Furthermore, the REDISTRIBUTE DATABASE PARTITION GROUP command must be issued to move data to the database partition.

ON DBPARTITIONNUMS
Specifies the database partitions that are in the database partition group. DBPARTITIONNUM is a synonym for DBPARTITIONNUMS.

db-partition-number1
Specify a database partition number. (A *node-name* of the form `NODE`*nnnnn* can be specified for compatibility with the previous version.)

TO *db-partition-number2*
Specify a range of database partition numbers. The value of *db-partition-number2* must be greater than or equal to the value of *db-partition-number1* (SQLSTATE 428A9). All database partitions between and including the specified database partition numbers are included in the database partition group.

Rules

- Each database partition specified by number must be defined in the `db2nodes.cfg` file (SQLSTATE 42729).
- Each *db-partition-number* listed in the ON DBPARTITIONNUMS clause must be appear at most once (SQLSTATE 42728).
- A valid *db-partition-number* is between 0 and 999 inclusive (SQLSTATE 42729).

Notes

- This statement creates a distribution map for the database partition group. A distribution map identifier (PMAP_ID) is generated for each distribution map. This information is recorded in the catalog and can be retrieved from SYSCAT.DBPARTITIONGROUPS and SYSCAT.PARTITIONMAPS. Each entry in the distribution map specifies the target database partition on which all rows that are hashed reside. For a single-partition database partition group, the corresponding distribution map has only one entry. For a multiple partition database partition group, the corresponding distribution map has 4096 entries, where the database partition numbers are assigned to the map entries in a round-robin fashion, by default.
- ***Compatibilities***
  - For compatibility with previous versions of DB2:
    - NODE can be specified in place of DBPARTITIONNUM
    - NODES can be specified in place of DBPARTITIONNUMS
    - NODEGROUP can be specified in place of DATABASE PARTITION GROUP

Examples

Assume that you have a partitioned database with six database partitions defined as 0, 1, 2, 5, 7, and 8.

- Assume that you want to create a database partition group called MAXGROUP on all six database partitions. The statement is as follows:

```
CREATE DATABASE PARTITION GROUP MAXGROUP ON ALL DBPARTITIONNUMS
```

- Assume that you want to create a database partition group called MEDGROUP on database partitions 0, 1, 2, 5, and 8. The statement is as follows:

```
CREATE DATABASE PARTITION GROUP MEDGROUP
  ON DBPARTITIONNUMS( 0 TO 2, 5, 8)
```

- Assume that you want to create a single-partition database partition group MINGROUP on database partition 7. The statement is as follows:

```
CREATE DATABASE PARTITION GROUP MINGROUP
  ON DBPARTITIONNUM (7)
```

Chapter 34. Supported administrative SQL routines and views

ADMIN_CMD stored procedure and associated administrative SQL routines

GET STMM TUNING DBPARTITIONNUM command using the ADMIN_CMD procedure

Used to read the catalog tables to report the user preferred self tuning memory manager (STMM) tuning database partition number and current STMM tuning database partition number.

Authorization

SYSADM or DBADM authority

Required connection

Database

Command syntax

Example

```
CALL SYSPROC.ADMIN_CMD( 'get stmm tuning dbpartitionnum' )
```

The following is an example of output from this query.

```
Result set 1
  --------------

  USER_PREFERRED_NUMBER CURRENT_NUMBER
  --------------------- --------------
                      2              2

  1 record(s) selected.

  Return Status = 0
```

Usage notes

The user preferred self tuning memory manager (STMM) tuning database partition number (USER_PREFERRED_NUMBER) is set by the user and specifies the database partition on which the user wishes to run the memory tuner. While the database is running, the tuning partition is updated asynchronously a few times an hour. As a result, it is possible that the CURRENT_NUMBER and USER_PREFERRED_NUMBER returned are not in sync after an update of the user preferred STMM partition number. To resolve this, either wait for the CURRENT_NUMBER to be updated asynchronously, or stop and start the database to force the update of CURRENT_NUMBER.

© Copyright IBM Corp. 1993, 2009

UPDATE STMM TUNING DBPARTITIONNUM command using the ADMIN_CMD procedure

Update the user preferred self tuning memory manager (STMM) tuning database partition.

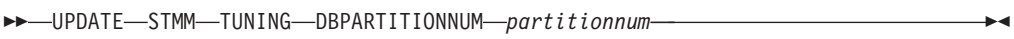
Authorization

SYSADM or DBADM authority

Required connection

Database

Command syntax



Command parameter

partitionnum
partitionnum is an integer. If -1 or a nonexistent database partition number is used, DB2 will automatically select an appropriate database partition on which to run the STMM memory tuner.

Example

Update the user preferred self tuning memory manager (STMM) tuning database partition to database partition 3.

```
CALL SYSPROC.ADMIN_CMD( 'update stmm tuning dbpartitionnum 3' )
```

Usage notes

The STMM tuning process periodically checks for a change in the user preferred STMM tuning database partition number value. The STMM tuning process will move to the user preferred STMM tuning database partition if *partitionnum* exists and is an active database partition. Once this command changes the STMM tuning database partition number an immediate change is made to the current STMM tuning database partition number.

Command execution status is returned in the SQLCA resulting from the CALL statement.

This command commits its changes in the ADMIN_CMD procedure.

Configuration administrative SQL routines and views

DB_PARTITIONS

The DB_PARTITIONS table function returns the contents of the db2nodes.cfg file in table form.

Syntax

The schema is SYSPROC.

Authorization

EXECUTE privilege on the DB_PARTITIONS table function.

Table function parameters

The function has no input parameters.

Example

Retrieve information from a 3 logical partition database.

```
SELECT * FROM TABLE(DB_PARTITIONS()) AS T
```

The following is an example of output from this query.

```
PARTITION_NUMBER HOST_NAME                PORT_NUMBER SWITCH_NAME
---------------- --------------------...- ----------- -----------...-
               0 jessicae.torolab.ibm.com           0 jessicae
               1 jessicae.torolab.ibm.com           1 jessicae
               2 jessicae.torolab.ibm.com           2 jessicae

  3 record(s) selected.
```

Information returned

Table 42. Information returned by the DB_PARTITIONS table function

Column name	Data type	Description
PARTITION_NUMBER	SMALLINT	A unique number between 0 and 999 that identifies a database partition server in a partitioned database environment.
HOST_NAME	VARCHAR(128)	The TCP/IP host name of the database partition server.
PORT_NUMBER	SMALLINT	The port number for the database partition server.
SWITCH_NAME	VARCHAR(128)	The name of a high speed interconnect, or switch, for database partition communications.

Stepwise redistribute administrative SQL routines

STEPWISE_REDISTRIBUTE_DBPG procedure - Redistribute part of database partition group

The STEPWISE_REDISTRIBUTE_DBPG procedure redistributes part of the database partition group according to the input specified for the procedure, and the setting file created or updated by the SET_SWRD_SETTINGS procedure.

