IBM

Database Partitioning, Table Partitioning, and MDC for DB2 9

- Differentiating database partitioning, table partitioning, and MDC
- Examining implementation examples
- Discussing best practices

Whei-Jen Chen
Alain Fisher
Aman Lalla
Andrew D McLauchlan
Doug Agnew

Redbooks

ibm.com/redbooks

International Technical Support Organization

Database Partitioning, Table Partitioning, and MDC for DB2 9

August 2007

SG24-7467-00

Note: Before using this information and the product it supports, read the information in "Notices" on page vii.

First Edition (August 2007)

This edition applies to DB2 Enterprise Server Edition Version 9 for Linux, UNIX, and Windows.

© Copyright International Business Machines Corporation 2007. All rights reserved.
Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices vii
Trademarks viii

Preface ix
The team that wrote this book ix
Acknowledgement xi
Become a published author xi
Comments welcome xi

Chapter 1. Introduction to partitioning technologies 1
1.1 Databases and partitioning 2
1.1.1 Database concepts 2
1.2 Table partitioning 8
1.2.1 Concepts 8
1.3 Multi-dimensional clustering 9
1.3.1 Concepts 9

Chapter 2. Benefits and considerations of database partitioning, table partitioning, and MDC 15
2.1 Database partitioning feature 16
2.1.1 The benefits of using database partitioning feature 16
2.1.2 Usage considerations 19
2.2 Table partitioning 21
2.2.1 Benefits 22
2.2.2 Usage considerations 25
2.3 Multi-dimensional clustering 27
2.3.1 Benefits 27
2.3.2 Usage considerations 28
2.4 Combining usage 29

Chapter 3. Database partitioning 35
3.1 Requirements 36
3.1.1 Supported operating systems and hardware 36
3.1.2 Minimum memory requirements 37
3.2 Planning considerations 37
3.2.1 Deciding on the number of database partitions 38
3.2.2 Logical and physical database partitions 39
3.2.3 Partition groups 39
3.2.4 Distribution maps and distribution keys 40

© Copyright IBM Corp. 2007. All rights reserved.

3.2.5 Table spaces and containers 41
3.2.6 Sizing the tables 42
3.2.7 Buffer pools 43
3.2.8 Catalog partition 43
3.2.9 Coordinator partition 44
3.2.10 Data placement and table join strategies 44
3.3 Implementing DPF on UNIX and Linux 45
3.3.1 Creating instances and databases 46
3.3.2 Defining database partitions 47
3.3.3 Setting up inter-partition communications 48
3.3.4 Creating database 50
3.3.5 Switching partitions 51
3.3.6 Adding database partitions 52
3.3.7 Removing database partitions 53
3.3.8 Creating database partition groups 55
3.3.9 Viewing partition groups 55
3.3.10 Redistributing partition groups 56
3.3.11 Altering database partition groups 58
3.3.12 Dropping a database partition group 61
3.3.13 Implementing buffer pools 61
3.3.14 Implementing table spaces 63
3.3.15 Implementing tables 66
3.4 Implementing DPF on Windows 71
3.4.1 Installing DB2 Enterprise 9 on Windows 71
3.4.2 Working with partitioned databases 79
3.4.3 DB2 Remote Command Service 81
3.5 Administration and management 81
3.5.1 DB2 utilities 81
3.5.2 Monitoring 94
3.5.3 Rebalancer 106
3.6 Using Materialized Query Tables to speed up performance in a DPF environment 119
3.6.1 An overview of MQTs 119
3.6.2 When to consider a MQT 119
3.6.3 When to use the MQT 120
3.6.4 Intra-database replicated tables and partitioning 121
3.7 Best practices 122
3.7.1 Selecting the number of partitions 122
3.7.2 Distribution key selection 123
3.7.3 Collocation 124

Chapter 4. Table partitioning 125
4.1 Planning considerations 126
4.1.1 Roll-in and roll-out strategies 126
4.1.2 Range selection 128
4.1.3 Handling large objects. 128
4.1.4 Indexing partitioned tables 128
4.2 Implementing table partitioning 129
4.2.1 Creating a data partitioned table 130
4.2.2 Adding a new partition. 141
4.2.3 Detaching a partition 143
4.2.4 Re-attaching a partition 145
4.2.5 RANGE option. 146
4.2.6 Handling large objects. 154
4.2.7 Optimal storage configurations for table partitioning 156
4.2.8 Partition elimination. 157
4.3 Administration and management. 159
4.3.1 Utilities. 159
4.3.2 DB2 Explain 161
4.3.3 Locking considerations 166
4.3.4 Troubleshooting 167
4.3.5 Using partitioned tables in your existing database 168
4.3.6 Authorization levels 171
4.4 Best practices 172

Chapter 5. Multi-dimensional clustering 177
5.1 Planning for the use of MDC on a table 178
5.1.1 Verify database configuration 178
5.1.2 Determine query workload 178
5.1.3 Identify dimensions and columns 179
5.1.4 Estimate space requirements 180
5.1.5 Adjust design as needed. 184
5.1.6 DB2 Design Advisor 185
5.2 Implementing MDC on a table. 185
5.3 Administering and monitoring MDC tables 188
5.3.1 Utilities. 188
5.3.2 Monitoring MDC tables 190
5.3.3 Explain 190
5.4 Application considerations for MDC tables 193
5.5 Examples of using MDC 194
5.5.1 Applying MDC to the TPC customer table 194
5.5.2 Utilizing both dimension and row-level indexes 201
5.5.3 Using the Control Center to run DB2 Design Advisor 206
5.5.4 Using MDC to provide roll-out functionality 211

5.5.5 Using MDC on a fact table in a star schema warehouse 213

Chapter 6. Using database partitioning, table partitioning, and MDC together 215
6.1 Database partitioning and MDC 216
6.2 Database partitioning and table partitioning 219
6.2.1 Logical representation 219
6.2.2 Implementing a table using table partitioning and database partitioning 220
6.3 Table partitioning and MDC 228
6.4 Database partitioning, table partitioning, and MDC 231

Appendix A. Configuring DB2 for SSH in a partitioned environment . . . 235
A.1 Setting up public key authentication 236
A.2 Setting up host-based authentication 238
A.2.1 SSH server configuration 238
A.2.2 SSH client configuration 241
A.3 Configuring DB2 to use ssh 241

Appendix B. Additional material 243
Locating the Web material 243
Using the Web material 244
System requirements for downloading the Web material 244
How to use the Web material 244

Related publications 245
IBM Redbooks publications 245
Other publications 245
Online resources 247
How to get Redbooks publications 248
Help from IBM 248

Index 249

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.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

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.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

© Copyright IBM Corp. 2007. All rights reserved.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

Redbooks (logo) ®	DB2 Connect™	Redbooks®
eServer™	DB2®	System p™
pSeries®	IBM®	Tivoli®
zSeries®	Lotus®	1-2-3®
AIX®	POWER™	

The following terms are trademarks of other companies:

Oracle, JD Edwards, PeopleSoft, Siebel, and TopLink are registered trademarks of Oracle Corporation and/or its affiliates.

Solaris, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Excel, Microsoft, SQL Server, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Itanium, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Preface

As organizations strive to do more with less, DB2® Enterprise Server Edition V9 for Linux®, Unix, and Windows® contains innovative features for delivering information on demand and scaling databases to new levels. The table partitioning, newly introduced in DB2 9, and database partitioning feature provide scalability, performance, and flexibility for data store. The multi-dimension clustering table enables rows with similar values across multiple dimensions to be physically clustered together on disk. This clustering allows for efficient I/O and provides performance gain for typical analytical queries.

How are these features and functions different? How do you decide which technique is best for your database needs? Can you use more than one technique concurrently?