Syntax

The schema is SYSPROC.

Procedure parameters

inDBPGroup
An input argument of type VARCHAR (128) that specifies the name of the target database partition group.

inStartingPoint
An input argument of type SMALLINT that specifies the starting point to use. If the parameter is set to a positive integer and is not NULL, the STEPWISE_REDISTRIBUTE_DBPG procedure uses this value instead of using the *nextStep* value specified in the setting file. This is a useful option when you want to rerun the STEPWISE_REDISTRIBUTE_DBPG procedure from a particular step. If the parameter is set to `NULL`, the *nextStep* value is used.

inNumSteps
An input argument of type SMALLINT that specifies the number of steps to run. If the parameter is set to a positive integer and is not `NULL`, the STEPWISE_REDISTRIBUTE_DBPG procedure uses this value instead of using the*stageSize* value specified in the setting file. This is a useful option when you want to rerun the STEPWISE_REDISTRIBUTE_DBPG procedure with a different number of steps than what is specified in the settings. For example, if there are five steps in a scheduled stage, and the redistribution process failed at step 3, the STEPWISE_REDISTRIBUTE_DBPG procedure can be called to run the remaining three steps once the error condition has been corrected. If the parameter is set to `NULL`, the *stageSize* value is used. The value `-2` can be used in this procedure to indicate that the number is unlimited.

Note: There is no parameter for specifying the equivalent of the **NOT ROLLFORWARD RECOVERABLE** option on the REDISTRIBUTE DATABASE PARTITION GROUP command. Logging is always performed for row data redistribution performed when the STEPWISE_REDISTRIBUTE_DBPG procedure is used.

Authorization

- EXECUTE privilege on the STEPWISE_REDISTRIBUTE_DBPG procedure
- SYSADM, SYSCTRL or DBADM

Example

Redistribute the database partition group "IBMDEFAULTGROUP" according to the redistribution plan stored in the registry by the SET_SWRD_SETTINGS procedure. It is starting with step 3 and redistributes the data until 2 steps in the redistribution plan are completed.

```
CALL SYSPROC.STEPWISE_REDISTRIBUTE_DBPG('IBMDEFAULTGROUP', 3, 2)
```

For a full usage example of the stepwise redistribute procedures, refer to STEPWISE_REDISTRIBUTE_DBPG procedure.

Usage notes

If the registry value for processState is updated to 1 using the SET_SWRD_SETTINGS procedure after the STEPWISE_REDISTRIBUTE_DBPG procedure execution is started, the process stops at the beginning to the next step and a warning message is returned.

Since SQL COMMIT statement is called by the redistribute process, running the redistribute process under a Type-2 connection is not supported.

Part 6. Appendixes

© Copyright IBM Corp. 1993, 2009

Appendix A. Install as non-root user

Installing a DB2 product as a non-root user

Most DB2 database products can be installed as a non-root user.

Before you begin

Before you install any DB2 database product as a non-root user, you should be aware of the differences between root installations and non-root installations, and the limitations of non-root installations. For more information on `non-root` installation, see "Non-root installation overview (Linux and UNIX)".

Prerequisites for installing a DB2 database product as a non-root user are:

- You must be able to mount the installation DVD, or have it mounted for you.
- You must have a valid user ID that can be used as the owner of a DB2 instance.
  User IDs have the following restrictions and requirements:
  - Must have a primary group other than guests, admins, users, and local
  - Can include lowercase letters (a–z), numbers (0–9), and the underscore character (_)
  - Cannot be longer than eight characters
  - Cannot begin with IBM, SYS, SQL, or a number
  - Cannot be a DB2 reserved word (USERS, ADMINS, GUESTS, PUBLIC, or LOCAL), or an SQL reserved word
  - Cannot use any User IDs with root privilege for the DB2 instance ID, DAS ID or fenced ID.
  - Cannot include accented characters
  - If existing user IDs are specified instead of creating new user IDs, make sure that the user IDs:
    - Are not locked
    - Do not have expired passwords
- The hardware and software prerequisites that exist for the product you are installing apply to the non-root user just as they do for root users.
- On AIX Version 5.3, Asynchronous I/O (AIO) must be enabled.
- Your home directory must be a valid DB2 path.
  DB2 installation paths have the following rules:
  - Can include lowercase letters (a–z), uppercase letters (A–Z), and the underscore character (_)
  - Cannot exceed 128 characters
  - Cannot contain spaces
  - Cannot contain non-English characters

About this task

Installing DB2 database products as a non-root user should be transparent to the non-root user. In other words, there is nothing special a non-root user needs to do to install a DB2 database product, other than being logged in as a non-root user.

© Copyright IBM Corp. 1993, 2009

Procedure

To perform a non-root installation:

1. Log in as a non-root user
2. Install your DB2 database product using any of the methods available to you. Options include:
   - The DB2 Setup wizard (GUI install)
   - The db2_install command
   - The db2setup command with a response file (silent install)

   Note: Since non-root users cannot choose the directory where DB2 database products are installed, any FILE keyword in your response file is ignored.
3. After the DB2 database product is installed, you need to open a new login session to use the non-root DB2 instance. Alternatively, you can use the same login session if you source the DB2 instance environment with *$HOME*/sqllib/db2profile (for Bourne shell and Korn shell users) or *$HOME*/sqllib/db2chsrc (for C shell users), where *$HOME* is the non-root user's home directory.

What to do next

After the DB2 database product is installed, you should verify your operating system user process resource limits (ulimits). If the minimum ulimit values are not met, the DB2 engine could encounter unexpected operating resource shortage errors. These errors can lead to a DB2 outage.

Appendix B. Using backup

Using backup

Use the BACKUP DATABASE command to take a copy of a database's data and store it on a different medium in case of failure or damage to the original. You can back up an entire database, database partition, or only selected table spaces.

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 using the ACTIVATE DATABASE statement, 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 NODES 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 is necessary because the backup operation may 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.

You can also use the Command Editor to back up database partitions. Because this approach does not support rollforward recovery, you should back up the database residing on these nodes regularly. You should also keep a copy of the db2nodes.cfg file with any backup copies you take, as protection against possible damage to this file.

© Copyright IBM Corp. 1993, 2009

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 migrated, you must migrate the database before you can back it up.

About this task

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, and you must have at least one backup image of the rest of the nodes in the system (even those 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 the database is not active by issuing the DEACTIVATE DATABASE command.

The backup utility can be invoked through the command line processor (CLP), the Backup Database wizard in the Control Center, by running the ADMIN_CMD procedure with the BACKUP DATABASE parameter, or the *db2Backup* application programming interface (API).

Following is an example of the BACKUP DATABASE command issued through the CLP:

```
db2 backup database sample to c:\DB2Backups
```

Procedure

To open the Backup Database wizard:

1. From the Control Center, expand the object tree until you find the database or table space object that you want to back up.