This IBM® Redbooks® publication addresses these questions and more. Learn how to set up and administer database partitioning. Explore the table partitioning function and how you can easily add and remove years of data on your warehouse. Analyze your data to discern how multi-dimensional clustering can drastically improve your query performance.

The team that wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, San Jose Center.

Whei-Jen Chen is a Project Leader at the International Technical Support Organization, San Jose Center. She has extensive experience in application development, database design and modeling, and DB2 system administration. Whei-Jen is an IBM Certified Solutions Expert in Database Administration and Application Development as well as an IBM Certified IT Specialist.

Alain Fisher is a software support specialist for IBM Technical Support Services in the United Kingdom. He has 10 years of IT experience on a number of platforms. His experience includes supporting DB2 on Linux, UNIX®, and Windows as well as VMWare and Windows server support. He has also worked on the *Microsoft® SQL Server™ to IBM DB2 UDB Conversion Guide* IBM Redbooks publication.

Aman Lalla is a software development analyst at the IBM Toronto Lab in Canada. He has 10 years of experience with DB2 on the distributed platforms.

© Copyright IBM Corp. 2007. All rights reserved.

He has worked at IBM for 10 years. His areas of expertise include DB2 problem determination. He has previously written an IBM Redbooks publication on DB2 Data Links Manager.

Andrew D McLauchlan is a DB2 Database Administrator and Systems programmer in Australia. He has 29 years of experience in IBM in mainframe hardware maintenance, manufacturing, Oracle® database administration, and DB2 database administration. He holds a degree in Electrical Engineering from Swinburne University, Melbourne, Australia. His areas of expertise include DB2, AIX®, Linux, and Z/OS. He is a member of the IBM Global Services Australia DB2 support group.

Doug Agnew is a DB2 Database Administrator in the United States. He has 34 years of experience in applications development, database design and modeling, and DB2 administration. He holds a degree in Applied Mathematics from the University of North Carolina - Charlotte. His areas of expertise include database administration, data modeling, SQL optimization, and AIX administration. He is an IBM Certified DBA for DB2 9 on Linux, UNIX, and Windows and is a member of the DBA Service Center.

Figure 0-1 Left to right: Alain, Doug, Aman, and Andrew

Acknowledgement

Thanks to the following people for their contributions to this project:

Liping Zhang
Kevin Beck
Paul McInerney
Sherman Lau
Kelly Schlamb
IBM Software Group, Information Management

Emma Jacobs and Sangam Racherla
International Technical Support Organization, San Jose Center

Become a published author

Join us for a two- to six-week residency program! Help write a book dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an e-mail to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

1

Introduction to partitioning technologies

As databases and data warehouses become larger, organizations have to be able to manage an increasing amount of stored data. To handle database growth, Relational Database Management Systems (RDBMS) have to demonstrate scalable performance as additional computing resources are applied.

In this IBM Redbooks publication, we introduce three features available in DB2 9 for Linux, UNIX, and Windows to help manage the growth of your data. In this chapter, we give an overview of the terminologies and concepts that we use throughout this book. The topics we look at are:

- DB2 9 database concepts, including concepts specifically related to the Database Partitioning Feature (DPF), such as partitions, partition groups, distribution keys, distribution maps, and the coordinator partition
- Table partitioning concepts including table partition, partition key, and roll-in and roll-out
- Multi-dimensional clustering (MDC) concepts, such as block, block index, dimension, slice, and cell

Although DPF is a licensed feature while table partitioning and MDC are functions built into the database engine, we refer to all three collectively in this chapter as *partitioning technologies*.

© Copyright IBM Corp. 2007. All rights reserved.

1.1 Databases and partitioning

The Database Partitioning Feature (DPF) is a value-added option available with DB2 Enterprise 9. It extends the capability of DB2 9 into the parallel, multi-partition environment, improving the performance and the scalability of very large databases. This allows very complex queries to execute much faster. DB2 Enterprise 9 with DPF is an ideal solution for managing data warehousing and data mining environments, but it can also be used for large online transaction processing (OLTP) workloads.

1.1.1 Database concepts

In this section, we give an overview of several of the database partitioning and DB2 database concepts that we discuss throughout this book.

Partition

A database partition can be either logical or physical. Logical partitions reside on the same physical server and can take advantage of symmetric multiprocessor (SMP) architecture. Having a partitioned database on a single machine with multiple logical nodes is known as having a *shared-everything* architecture, because the partitions use common memory, CPUs, disk controllers, and disks. Physical partitions consist of two or more physical servers, and the database is partitioned across these servers. This is known as a *shared-nothing* architecture, because each partition has its own memory, CPUs, disk controllers, and disks.

Each partitioned instance is owned by an instance owner and is distinct from other instances. A DB2 instance is created on any one of the machines in the configuration, which becomes the “primary machine.” This primary server is known as the DB2 instance-owning server, because its disk physically stores the instance home directory. This instance home directory is exported to the other servers in the DPF environment. On the other servers, a DB2 instance is separately created: all using the same characteristics, the same instance name, the same password, and a shared instance home directory. Each instance can manage multiple databases; however, a single database can only belong to one instance. It is possible to have multiple DB2 instances on the same group of parallel servers.

Figure 1-1 on page 3 shows an environment with four database partitions across four servers.

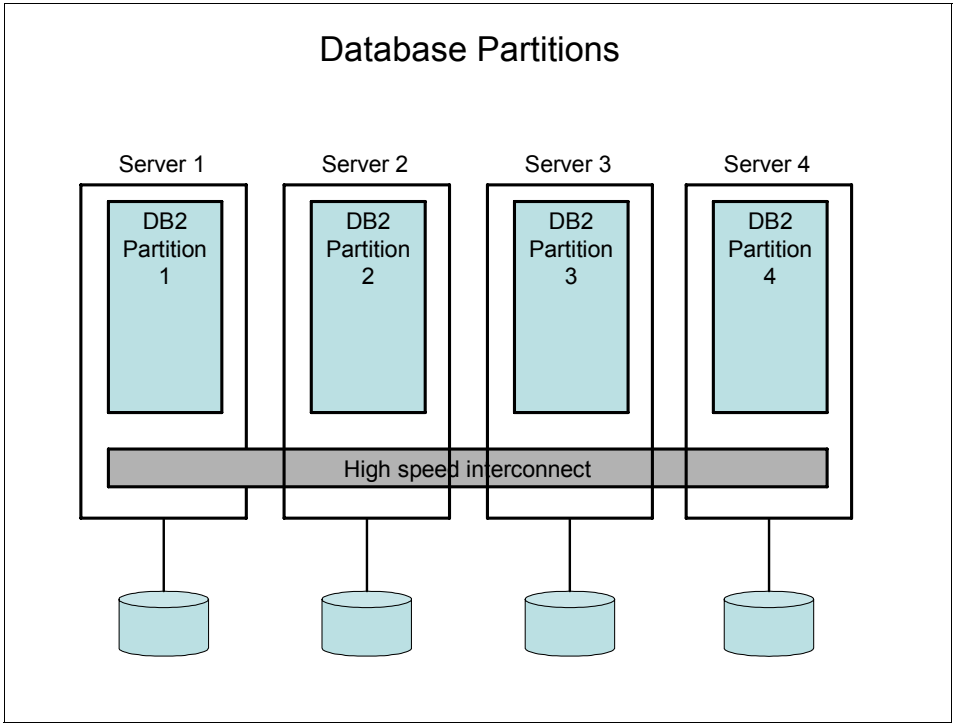


Figure 1-1 Database partitions

Partition groups

A database *partition group* is a logical layer that allows the grouping of one or more partitions. Database partition groups allow you to divide table spaces across all partitions or a subset of partitions. A database partition can belong to more than one partition group. When a table is created in a multi-partition group table space, some of its rows are stored in one partition, and other rows are stored in other partitions. Usually, a single database partition exists on each physical node, and the processors on each system are used by the database manager at each database partition to manage its own part of the total data in the database. Figure 1-2 on page 4 shows four partitions with two partition groups. Partition group A spans three of the partitions and partition group B spans a single partition.