2. Right-click on the object and select Backup from the pop-up menu. The Backup Database wizard opens. .

What to do next

Detailed information is provided through the contextual help facility within the Control Center

If you performed an offline backup, after the backup completes, you must reactivate the database:

```
ACTIVATE DATABASE sample
```

Appendix C. Partitioned database environment catalog views

SYSCAT.BUFFERPOOLDBPARTITIONS

Each row represents a combination of a buffer pool and a database partition, in which the size of the buffer pool on that partition is different from its default size for other partitions in the same database partition group (as represented in SYSCAT.BUFFERPOOLS).

Table 43. SYSCAT.BUFFERPOOLDBPARTITIONS Catalog View

Column Name	Data Type	Nullable	Description
BUFFERPOOLID	INTEGER		Internal buffer pool identifier.
DBPARTITIONNUM	SMALLINT		Database partition number.
NPAGES	INTEGER		Number of pages in this buffer pool on this database partition.

SYSCAT.DATAPARTITIONEXPRESSION

Each row represents an expression for that part of the table partitioning key.

Table 44. SYSCAT.DATAPARTITIONEXPRESSION Catalog View

Column Name	Data Type	Nullable	Description
TABSCHEMA	VARCHAR (128)		Schema name of the partitioned table.
TABNAME	VARCHAR (128)		Unqualified name of the partitioned table.
DATAPARTITIONKEYSEQ	INTEGER		Expression key part sequence ID, starting from 1.
DATAPARTITIONEXPRESSION	CLOB (32K)		Expression for this entry in the sequence, in SQL syntax.
NULLSFIRST	CHAR (1)		• N = Null values in this expression compare high • Y = Null values in this expression compare low

SYSCAT.DATAPARTITIONS

Each row represents a data partition.

Table 45. SYSCAT.DATAPARTITIONS Catalog View

Column Name	Data Type	Nullable	Description
DATAPARTITIONNAME	VARCHAR (128)		Name of the data partition.
TABSCHEMA	VARCHAR (128)		Schema name of the table to which this data partition belongs.
TABNAME	VARCHAR (128)		Unqualified name of the table to which this data partition belongs.
DATAPARTITIONID	INTEGER		Identifier for the data partition.

© Copyright IBM Corp. 1993, 2009

Table 45. SYSCAT.DATAPARTITIONS Catalog View (continued)

Column Name	Data Type	Nullable	Description
TBSPACEID	INTEGER	Y	Identifier for the table space in which this data partition is stored. Null when STATUS is 'I'.
PARTITIONOBJECTID	INTEGER	Y	Identifier for the data partition within the table space.
LONG_TBSPACEID	INTEGER	Y	Identifier for the table space in which long data is stored. Null when STATUS is 'I'.
ACCESS_MODE	CHAR (1)		Access restriction state of the data partition. These states only apply to objects that are in set integrity pending state or to objects that were processed by a SET INTEGRITY statement. Possible values are: • D = No data movement • F = Full access • N = No access • R = Read-only access
STATUS	VARCHAR (32)		• A = Data partition is newly attached • D = Data partition is detached • I = Detached data partition whose entry in the catalog is maintained only during asynchronous index cleanup; rows with a STATUS value of 'I' are removed when all index records referring to the detached partition have been deleted • Empty string = Data partition is visible (normal status) Bytes 2 through 32 are reserved for future use.
SEQNO	INTEGER		Data partition sequence number (starting from 0).
LOWINCLUSIVE	CHAR (1)		• N = Low key value is not inclusive • Y = Low key value is inclusive
LOWVALUE	VARCHAR (512)		Low key value (a string representation of an SQL value) for this data partition.
HIGHINCLUSIVE	CHAR (1)		• N = High key value is not inclusive • Y = High key value is inclusive
HIGHVALUE	VARCHAR (512)		High key value (a string representation of an SQL value) for this data partition.

SYSCAT.DBPARTITIONGROUPDEF

Each row represents a database partition that is contained in a database partition group.

Table 46. SYSCAT.DBPARTITIONGROUPDEF Catalog View

Column Name	Data Type	Nullable	Description
DBPGNAME	VARCHAR (128)		Name of the database partition group that contains the database partition.
DBPARTITIONNUM	SMALLINT		Partition number of a database partition that is contained in the database partition group. A valid partition number is between 0 and 999, inclusive.
IN_USE	CHAR (1)		Status of the database partition. • A = The newly added database partition is not in the distribution map, but the containers for the table spaces in the database partition group have been created; the database partition is added to the distribution map when a redistribute database partition group operation has completed successfully • D = The database partition will be dropped when a redistribute database partition group operation has completed successfully • T = The newly added database partition is not in the distribution map, and it was added using the WITHOUT TABLESPACES clause; containers must be added to the table spaces in the database partition group • Y = The database partition is in the distribution map

SYSCAT.DBPARTITIONGROUPS

Each row represents a database partition group.

Table 47. SYSCAT.DBPARTITIONGROUPS Catalog View

Column Name	Data Type	Nullable	Description
DBPGNAME	VARCHAR (128)		Name of the database partition group.
OWNER	VARCHAR (128)		Authorization ID under which the database partition group was created.
OWNERTYPE	CHAR (1)		• S = The owner is the system • U = The owner is an individual user
PMAP_ID	SMALLINT		Identifier for the distribution map in the SYSCAT.PARTITIONMAPS catalog view.
REDISTRIBUTE_PMAP_ID	SMALLINT		Identifier for the distribution map currently being used for redistribution; -1 if redistribution is currently not in progress.
CREATE_TIME	TIMESTAMP		Creation time of the database partition group.
DEFINER[1]	VARCHAR (128)		Authorization ID under which the database partition group was created.
REMARKS	VARCHAR (254)	Y	User-provided comments, or null.

Table 47. SYSCAT.DBPARTITIONGROUPS Catalog View (continued)

Column Name	Data Type	Nullable	Description

Note:

1. The DEFINER column is included for backwards compatibility. See OWNER.

SYSCAT.PARTITIONMAPS

Each row represents a distribution map that is used to distribute the rows of a table among the database partitions in a database partition group, based on hashing the table's distribution key.

Table 48. SYSCAT.PARTITIONMAPS Catalog View

Column Name	Data Type	Nullable	Description
PMAP_ID	SMALLINT		Identifier for the distribution map.
PARTITIONMAP	BLOB (8192)		Distribution map, a vector of 4096 two-byte integers for a multiple partition database partition group. For a single partition database partition group, there is one entry denoting the partition number of the single partition.

Appendix D. Overview of the DB2 technical information

DB2 technical information is available through the following tools and methods:

- *DB2 Information Center*
  - Topics (Task, concept and reference topics)
  - Help for DB2 tools
  - Sample programs
  - Tutorials
- DB2 books
  - PDF files (downloadable)
  - PDF files (from the DB2 PDF DVD)
  - printed books
- Command line help
  - Command help
  - Message help

Note: The *DB2 Information Center* topics are updated more frequently than either the PDF or the hard-copy books. To get the most current information, install the documentation updates as they become available, or refer to the *DB2 Information Center* at ibm.com®.