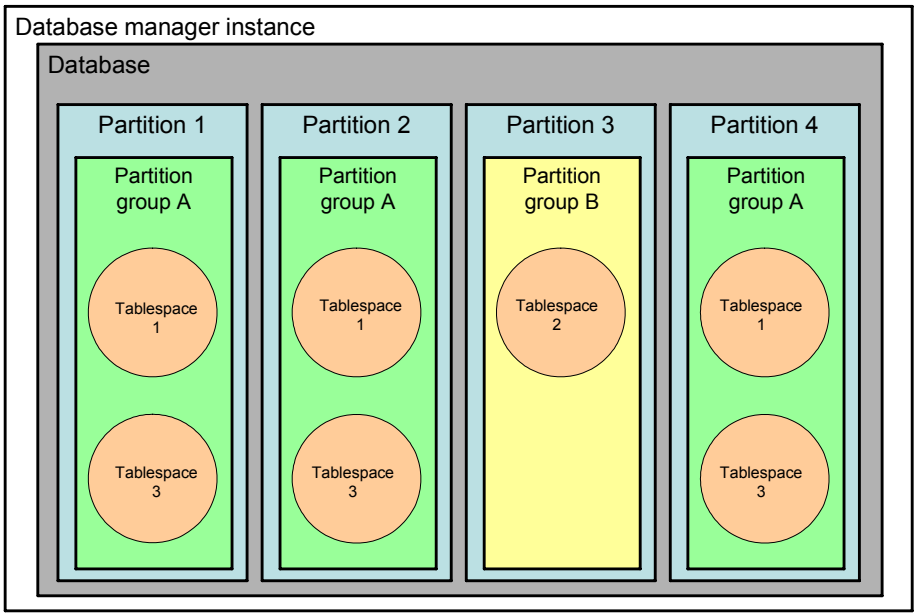


Figure 1-2 Partition groups

Table space

A *table space* is the storage area for your database tables. When you issue the CREATE TABLE command you specify the table space in which the table is stored. DB2 uses two types of table spaces: *System-Managed Space* (SMS) and *Database-Managed Space* (DMS). SMS table spaces are managed by the operating system, and DMS table spaces are managed by DB2.

In general, SMS table spaces are better suited for a database that contains many small tables, because SMS is easier to maintain. With an SMS table space, data, indexes, and large objects are all stored together in the same table space. DMS table spaces are better suited for large databases where performance is a key factor. Containers are added to DMS table spaces in the form of a file or raw device. DMS table spaces support the separation of data, indexes, and large objects. In a partitioned database environment, table spaces belong to one partition group allowing you to specify which partition or partitions the table space spans.

Container

A *container* is the physical storage used by the table space. When you define a table space, you must select the type of table space that you are creating (SMS or DMS) and the type of storage that you are using (the container). The container for an SMS table space is defined as a directory and is not pre-allocated;

therefore, data can be added as long as there is enough space available on the file system. DMS table spaces use either a file or a raw device as a container, and additional containers can be added to support growth. In addition, DMS containers can be resized and extended when necessary.

Buffer pools

The *buffer pool* is an area of memory designed to improve system performance by allowing DB2 to access data from memory rather than from disk. It is effectively a cache of data that is contained on disk which means that DB2 does not have the I/O overhead of reading from disk. Read performance from memory is far better than from disk.

In a partitioned environment, a buffer pool is created in a partition group so that it can span all partitions or single partition, depending on how you have set up your partition groups.

Prefetching

Prefetching is the process by which index and data pages are fetched from disk and passed to the buffer pools in advance of the index and data pages being needed by the application. This can improve I/O performance; however, the most important factors in prefetching performance are the extent size, prefetch size, and placement of containers on disk.

Extent

An *extent* is the basic unit for allocations and is a block of pages that is written to and read from containers by DB2. If you have specified multiple containers for your table space, the extent size can determine how much data is written to the container before the next container is used, in effect, striping the data across multiple containers.

Page

A *page* is a unit of storage within a table space, index space, or virtual memory. Pages can be 4 KB, 8 KB, 16 KB, or 32 KB in size. Table spaces can have a page with any of these sizes. Index space and virtual memory can have a page size of 4 KB. The page size can determine the maximum size of a table that can be stored in the table space.

Distribution maps and distribution keys

When a database partition group is created, a distribution map is generated. A distribution map is an array of 4096 entries, each of which maps to one of the database partitions associated with the database partition group. A distribution key is a column (or group of columns) that is used to determine the partition in which a particular row of data is stored. The distribution key value is hashed to

generate the partition map index value. The hash value ranges from 0 to 4095. The distribution map entry for the index provides the database partition number for the hashed row. DB2 uses a round-robin algorithm to specify the partition numbers for multiple-partition database partition groups. There is only one entry in the array for a database with a single partition, because all the rows of a database reside in one partition database partition group.

Figure 1-3 shows the partition map for database partition group 1 with partitions 1,2, and 4.

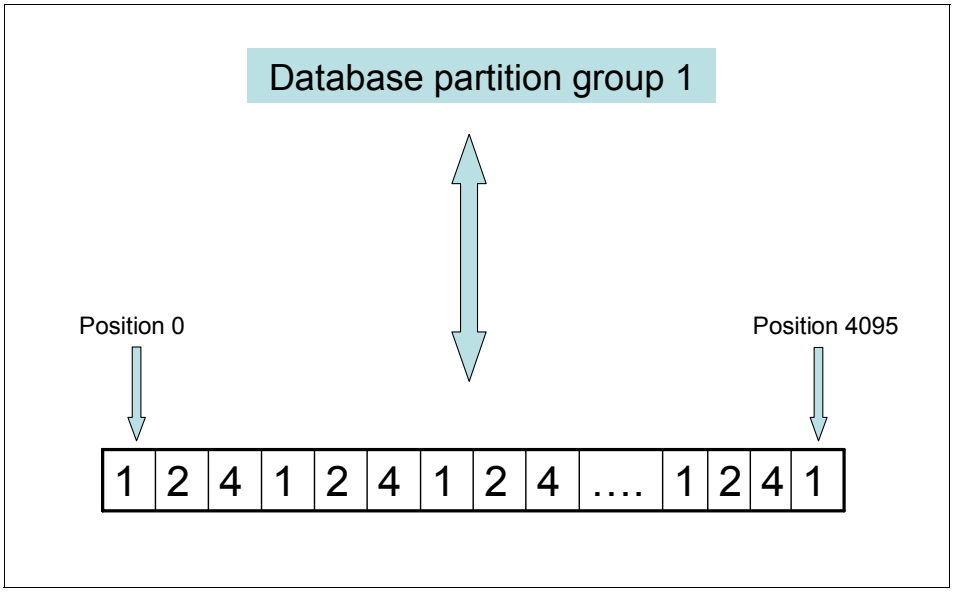


Figure 1-3 Distribution map for partition group 1

Figure 1-4 on page 7 shows how DB2 determines on which partition a given row is stored. Here, the partition key `EMPNO` (value `0000011`) is hashed to a value 3, which is used to index to the partition number 1.

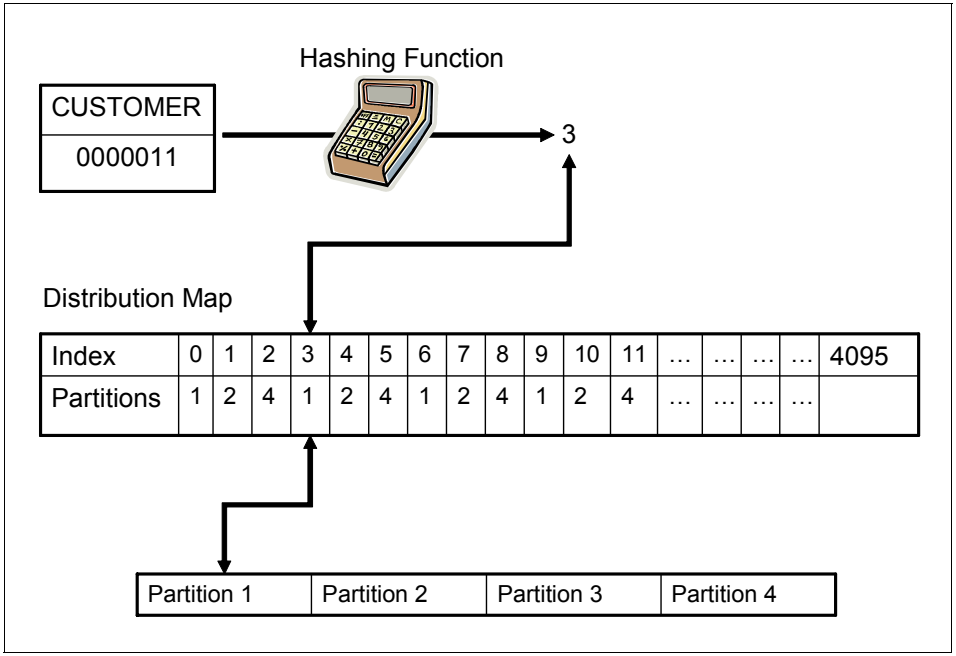


Figure 1-4 DB2 process to identify the partition where the row is to be placed

Coordinator partition

The *coordinator partition* is the database partition to which a client application or a user connects. The coordinator partition compiles the query, collects the results, and then passes the results back to the client, therefore handling the workload of satisfying client requests. In a DPF environment, any partition can be the coordinator node.

Catalog partition

The SYSCATSPACE table space contains all the DB2 system catalog information (metadata) about the database. In a DPF environment, SYSCATSPACE cannot be partitioned, but must reside in one partition. This is known as the *catalog partition*. When creating a database, the partition on which the CREATE DATABASE command is issued becomes the catalog partition for the new database. All access to system tables goes through this database partition.

1.2 Table partitioning

The *table partitioning* feature provides a way of creating a table where each range of the data in the table is stored separately. For example, you can partition a table by month. Now, all data for a specific month is kept together. In fact, internally, the database represents each range as a separate table. It makes managing table data much easier by providing the roll-in and roll-out through table partition attach and detach. It can also potentially boost query performance through data partition elimination.

1.2.1 Concepts

This section introduces some of the table partitioning concepts that we cover in more depth in this book.

Table partitions

Each partition in your partitioned table is stored in a storage object that is also referred to as a data partition or a *range*. Each data partition can be in the same table space, separate table spaces, or a combination of both. For example, you can use date for a range, which allows you to group data into partitions by year, month, or groups of months. Figure 1-5 shows a table partitioned into four data partition ranges with three months in each range.

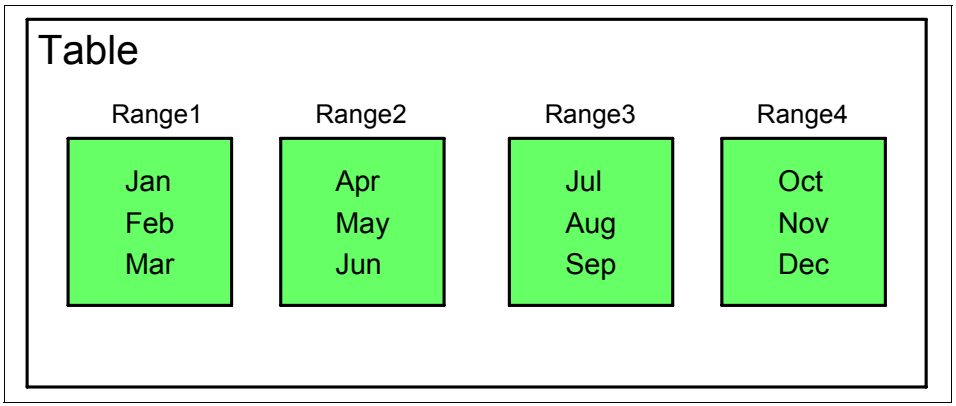


Figure 1-5 Table partition range

Partition keys

The value which determines how a partitioned table is divided is the partition key. A *partition key* is one or more columns in a table that are used to determine to which table partitions the rows belong.

Roll-in and roll-out (attach and detach)

Table partitioning provides the capability of rolling a data partition into and out from a table by using DB2 commands or the GUI tool. By attaching a new partition, you can add an existing table, including data, or an empty partition to a partitioned table as a new partition. Data can be loaded or inserted into the empty partition afterward. The recently added partition must conform to certain criteria for the partitioning range. You also can split off a table partition as an independent table by detaching the partition.

Table partition elimination

Table partition elimination (also known as data partition elimination) refers to the database server's ability to determine, based on the query predicates, that only a subset of the table partitions of a table needs to be accessed to answer a query. Table partition elimination offers particular benefit when running decision support queries against a partitioned table.

1.3 Multi-dimensional clustering

Multi-dimensional clustering (MDC) provides an elegant method for flexible, continuous, and automatic clustering of data along multiple dimensions. MDC is primarily intended for data warehousing and large database environments, but can also be used in OLTP environments. MDC enables a table to be physically clustered on more than one key (or dimension) simultaneously.

1.3.1 Concepts

This section introduces several of the multi-dimensional clustering concepts that we discuss in further detail in this book.

Block

A *block* is the smallest allocation unit of an MDC. A block is a consecutive set of pages on the disk. The block size determines how many pages are in a block. A block is equivalent to an extent.

Block index

The structure of a block index is almost identical to a regular index. The major difference is that the leaf pages of a regular index are made up of pointers to rows, while the leaf pages of a block index contain pointers to extents. Because each entry of a block index points to an extent, whereas the entry in a regular index points to a row, a block index is much smaller than a regular index, but it still points to the same number of rows. In determining access paths for queries,

the optimizer can use block indexes in the same way that it uses regular indexes. You can use AND and OR for the block indexes with other block indexes. You can also use AND and OR for block indexes with regular indexes. Block indexes can also be used to perform reverse scans. Because the block index contains pointers to extents, not rows, a block index cannot enforce the uniqueness of rows. For that, a regular index on the column is necessary. Figure 1-6 shows a regular table with a clustering index and Figure 1-7 on page 11 shows an MDC.