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

We value 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 documentation team reads all of your feedback, but cannot respond to you directly. Provide specific examples wherever possible so that we can 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 this email address to contact DB2 Customer Support. If you have a DB2 technical issue that the documentation does not resolve, contact your local IBM service center for assistance.

If you would like to help IBM make the IBM Information Management products easier to use, take the Consumability Survey: http://www.ibm.com/software/data/info/consumability-survey/.

DB2 technical library in hardcopy or PDF format

The following tables describe the DB2 library available from the IBM Publications Center at www.ibm.com/shop/publications/order. English DB2 Version 9.5 manuals in PDF format and translated versions can be downloaded from www.ibm.com/support/docview.wss?rs=71&uid=swg2700947.

© Copyright IBM Corp. 1993, 2009

Although the tables identify books 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.

Note: The *DB2 Information Center* is updated more frequently than either the PDF or the hard-copy books.

Table 49. DB2 technical information

Name	Form Number	Available in print	Last updated
Administrative API Reference	SC23-5842-02	Yes	April, 2009
Administrative Routines and Views	SC23-5843-02	No	April, 2009
Call Level Interface Guide and Reference, Volume 1	SC23-5844-02	Yes	April, 2009
Call Level Interface Guide and Reference, Volume 2	SC23-5845-02	Yes	April, 2009
Command Reference	SC23-5846-02	Yes	April, 2009
Data Movement Utilities Guide and Reference	SC23-5847-02	Yes	April, 2009
Data Recovery and High Availability Guide and Reference	SC23-5848-02	Yes	April, 2009
Data Servers, Databases, and Database Objects Guide	SC23-5849-02	Yes	April, 2009
Database Security Guide	SC23-5850-02	Yes	April, 2009
Developing ADO.NET and OLE DB Applications	SC23-5851-02	Yes	April, 2009
Developing Embedded SQL Applications	SC23-5852-02	Yes	April, 2009
Developing Java Applications	SC23-5853-02	Yes	April, 2009
Developing Perl and PHP Applications	SC23-5854-02	No	April, 2009
Developing User-defined Routines (SQL and External)	SC23-5855-02	Yes	April, 2009
Getting Started with Database Application Development	GC23-5856-02	Yes	April, 2009
Getting Started with DB2 installation and administration on Linux and Windows	GC23-5857-02	Yes	April, 2009
Internationalization Guide	SC23-5858-02	Yes	April, 2009

Table 49. DB2 technical information (continued)

Name	Form Number	Available in print	Last updated
Message Reference, Volume 1	GI11-7855-01	No	April, 2009
Message Reference, Volume 2	GI11-7856-01	No	April, 2009
Migration Guide	GC23-5859-02	Yes	April, 2009
Net Search Extender Administration and User's Guide	SC23-8509-02	Yes	April, 2009
Partitioning and Clustering Guide	SC23-5860-02	Yes	April, 2009
Query Patroller Administration and User's Guide	SC23-8507-01	Yes	April, 2009
Quick Beginnings for IBM Data Server Clients	GC23-5863-02	No	April, 2009
Quick Beginnings for DB2 Servers	GC23-5864-02	Yes	April, 2009
Spatial Extender and Geodetic Data Management Feature User's Guide and Reference	SC23-8508-02	Yes	April, 2009
SQL Reference, Volume 1	SC23-5861-02	Yes	April, 2009
SQL Reference, Volume 2	SC23-5862-02	Yes	April, 2009
System Monitor Guide and Reference	SC23-5865-02	Yes	April, 2009
Text Search Guide	SC23-5866-01	Yes	April, 2009
Troubleshooting Guide	GI11-7857-02	No	April, 2009
Tuning Database Performance	SC23-5867-02	Yes	April, 2009
Visual Explain Tutorial	SC23-5868-00	No	
What's New	SC23-5869-02	Yes	April, 2009
Workload Manager Guide and Reference	SC23-5870-02	Yes	April, 2009
pureXML Guide	SC23-5871-02	Yes	April, 2009
XQuery Reference	SC23-5872-02	No	April, 2009

Table 50. DB2 Connect-specific technical information

Name	Form Number	Available in print	Last updated
Quick Beginnings for DB2® Connect™ Personal Edition	GC23-5839-02	Yes	April, 2009
Quick Beginnings for DB2 Connect Servers	GC23-5840-02	Yes	April, 2009
DB2 Connect User's Guide	SC23-5841-02	Yes	April, 2009

Table 51. Information Integration technical information

Name	Form Number	Available in print	Last updated
Information Integration: Administration Guide for Federated Systems	SC19-1020-01	Yes	March, 2008
Information Integration: ASNCLP Program Reference for Replication and Event Publishing	SC19-1018-02	Yes	March, 2008
Information Integration: Configuration Guide for Federated Data Sources	SC19-1034-01	No	
Information Integration: SQL Replication Guide and Reference	SC19-1030-01	Yes	March, 2008
Information Integration: Introduction to Replication and Event Publishing	SC19-1028-01	Yes	March, 2008

Ordering printed DB2 books

If you require printed DB2 books, you can buy them online in many but not all countries or regions. You can always order printed DB2 books from your local IBM representative. Keep in mind that some softcopy books on the *DB2 PDF Documentation* DVD are unavailable in print. For example, neither volume of the *DB2 Message Reference* is available as a printed book.

Printed versions of many of the DB2 books available on the DB2 PDF Documentation DVD can be ordered for a fee from IBM. Depending on where you are placing your order from, you may be able to order books online, from the IBM Publications Center. If online ordering is not available in your country or region, you can always order printed DB2 books from your local IBM representative. Note that not all books on the DB2 PDF Documentation DVD are available in print.

Note: The most up-to-date and complete DB2 documentation is maintained in the DB2 Information Center at http://publib.boulder.ibm.com/infocenter/db2luw/v9r5.

To order printed DB2 books:

- To find out whether you can order printed DB2 books online in your country or region, check the IBM Publications Center at http://www.ibm.com/shop/publications/order. You must select a country, region, or language to access publication ordering information and then follow the ordering instructions for your location.
- To order printed DB2 books from your local IBM representative:
  1. Locate the contact information for your local representative from one of the following Web sites:
     - The IBM directory of world wide contacts at www.ibm.com/planetwide
     - The IBM Publications Web site at http://www.ibm.com/shop/publications/order. You will need to select your country, region, or

language to the access appropriate publications home page for your location. From this page, follow the "About this site" link.

2. When you call, specify that you want to order a DB2 publication.
3. Provide your representative with the titles and form numbers of the books that you want to order. For titles and form numbers, see "DB2 technical library in hardcopy or PDF format" on page 421.

Displaying SQL state help from the command line processor

DB2 returns an SQLSTATE value for conditions that could be the result of an SQL statement. SQLSTATE help explains the meanings of SQL states and SQL state class codes.