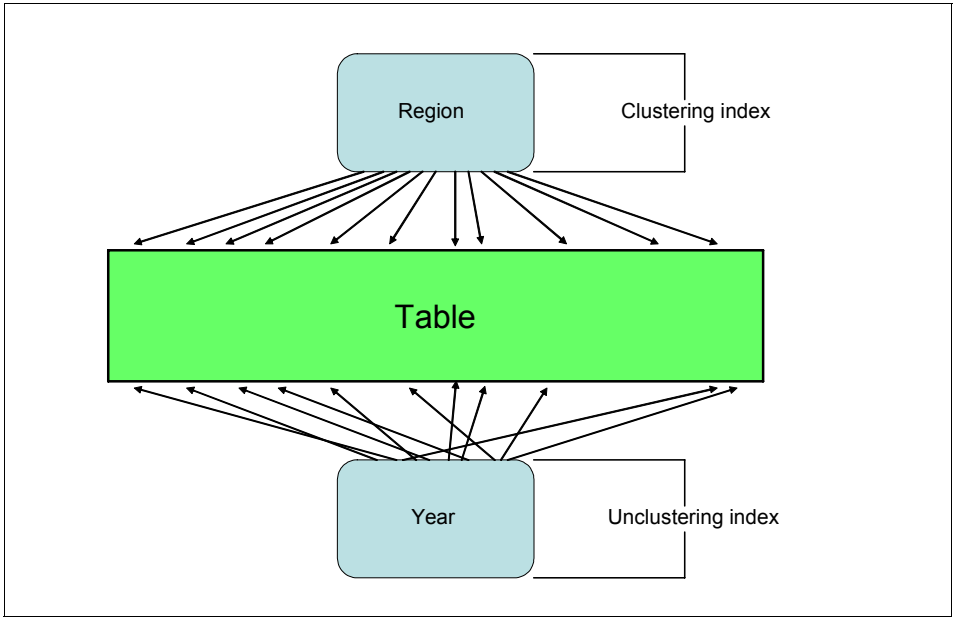


Figure 1-6 A regular table with a clustering index

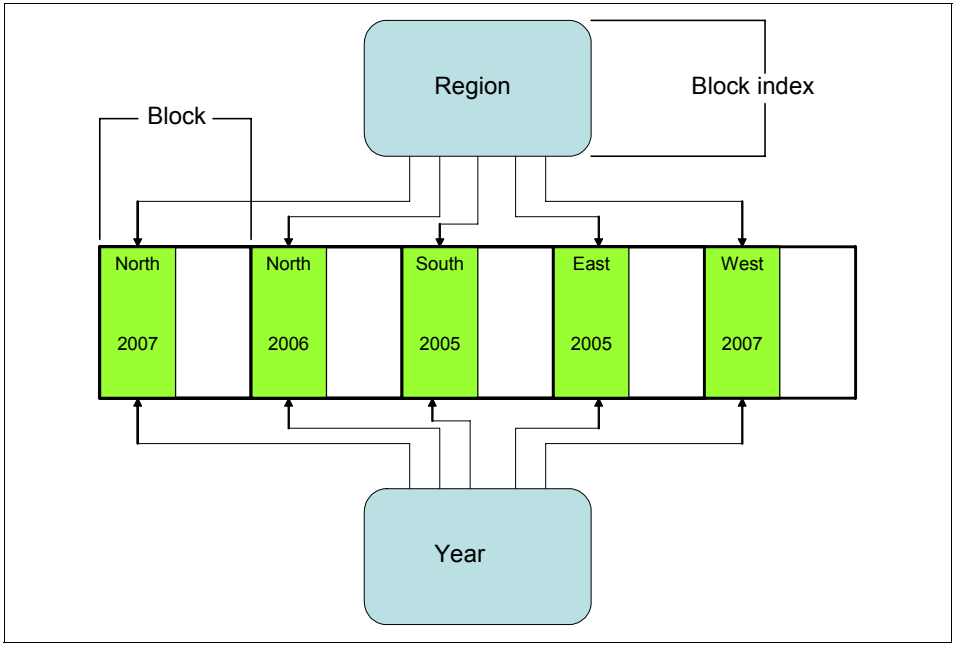


Figure 1-7 Multi-dimensional clustering table

Dimension

A *dimension* is an axis along which data is organized in an MDC table. A dimension is an ordered set of one or more columns, which you can think of as one of the clustering keys of the table. Figure 1-8 on page 12 illustrates a dimension.

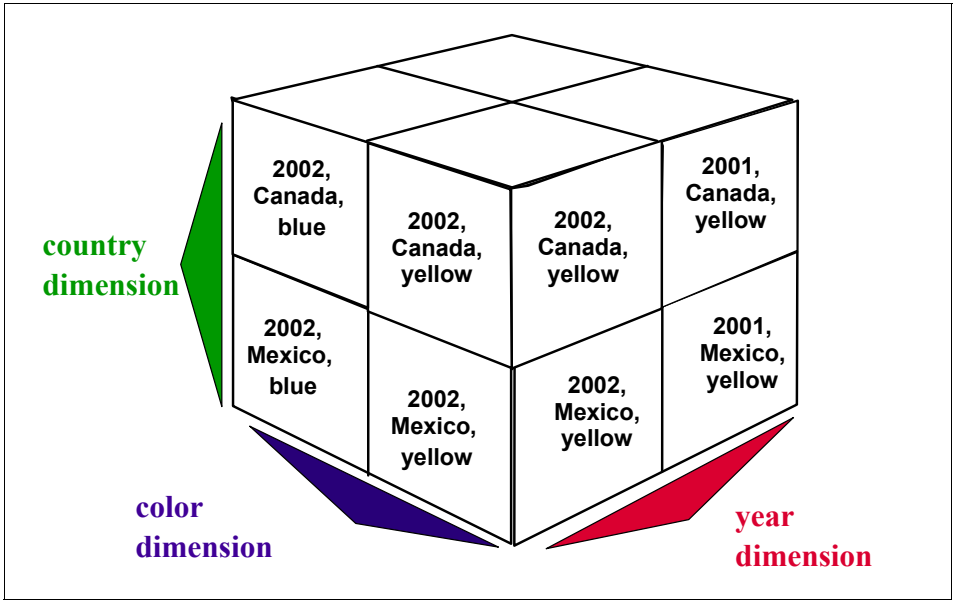


Figure 1-8 Dimensions for country, color, and year

Slice

A *slice* is the portion of the table that contains all the rows that have a specific value for one of the dimensions. Figure 1-9 on page 13 shows the `Canada` slice for the country dimension.

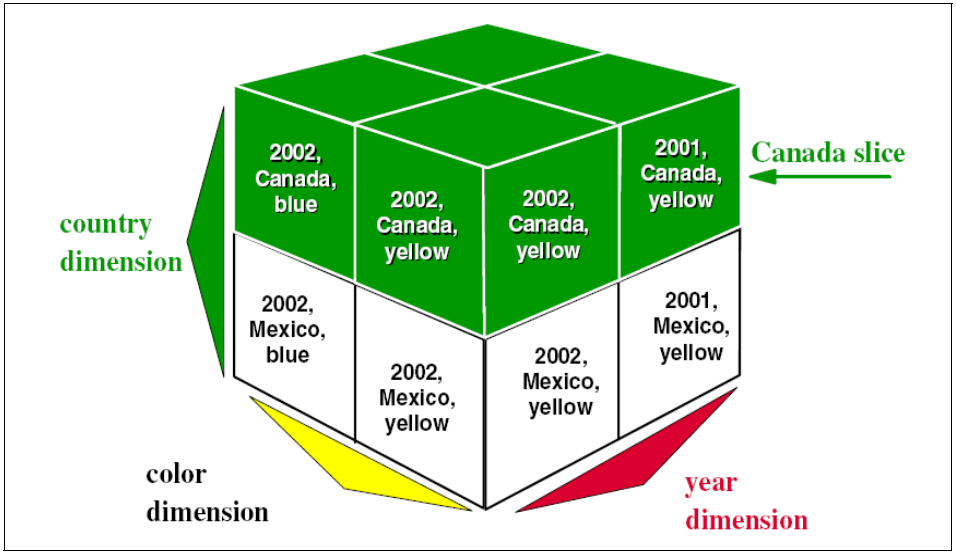


Figure 1-9 The slice for country Canada

Cell

A *cell* is the portion of the table that contains rows having the same unique set of dimension values. The cell is the intersection of the slices from each of the dimensions. The size of each cell is at least one block (extent) and possibly more. Figure 1-10 on page 14 shows the cell for year `2002`, country `Canada`, and color `yellow`.