To invoke 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

For DB2 Version 9.5 topics, the DB2 Information Center URL is http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/

For DB2 Version 9 topics, the DB2 Information Center URL is http://publib.boulder.ibm.com/infocenter/db2luw/v9/

For DB2 Version 8 topics, go to the Version 8 Information Center URL at: http://publib.boulder.ibm.com/infocenter/db2luw/v8/

Displaying topics in your preferred language in the DB2 Information Center

The DB2 Information Center attempts to display topics in the language specified in your browser preferences. If a topic has not been translated into your preferred language, the DB2 Information Center displays the topic in English.

- To display topics in your preferred language in the Internet Explorer browser:
  1. In Internet Explorer, click the **Tools** —> **Internet Options** —> **Languages...** button. The Language Preferences window opens.
  2. Ensure your preferred language is specified as the first entry in the list of languages.
     - To add a new language to the list, click the **Add...** button.

       Note: Adding a language does not guarantee that the computer has the fonts required to display the topics in the preferred language.
     - To move a language to the top of the list, select the language and click the **Move Up** button until the language is first in the list of languages.
  3. Clear the browser cache and then refresh the page to display the DB2 Information Center in your preferred language.
- To display topics in your preferred language in a Firefox or Mozilla browser:

1. Select the button in the **Languages** section of the **Tools** —> **Options** —> **Advanced** dialog. The Languages panel is displayed in the Preferences window.
2. Ensure your preferred language is specified as the first entry in the list of languages.
   - To add a new language to the list, click the **Add...** button to select a language from the Add Languages window.
   - To move a language to the top of the list, select the language and click the **Move Up** button until the language is first in the list of languages.
3. Clear the browser cache and then refresh the page to display the DB2 Information Center in your preferred language.

On some browser and operating system combinations, you might have to also change the regional settings of your operating system to the locale and language of your choice.

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.

Updating your locally-installed *DB2 Information Center* 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. Non-Administrative and Non-Root *DB2 Information Centers* always run in stand-alone mode. .
2. Use the update feature to see what updates are available. If there are updates that you would like to 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, you have to mirror the update site to a local file system 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 Vista, the commands listed below must be run as an administrator. To launch a command prompt or graphical tool with full administrator privileges, right-click on the shortcut and then select **Run as administrator**.

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 on **DB2 Information Center** service and select **Stop**.

- On Linux, enter the following command:
  ```
  /etc/init.d/db2icdv95 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 9.5 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/V9.5 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 launches to display the stand-alone Information Center.
3. Click the **Update** button ( ). On the right hand panel of the Information Center, click **Find Updates**. A list of updates for existing documentation displays.
4. To initiate the installation process, check 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 installation directory's doc\bin directory, and run the help_end.bat file:
     ```
     help_end.bat
     ```

     Note: The help_end batch file contains the commands required to safely terminate the processes that were started with the help_start batch file. Do not use `Ctrl-C` or any other method to terminate help_start.bat.
   - On Linux, navigate to the installation directory's doc/bin directory, and run the help_end script:
     ```
     help_end
     ```

     Note: The help_end script contains the commands required to safely terminate the processes that were started with the help_start script. Do not use any other method to terminate the help_start script.
7. Restart the *DB2 Information Center*.
   - On Windows, click **Start** → **Control Panel** → **Administrative Tools** → **Services**. Then right-click on **DB2 Information Center** service and select **Start**.
   - On Linux, enter the following command:
     ```
     /etc/init.d/db2icdv95 start
     ```

The updated *DB2 Information Center* displays the new and updated topics.

DB2 tutorials

The DB2 tutorials help you learn about various aspects of DB2 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/db2help/.

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 on 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.

"Visual Explain" in *Visual Explain Tutorial*
Analyze, optimize, and tune SQL statements for better performance using Visual Explain.

DB2 troubleshooting information

A wide variety of troubleshooting and problem determination information is available to assist you in using DB2 database products.

DB2 documentation
Troubleshooting information can be found in the DB2 Troubleshooting Guide or the Database fundamentals section of the DB2 Information Center. There you will find information on how to isolate and identify problems using DB2 diagnostic tools and utilities, solutions to some of the most common problems, and other advice on how to solve problems you might encounter with your DB2 database products.

DB2 Technical Support Web site
Refer to the DB2 Technical Support Web site 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 DB2 Technical Support Web site at http://www.ibm.com/software/data/db2/support/db2_9/

Terms and Conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.

Personal use: You may reproduce these Publications for your personal, non commercial 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.

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.

Appendix E. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country/region or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, 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 and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

This document may provide links or references to non-IBM Web sites and resources. IBM makes no representations, warranties, or other commitments whatsoever about any non-IBM Web sites or third-party resources that may be referenced, accessible from, or linked from this document. A link to a non-IBM Web site does not mean that IBM endorses the content or use of such Web site or

© Copyright IBM Corp. 1993, 2009

its owner. In addition, IBM is not a party to or responsible for any transactions you may enter into with third parties, even if you learn of such parties (or use a link to such parties) from an IBM site. Accordingly, you acknowledge and agree that IBM is not responsible for the availability of such external sites or resources, and is not responsible or liable for any content, services, products, or other materials on or available from those sites or resources. Any software provided by third parties is subject to the terms and conditions of the license that accompanies that software.

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
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
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 may contain 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.

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 of Sun Microsystems, Inc. in the United States, other countries, 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, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries. Intel trademark information
- 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 plans
  collecting current statistics
    in a partitioned database environment 327
  creating additional indexes
    in a partitioned database environment 335
  creating indexes on columns used to join tables
    in a partitioned database environment 331
  improving
    in a partitioned database environment 323
  output example showing no parallelism 305
  query with no indexes and no statistics
    in a partitioned database environment 324
Add Database Partition Server to an Instance command 385
Add Node API 365
ADMIN_CMD procedure
  supported commands
    GET STMM TUNING DBPARTITIONNUM 403
    UPDATE STMM TUNING DBPARTITIONNUM 404
administration notification log 251
agents
  in a partitioned database 299
AIX
  creating DB2 home file systems 107
  creating required users 110
  distributing commands to ESE workstations 103
  installing DB2 servers 93
  updating environment settings 101
  verifying NFS is running 103
ALTER DATABASE PARTITION GROUP statement 397
ALTER NODEGROUP statement
  see ALTER DATABASE PARTITION GROUP 397
APIs
  sqleaddn 365
  sqlecran 367
  sqledpan 368
  sqledrpn 369
  sqlugrpn 371
attaching data partitions
  description 195
authentication
  partitioned database considerations 5
automatic restart 251

B

BACKUP DATABASE command 413
backup utility
  restrictions 413
books
  printed
    ordering 424
boundary ranges
  specifying 172

C

capacity
  expansion 138
  for each environment 78
catalog nodes 3, 123
catalog statistics
  index cluster ratio 301
catalog tables
  stored on database catalog node 3, 123
catalog views
  BUFFERPOOLDBPARTITIONS 417
  DATAPARTITIONEXPRESSION 417
  DATAPARTITIONS 417
  DBPARTITIONGROUPDEF 418
  DBPARTITIONGROUPS 419
  PARTITIONMAPS 420
change database partition server configuration command 384
choosing
  multidimensional table dimensions 41
clustering
  data 39
  definition 303
clustering indexes
  benefits with partitioned tables 284
  with partitioned tables 284
collocation
  table 4, 10
column expressions
  multidimensional tables 48, 184
commands
  db2adutl
    cross-node recovery example 256
  db2nchg 91, 384
  db2ncrt 385
  db2ndrop 387
  GET STMM TUNING DBPARTITIONNUM 403
  REDISTRIBUTE DATABASE PARTITION GROUP 375
  running in parallel 161
  UPDATE STMM TUNING DBPARTITIONNUM 404
communications
  addresses 99, 139
  connection elapse time 355
  fast communication manager 99, 139
comparison of indexes
  clustering and block-based 39
comparison of tables
  regular and multidimensional clustering 39
compatibility
  partition 10
configuration
  multiple partition 78
configuration parameters
  conn_elapse 355
  fcm_num_buffers 93, 139, 356
  fcm_num_channels 357
  intra_parallel 359
  logarchopt1
    cross-node recovery example 256
  max_connretries 358
  max_querydegree 360
  max_time_diff 358
  partitioned database 3, 123
  start_stop_time 359
  vendoropt
    cross-node recovery example 256

© Copyright IBM Corp. 1993, 2009

conn_elapse configuration parameter 355
connection concentrators
  use of agents in partitioned database 299
connection elapse time configuration parameter 355
connections
  elapse time 355
containers
  adding to SMS table spaces 155
Control Center
  managing database partitions 191
controlling the rah command 164
Coordinated Universal Time 358
coordinator partition 77
crash recovery 251
create database at node API 367
CREATE DATABASE PARTITION GROUP statement 400
CREATE NODEGROUP statement
  CREATE DATABASE PARTITION GROUP statement 400
CREATE TABLE statement
  OVERFLOW clause 35, 36
creating
  multidimensional tables 48, 184
  required users on AIX 110
  required users on Linux 109
cross-node database recovery example 256
CURRENT DBPARTITIONNUM special register 390

D

data
  distribution 77
  organization schemes
    Informix comparison 19
    overview 14
  partitioning 4
  redistribution
    altering database partition group 191
    determining need 341
    event logging and recovery 345
    log space requirements 344
    overview 339, 342
    Redistribute Data wizard 343
    REDISTRIBUTE DATABASE PARTITION GROUP command 375
  table distribution 14
data partition elimination 273
data partitions
  adding 202
  altering 194
  attaching
    overview 193, 195
    scenario 206
  attributes 201
  creating 172
  detaching
    overview 193, 199
    scenario 206
  dropping 204
  overview 11, 13, 14
  range definition 172