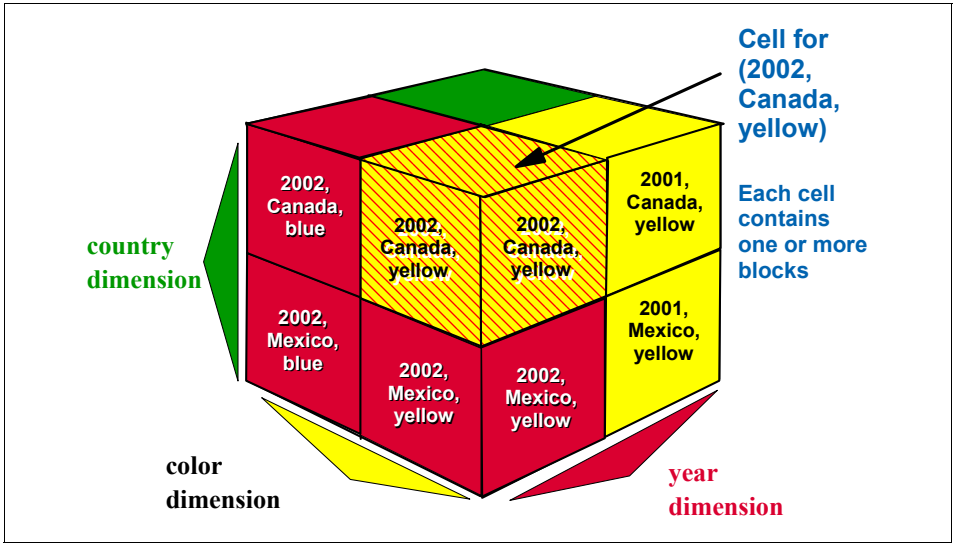


Figure 1-10 The cell for dimension values (2002, Canada, and yellow)

You can read more information about the topics that we discuss in this chapter in the DB2 9 Information Center at:

http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp

2

Benefits and considerations of database partitioning, table partitioning, and MDC

In this chapter, we discuss the benefits and considerations of the database partitioning, the table partitioning, and MDC. We provide information about the advantages of using these features and functions to help you gain an understanding of the added complexity when using them. Used appropriately, database partitioning, table partitioning, and MDC enhance your use of DB2.

© Copyright IBM Corp. 2007. All rights reserved.

2.1 Database partitioning feature

This section highlights several of the benefits and usage considerations that you need to take into account when evaluating the use of the Database Partitioning Feature (DPF) in DB2 V9.

2.1.1 The benefits of using database partitioning feature

In this section, we discuss several of the key benefits of using database partitioning.

Support for various hardware configurations

DB2 9 with DPF enabled supports symmetric multiprocessor (SMP), massively parallel processing (MPP), and clustered hardware configurations.

SMP is a hardware concept in which a single physical machine has multiple CPUs. From a DB2 with DPF-enabled point of view, this is a shared everything architecture, because each database partition shares common CPUs, memory, and disks as shown in Figure 2-1.

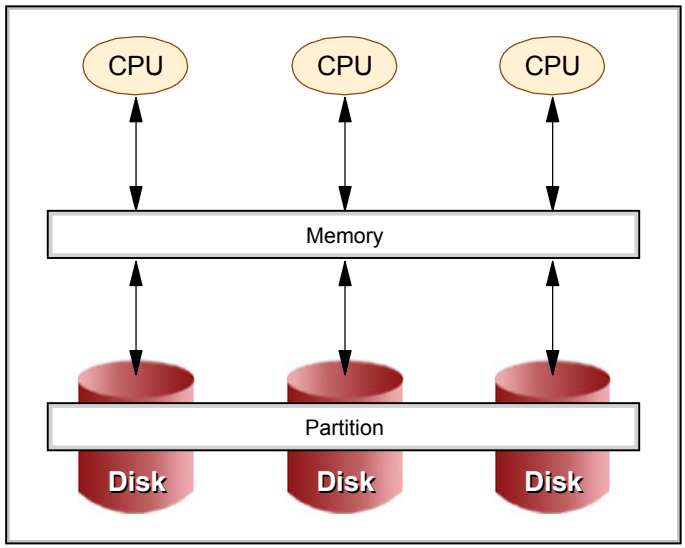


Figure 2-1 An SMP configuration

MPP is a hardware concept in which a set of servers is connected by a high-speed network and a shared nothing application, such as DB2, is used as shown in Figure 2-2 on page 17.

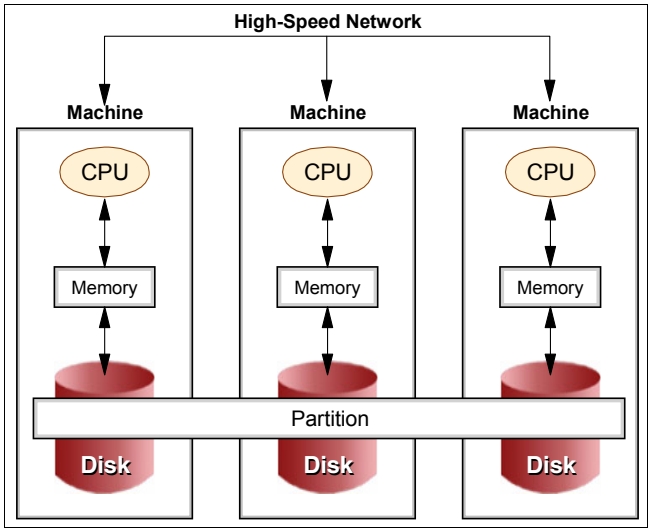


Figure 2-2 An MPP configuration

A *clustered configuration* is simply an interconnection of SMP machines with the database on a shared disk as shown in Figure 2-3.

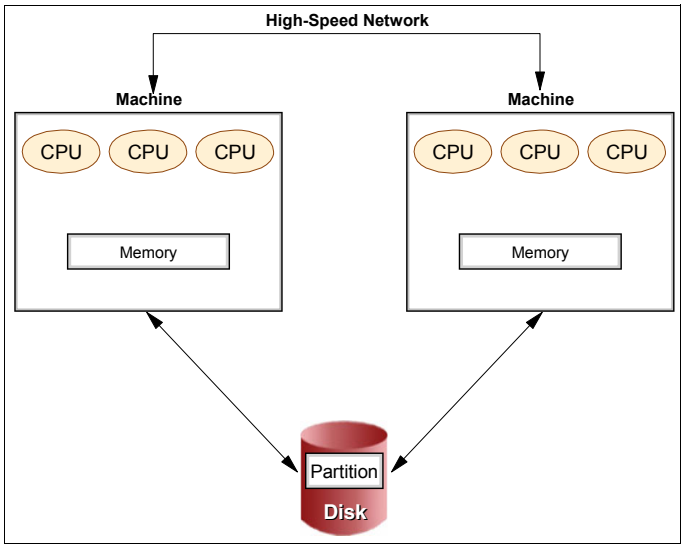


Figure 2-3 A clustered configuration

Shared nothing architecture

DB2 with DPF enabled can be configured with multiple database partitions and utilize a shared nothing architecture. This implies that no CPUs, data, or memory are shared between database partitions, which provides good scalability.

Scalability

Scalability refers to the ability of a database to grow while maintaining operating and response time characteristics. As a database grows, scalability in a DPF environment can be achieved by either scaling up or scaling out. *Scaling up* refers to growing by adding more resources to a physical machine. *Scaling out* refers to growing by adding additional physical machines.

Scale up a DPF-enabled database by adding more physical resources, such as CPUs, disks, and memory, to the physical machine.

Scale out a DPF-enabled database by adding a physical machine to the configuration. This implies adding another physical database partition on the physical machine. The database can then be spread across these new physical partitions to take advantage of the new configuration.