  rolling in data
    overview 193, 195
    scenario 206
  rolling out data
    overview 193, 199
    scenario 206
data partitions *(continued)*
  rotating
    scenario 205
data servers
  capacity management 138
data types
  partition compatibility 389
database configuration file
  changing 191
database configuration parameters
  autorestart 251
database manager
  start timeout 359
  stop timeout 359
database partition compatibility
  overview 389
database partition groups
  adding partitions 397
  changing 191
  collocation 7
  creating 143, 400
  data location determination 7
  designing 7
  distribution map creation 400
  dropping partitions 397
  IBMDEFAULTGROUP 169
  initial 143
  overview 5
  query optimization impact 314
  tables 169
database partition servers
  dropping 155
  enabling communications on UNIX 141
  installing using a response file 116
  issuing commands 158, 265
  multiple logical nodes 134
  multiple logical partitions 134
  recovering from failure 255
  specifying 132
Database Partitioning Feature (DPF)
  enabling communications 141
  overview 77
database partitions
  adding
    overview 146
    running system 147
    stopped system (UNIX) 148
    stopped system (Windows) 147
  adding using a wizard 153
  catalog 3, 123
  changing on WIndows 153
  database configuration updates 191
  dropping from instance 156
  managing 144
    from the Control Center 191
  overview 77
  synchronizing clocks 262
databases
  altering database partition groups 191
  configuring across multiple partitions 353
  creating
    partitioned database environments 3, 123
  data distribution
    changing 191
  data partitioning enabling 3, 123
  rebuilding
    partitioned 255

DATAPARTITIONNUM scalar function 393
DB_PARTITIONS table function 404
DB2 Information Center
  languages 425
  updating 426
  versions 425
  viewing in different languages 425
DB2 servers
  installing
    Linux 93
    UNIX 93
    Windows 89
  partitioned
    Windows environment preparation 91
DB2 Setup wizard
  installing DB2 servers
    Linux 96
    UNIX 96
  installing DB2 servers on UNIX 96
db2_all command
  overview 159, 162
  partitioned database environments 158, 265
  specifying 160
db2_call_stack command 162
db2_kill command 162
DB2_NUM_FAILOVER_NODES registry variable 354
DB2_PARTITIONEDLOAD_DEFAULT registry variable
  description 354
db2adutl command
  cross-node recovery example 256
db2Backup API
  backing up data 413
DB2CHGPWD_EEE registry variable 354
db2exfmt command
  output samples 305
db2expln command
  output samples
    description 305
    multipartition plan with full parallelism 311
    multipartition plan with inter-partition parallelism 309
    no parallelism 305
    single-partition plan with intra-partition parallelism 307
db2fcmr daemons 99, 139
db2fcms daemons 99, 139
db2nchg command
  changing database partition server configurations 153
  description 384
db2ncrt command
  adding database partition servers 152
  description 385
db2ndrop command
  description 387
  dropping database partition servers 155
db2nlist command 151
db2nodes.cfg file
  ALTER DATABASE PARTITION GROUP statement 397
  CREATE DATABASE PARTITION GROUP statement 400
  creating 124
  DBPARTITIONNUM function 394
  format 126
  netname field 91
  updating 133
DB2PORTRANGE registry variable 354
db2start addnode command 152
DBPARTITIONNUM function
  description 394
declustering
  partial 4, 77
Design Advisor
  migrating single-partition to multiple-partition databases 287
detached data partitions
  attributes 201
  description 199
dimensions
  multidimensional tables 41
distribution keys
  description 9
  loading data 217
  partitioned database environments 169
distribution maps
  description 7
documentation
  overview 421
  PDF 421
  printed 421
  terms and conditions of use 428
DPF (Database Partitioning Feature)
  See Database Partitioning Feature (DPF) 77
drop database on database partition server API 368
drop database partition server from an instance command 387
Drop Partitions launchpad 156
dropping
  database partitions using launchpad 156

E

environment variables
  $RAHBUFDIR 161
  $RAHBUFNAME 161
  $RAHENV 164
  rah command 164
  RAHDOTFILES 165
error messages
  partitioned databases 150
event monitors
  creating
    partitioned database 247

F

failed database partition server 252
failure transaction 251
FCM (Fast Communications Manager)
  channels 357
  enabling communications between database partition servers 141
  message buffers 93, 139
  monitor elements
    fcm_num_channels 357
  overview 93, 139
  port numbers 141
  service entry syntax 140
  Windows 93, 139
fcm_num_buffers configuration parameter 93, 139, 356
fcm_num_channel configuration parameter 357
file sets
  db2fcmr daemons 99, 139
  db2fcms daemons 99, 139
  description 99, 139

file systems
creating for a partitioned DB2 server
Linux 105
fragment by expression
comparison with table partitioning 19
fragment elimination
see data partition elimination 273
free space control record (FSCR)
in MDC tables 301
functions
scalar
DBPARTITIONNUM 394
NODENUMBER (see DBPARTITIONNUM) 394
table functions
DB_PARTITIONS 404

G

Get Row Distribution Number API 371
GET STMM TUNING DBPARTITIONNUM command
using ADMIN_CMD 403
global snapshots on partitioned database systems 246
guidelines
range-clustered tables 35, 182

H

hardware
parallelism 78
partitions 78
processors 78
hash partitioning 4
help
configuring language 425
SQL statements 425
highlighting conventions xiii
home file system
AIX 107
how this book is structured ix
HP-UX
installing
DB2 servers 93
Network File System (NFS)
verifying that it is running 103

I

I/O
parallelism 73
IBMCATGROUP database partition group 143
IBMDEFAULTGROUP database partition group 143
IBMTEMPGROUP database partition group 143
indexes
behavior on partitioned tables 281
block
scan lock mode 292
cluster ratio 301
clustering 284
managing
for MDC tables 301
on table columns in a partitioned database environment 335
with partitioned tables 281
installation
methods 94
installing
database partition servers
response files 116
database partition servers using response files 115
DB2 Enterprise Server Edition (Windows) 91
DB2 products as a non-root user 411
methods 94
updating AIX environment settings 101
instances
adding partition servers 152
listing database partition servers 151
partition servers
changing 153
dropping 155
inter-partition parallelism
db2expln tool
output sample 309
output samples 311
used with intra-partition parallelism 73
inter-partition query parallelism
enabling 136
inter-query parallelism 73
intra_parallel database manager configuration parameter 359
intra-partition parallelism
db2expln tool
output sample 307
output samples 311
enabling 137
optimization strategies 303
used with inter-partition parallelism 73
intra-query parallelism 73

J

joins
broadcast inner-table 316
broadcast outer-table 316
collocated 316
overview 313
partitioned database environments 316
table-queue strategy in partitioned databases 314
types
directed inner-table 316
directed outer-table 316

K

keys
distribution 9
table partitioning 24

L

large objects (LOBs)
behavior 170
with partitioned tables 170
licenses
overview 77
Linux
creating
file system for partitioned DB2 servers 105
creating required users 109
default port ranges 141
installing
DB2 servers 93
DB2 Setup wizard 96

Linux *(continued)*
verifying NFS is running 103
LOAD command
in a partitioned database environment 220, 236
LOAD QUERY command
in a partitioned database environment 225
load utility
parallelism 211
loading
configuration options 228
database partitions 219
examples
partitioned database environments 233
partitioned database sessions 233
into database partitions 217
into partitioned tables 26, 212
multidimensional clustered (MDC) tables 211
partitioned database environments 228
lock modes
MDC (multidimensional clustering) tables
table and RID index scans 289
locks
behavior on partitioned tables 296
block index-scan modes 292
discrete
overview 35
range-clustered tables 37
log file space
required for data redistribution 344
logarchopt1 configuration parameter
cross-node recovery example 256
logical database partitions 78
logical nodes
database partition servers 132, 134

M

machine list
for partitioned database environment 132
materialized query tables (MQTs)
behavior 177
replicated 29
replicated, in partitioned databases 321
with partitioned tables 177
max_connretries configuration parameter 358
max_querydegree configuration parameter 360
max_time_diff configuration parameter 358
maximum query degree of parallelism configuration parameter 360
maximum time difference among nodes configuration parameter 358
MDC 57
MDC (multidimensional clustering) tables 48, 184
choosing dimensions 41
description 39
lock modes
table and RID index scans 289
management of tables and indexes 301
optimization strategies 278
rollout deletes 278
with partitioned tables 29, 68, 269
MDC tables
block index considerations 55
block indexes 55
block indexes and query performance 60
block maps 66
deleting from 68
MDC tables *(continued)*
load considerations 53
loading considerations 54
maintaining clustering automatically 64
updating 68
memory tuner
partitioned database environments 351
message buffers
Fast Communications Manager (FCM) 93, 139
migration
partitioned database environment 237
monitoring
data partitions 239
rah processes 168
monotonicity 48, 184
moving data
to multidimensional tables 48, 184
MPP environment 78
MQTs (materialized query tables)
replicated 29
replicated, in partitioned databases 321
multi-partition databases
database partition group 5
migrating from single-partition databases
Design Advisor 287
multidimensional clustered tables 57
multidimensional clustering (MDC) tables
choosing dimensions 41
creating 48, 184
density of values 41
in SMS table spaces 48, 184
load considerations 211
moving data to 48, 184
using column expressions as dimensions 48, 184
multiple logical nodes
configuring 135
multiple partition configurations 78

N

NFS (Network File System)
verifying operation 103
node configuration files
creating 124
description 126
updating (UNIX) 133
node connection retries configuration parameter 358
nodegroups (database partition groups)
creating 143
NODENUMBER function
DBPARTITIONNUM 394
nodes 77
connection elapse time 355
FCM daemon (UNIX) 99, 139
maximum time difference among 358
synchronization 262
non-root installations
installing 411
notices 431

O

optimization
intra-partition parallelism 303
joins
definition 313

optimization *(continued)*
joins *(continued)*
partitioned database 316
partitioned tables 273
strategies for MDC tables 278
ordering DB2 books 424
organizing data
approaches 14

P

parallelism
backups 73
configuration parameters
intra_parallel 359
max_querydegree 360
hardware environments 78
I/O
overview 73
index creation 73
inter-partition 73
intra-partition
enabling 137
optimization strategies 303
overview 73
load utility 73, 211
overview 73
partitioned database environments 77
partitions 78
processors 78
query 73
partial declustering
overview 77
partitioned database environments
creating 3, 123
dropping partitions 151
duplicate machine entries 132
event monitoring 247
global snapshots 246
installation verification
UNIX 119
Windows 119
joins
methods 316
strategies 314
loading data
migration 236
monitoring 225
overview 217, 219
restrictions 220
version compatibility 236
machine list
duplicate entry elimination 132
specifying 132
migrating 236
node addition errors 150
overview 4, 77
partition compatibility 10, 389
rebuilding databases 255
redistributing data 342, 345
replicated materialized query tables 321
scenario 157
self-tuning memory 349, 351
setting up 3, 113, 123
transactions
failure recovery 252
version compatibility 236
partitioned tables
adding data partitions 193, 202
altering 193, 194
attaching partitions 193, 195
converting 196
creating 171, 172
data ranges 172
detached data partitions
attributes 201
detaching data partitions 193, 199, 204
indexes 281
large objects (LOBs) 170
loading 26, 175, 212
locking 296
materialized query tables (MQTs) 177
migrating
pre-Version 9.1 196
tables 175
views 175
mismatches 196
multidimensional clustering (MDC) tables 29, 68, 269
optimization 273
overview 11, 12
reorganizing 239
restrictions 11, 194
rolling in data partitions 193, 195
rolling out data partitions 193
scenarios
attaching and detaching data partitions 206
rolling in and rolling out data partitions 206
rotating data 205
partitioning keys
overview 24
partitioning maps
creating for database partition groups 400
partitions
processors
multiple 78
single 78
performance
catalog information 3, 123
point of consistency
database 251
port number ranges
Linux
availability 105, 141
default 141
UNIX
availability 105, 141
default 141
Windows
defining 152
prefix sequences 162
problem determination
information available 428
tutorials 428
procedures
STEPWISE_REDISTRIBUTE_DBPG 346, 406
processors
adding 138

Q

queries
multidimensional clustering 41
parallelism 73

query optimization
effect of database partition groups 314

R

rah command
controlling 164
description 162
determining problems 166
environment variables 164
introduction 158, 265
monitoring processes 168
overview 159
prefix sequences 162
RAHCHECKBUF environment variable 161
RAHDOTFILES environment variable 165
RAHOSTFILE environment variable 132
RAHOSTLIST environment variable 132
RAHWAITTIME environment variable 168
recursively invoked 161
running commands in parallel 161
setting the default environment profile 168
specifying
as a parameter or response 160
database partition server list 132
RAHCHECKBUF environment variable 161
RAHDOTFILES environment variable 165
RAHOSTFILE environment variable 132
RAHOSTLIST environment variable 132
RAHTREETHRESH environment variable 161
RAHWAITTIME environment variable 168
range partitioning
see data partition 11
see data partitions 13
see table partitioning 12
range-clustered tables
access path determination 35, 182
advantages 35
algorithms 180
description 35
differences from regular tables 181
guidelines 35, 182
incompatibilities 35
indexes 181
out-of-range record keys 35, 36
scenarios 183
table locks 35, 37
range-partitioned tables
see partitioned tables 11
ranges
defining for data partitions 172
generating 172
restrictions 172
recovering
from failure of database partition server 255
recovery
crash 251
cross-node example 256
two-phase commit protocol 252
REDISTRIBUTE DATABASE PARTITION GROUP command 375
redistribute utility
restrictions 339
redistributing data 343
across database partitions 191
necessity 341
procedures 406
redistributing data *(continued)*
step-wise redistribute procedures 346
redistribution 345
registry variables
DB2_FCM_SETTINGS 354
DB2_NO_MPFA_FOR _NEW_DB 48, 184
DB2_NUM_FAILOVER_NODES 354
DB2_PARTITIONEDLOAD__DEFAULT 354
DB2CHGPWD_ESE 354
DB2PORTRANGE 354
replicated materialized query tables 29
response files
installation
database partition servers 115, 116
RESTART DATABASE command 251
restarting a load operation
multi-partition database load operations 227

S

scalability
environments 78
scenarios
multidimensional clustered tables 57
range-clustered tables 183
self tuning memory
enabling
non-uniform environments 351
partitioned database environments 349, 351
settings
default environment profile for rah 168
SIGTTIN message 160
single partition
multiple processor environment 78
single processor environment 78
SMP cluster environment 78
SMS (system managed space)
table spaces
adding containers 155
snapshot monitoring
interpreting output for data partitions 239
on data partitions 239
on partitioned database systems 246
Solaris Operating System
installing DB2 servers 93
verifying NFS is running 103
special registers
CURRENT DBPARTITIONNUM 390
CURRENT NODE (see CURRENT DBPARTITIONNUM) 390
SQL statements
ALTER DATABASE PARTITION GROUP 397
ALTER NODEGROUP (see ALTER DATABASE PARTITION GROUP) 397
CREATE DATABASE PARTITION GROUP 400
CREATE NODEGROUP (see CREATE DATABASE PARTITION GROUP) 400
displaying help 425
sqleaddn API 365
sqlecran API 367
sqledpan API 368
sqledrpn API 369
sqlugrpn API 371
start and stop timeout configuration parameter 359
start_stop_time configuration parameter 359
stdin 160
STEPWISE_REDISTRIBUTE_DBPG procedure 406

STEPWISE_REDISTRIBUTE_DBPG procedure *(continued)*
using to redistributing data 346
synchronization
database partition 262
node 262
recovery considerations 262

T

table partitioning
benefits 12
description 12
table spaces
creating
in database partition groups 29, 144
tables
altering partitioned tables 202, 204
collocation 4, 10
converting 175
creating
in partitioned databases 169
materialized query tables 177
MDC (multidimensional clustering) tables 29, 68, 269
migrating into partitioned tables 175
multidimensional clustering 301
multidimensional clustering (MDC) 39
partitioned 12
partitioned tables 11, 29, 68, 177, 269
queues, for join strategies in partitioned databases 314
range-clustered 35, 182
termination
load operations
in multi-partition databases 227
terms and conditions
use of publications 428
time
difference among nodes, maximum 358
transactions
failure recovery
n the failed database partition server 252
on active database partition server 252
reducing the impact of failure 251
troubleshooting
description 265
online information 428
tutorials 428
tuning partition
determining 351
tutorials
problem determination 428
troubleshooting 428
Visual Explain 428
two-phase commit
protocol 252

U

union
all views, converting 175
uniprocessor environment 78
UNIX
default port ranges 141
installing
using the DB2 Setup wizard 96
updating the node configuration file 133
verifying a partitioned database server installation 119
UPDATE STMM TUNING DBPARTITIONNUM Command
using ADMIN_CMD 404
updates
DB2 Information Center 426
db2nodes.cfg (UNIX) 133
node configuration file 133
users
creating required users on AIX 110
creating required users on Linux 109
utility parallelism 73

V

vendoropt configuration parameter
cross-node recovery example 256
verifying
port range availability
Linux 105
UNIX 105
Visual Explain
tutorial 428

W

who should use this book ix
Windows operating systems
database partitions
adding 147
installation verification
partitioned database environment 119
installing
DB2 servers (with DB2 Setup wizard) 89
wizards
Add Database Partitions wizard 153

Printed in USA

SC23-5860-02

Spine information:

DB2 Version 9.5 for Linux, UNIX, and Windows

Partitioning and Clustering Guide