When planning a DPF environment, consider whether to add logical or physical database partitions to provide scalability.

Logical database partitions differ from physical database partitions in that logical database partitions share CPUs, memory, and the disk in an SMP configuration. A physical database partition on a physical machine does not share CPU, disks, or memory with other physical database partitions in an MPP configuration.

Consider adding logical database partitions if the machine has multiple CPUs that can be shared by the logical database partitions. Ensure that there is sufficient memory and disk space for each logical database partition.

Consider adding physical database partitions to a database environment when a physical machine does not have the capacity to have a logical database partition on the same machine. *Capacity* refers to the number of users and applications that can concurrently access the database. Capacity is usually determined by the amount of CPU, memory, and disk available on the machine.

Parallelism

DB2 supports query, utility, and input/output (I/O) parallelism. Parallelism in a database can dramatically improve performance. DB2 supports intrapartition and interpartition parallelism.

Intrapartition parallelism is the ability to subdivide a single database operation, such as a query, into subparts, many or all of which can be executed in parallel within a single database partition.

Interpartition parallelism is the ability to subdivide a single database operation into smaller parts that can be executed in parallel across multiple database partitions.

With the DPF feature enabled, DB2 can take advantage of both intrapartition and interpartition parallelism:

- Query parallelism

  DB2 provides two types of query parallelism: interquery and intraquery parallelism. *Interquery parallelism* refers to the ability to execute multiple queries from different applications at the same time. *Intraquery parallelism* is the ability to execute multiple parts of a single query by using interpartition parallelism, intrapartition parallelism, or both.

- Utility parallelism

  In a DPF-enabled environment, DB2 utilities can take advantage of both intrapartition and interpartition parallelism. This enables the utility to run in parallel on all database partitions.

- I/O parallelism

  DB2 can exploit I/O parallelism by simultaneously reading from and writing to devices. During the planning phases of database design, table space container placement, as well as the number of containers for the table space, must be considered. Having multiple containers for a single table space can dramatically improve I/O performance due to I/O parallelism.

Query optimization

DB2 9's cost-based query optimizer is DPF-aware. This implies that the query optimizer uses the system configuration, the database configuration, and the statistics stored in the system catalogs to generate the most efficient access plan to satisfy SQL queries across multiple database partitions.

2.1.2 Usage considerations

In this section, we highlight several of the key usage considerations that you need to take into account when planning to implement multiple database partitions.

Database administration

In general, administering a multi-partition database is similar to a single partition database with a few additional considerations.

Disk

When laying out the physical database on disk, the workload of the database must be taken into consideration. In general, workloads can be divided into two broad categories: online transaction processing (OLTP) and Decision Support Systems (DSS). For OLTP-type workloads, the database needs to support a great deal of concurrent activity, such as inserts, updates, and deletes. Disks must be laid out to support this concurrent activity. For DSS-type workloads, the database must support a small number of large and complex queries as compared to transactions reading records and many sequential I/O operations.

In a DPF-enabled database environment, each database partition has a set of transaction logs. Our recommendation is to place the transaction log files on separate physical disks from the data. This is especially true for an OLTP environment where transaction logging can be intensive. Each database partition's log files must be managed independently of other database partitions.

Memory

The main memory allocation in a DB2 database is for the database buffer pool. In a DPF-enabled database with multiple database partitions, the buffer pools are allocated per partition. In a DSS environment, most of the memory must be allocated for buffer pools and sorting. Decisions have to be made as to the number of buffer pools to create. A buffer pool is required for each page size that is used in the database. Having properly tuned buffer pools is important for the performance of the database, because the buffer pools are primarily a cache for I/O operations.

Database design

When designing a DPF-enabled database, the database administrator needs to:

- Understand database partitioning
- Select the number of database partitions
- Define the database partition groups
- Define the table space within these partition groups
- Understand distribution maps
- Select distribution keys

Recovery

In a DPF-enabled environment, each database partition must be backed up for recovery purposes. This includes archiving the transaction log files for each partition if archival logging is used. The granularity of the backups determines the recovery time. Decisions must be made about the frequency of the backups, the

type of backups (full offline or online, incremental backups, table space backups, flash copies, and so forth). In large DSS environments, high availability solutions might need to be considered as well.

Database configuration

In a DPF-enabled environment, each database partition has a database configuration file. Take this into consideration when you make changes on a database partition that might need to be made on other database partitions as well. For example, the LOGPATH needs to be set on each database partition.

System resources

System resources can be broadly divided into CPU, memory, disk, and network resources. When planning to implement a database with multiple partitions on the same physical machine, we recommend that the number of database partitions does not exceed the number of CPUs on the machine. For example, if the server has four CPUs, there must not be more than four database partitions on the server. This is to ensure that each database partition has sufficient CPU resources. Having multiple CPUs per database partition is beneficial whereas having fewer CPUs per database partition can lead to CPU bottlenecks.

Each database partition requires its own memory resources. By default on DB2 9, shared memory is used for the communication between logical database partitions on the same physical machine. This is to improve communications performance between logical database partitions. For multiple database partitions, sufficient physical memory must be available to prevent memory bottlenecks, such as the overcommitment of memory.

Designing the physical layout of the database on disk is critical to the I/O throughput of the database. In general, table space containers need to be placed across as many physical disks as possible to gain the benefits of parallelism. Each database partition has its own transaction logs, and these transaction logs need to be placed on disks that are separate from the data to prevent I/O bottlenecks. In a DPF configuration, data is not shared between database partitions.

2.2 Table partitioning

This section deals with the benefits and the considerations of using table partitioning.

2.2.1 Benefits

Several of the benefits of table partitioning are:

- The ability to roll-in and roll-out data easily
- Improved query performance
- Improved performance when archiving or deleting data in larger tables
- Larger table support

Roll-in and roll-out

Roll-in and roll-out are efficient ways of including or removing data in a table. You use the ALTER TABLE statement with the ATTACH PARTITION or DETACH PARTITION clause to add or remove a partition to the table. You can reuse the removed data partition for new data or store the removed data partition as an archive table. This is particularly useful in a Business Intelligence environment.

Query performance

Query performance is improved because the optimizer is aware of data partitions and therefore scans only partitions relevant to the query. This also presumes that the partition key is appropriate for the SQL. This is represented in Figure 2-4.

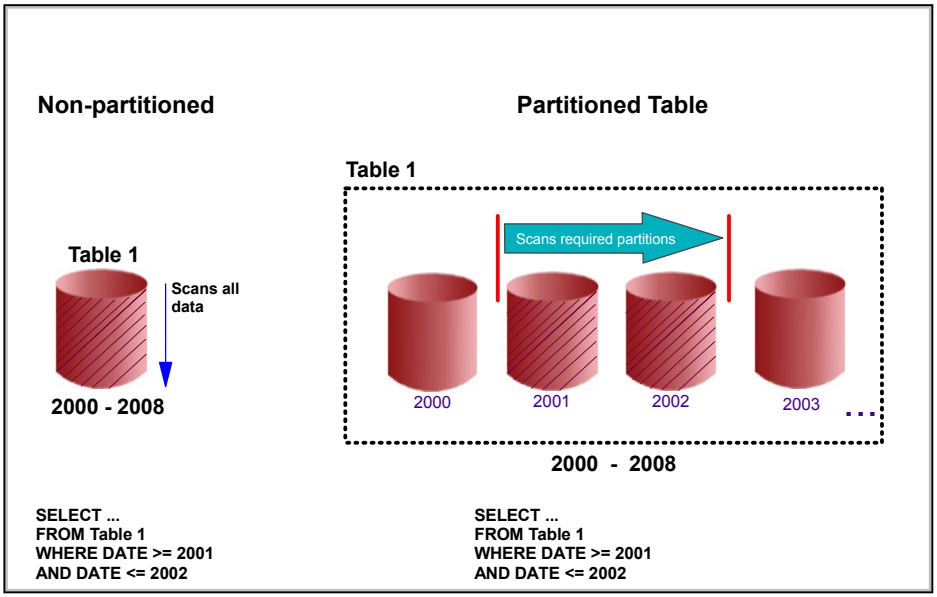


Figure 2-4 Performance improvement through scan limiting

Flexible index placement

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

- Improved performance for dropping an index and online index creation.
- 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 might be appropriate to ensure better space utilization.
- Reduced I/O contention providing more efficient concurrent access to the index data for the table.
- When individual indexes are dropped, space immediately becomes available to the system without the need for an index reorganization.
- If you choose to perform index reorganization, an individual index can be reorganized.

Note: Both System-Managed Space (SMS) table spaces and Database-Managed Space (DMS) table spaces support the use of indexes in a different location than the table.

Archive and delete

By using the ALTER TABLE statement with the DETACH PARTITION clause, the removed data partition can be reused for new data or stored as an archive table. Detaching a partition to remove large amounts of data also avoids logging, whereas the traditional delete of several million rows produces significant logging overhead.

Larger table support

Without table partitioning, there is a limit on the size of storage objects and hence table size. The use of table partitioning allows the table to be split into multiple storage objects. The maximum table size can effectively be unlimited, because the maximum number of data partitions is 32767.

Query simplification

In the past, if the table had grown to its limit, a view was required across the original and subsequent tables to allow a full view of all the data. Table partitioning negates the need for a UNION ALL view of multiple tables due to the limits of standard tables. The use of the view means that the view has to be dropped or created each time that a new table was added or deleted.

A depiction of a view over multiple tables to increase table size is in Figure 2-5 on page 24. If you modify the view (VIEW1), the view must be dropped and recreated with tables added or removed as required. During this period of

modification, the view is unavailable. After you have recreated the view, authorizations must again be made to match the previous authorizations.

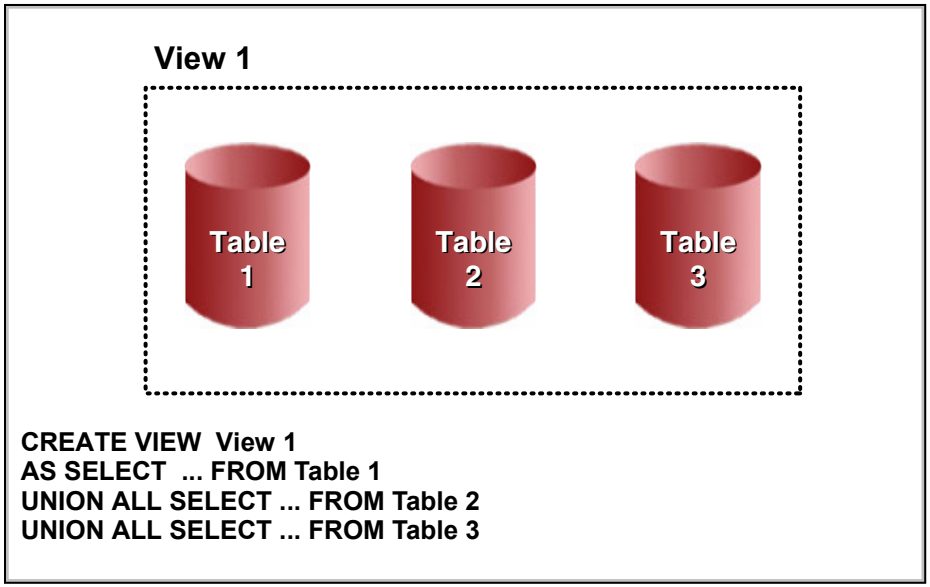


Figure 2-5 View over multiple tables to increase table size

You can achieve the same result by using table partitioning as shown in Figure 2-6 on page 25. In the case of the partitioned table, a partition can be detached, attached, or added with minimal impact.

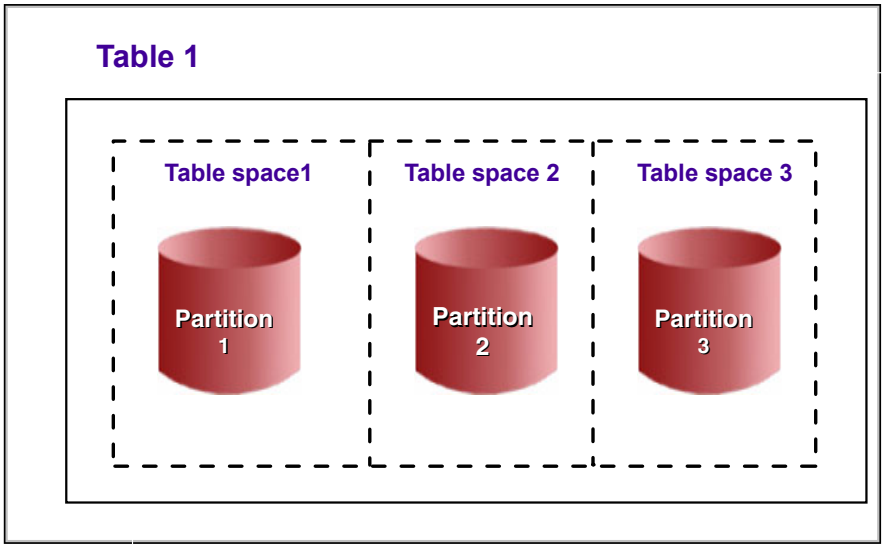


Figure 2-6 Large table achieved by using table partitioning

2.2.2 Usage considerations

Here, we discuss several of the subjects that you need to consider when using table partitioning.

Locking

When attaching or detaching a partition, applications need to acquire appropriate locks on the partitioned table and the associated table. The application needs to obtain the table lock first and then acquire data partition locks as dictated by the data accessed. Access methods and isolation levels might require locking of data partitions that are not in the result set. When 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 reacquiring 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 particular 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 an 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 can be accessed, a non-intent lock might be chosen at the table level to prevent deadlocks between data partition locks from concurrent transactions.

Administration

There is additional administration required above that of a single non-partitioned table. The range partitioning clauses and the table space options for table partitions, indexes, or large object placement must be considered when creating a table. When attaching a data partition, you must manage the content of the incoming table so that it complies with the range specifications. If data exists in the incoming table that does not fit within the range boundary, you receive an error message to that effect.

System resources

A partitioned table usually has more than one partition. We advise you always to carefully consider the table space usage and file system structures in this situation. We discuss optimizing the layout of table spaces and the underlying disk structure in Chapter 4, “Table partitioning” on page 125.

Replication

In DB2 9.1, replication does not yet support source tables that are partitioned by range (using the PARTITION BY clause of the CREATE TABLE statement).

Data type and table type

The following data type or special table types are not supported by the table partitioning feature at the time of writing this book:

- Partitioned tables cannot contain XML columns.
- DETACH from the parent table in an enforced referential integrity (RI) relationship is not allowed.
- Certain special types of tables cannot be table-partitioned:
  - Staging tables for materialized query tables (MQTs)
  - Defined global temporary tables
  - Exception tables

2.3 Multi-dimensional clustering

In this section, we present the benefits and usage considerations for using multi-dimensional clustering (MDC) on a table.

2.3.1 Benefits

The primary benefit of MDC is query performance. In addition, the benefits include:

- Reduced logging
- Reduced table maintenance
- Reduced application dependence on clustering index structure

Query performance

MDC tables can show significant performance improvements for queries using the dimension columns in the WHERE, GROUP BY, and ORDER BY clauses. The improvement is influenced by a number of factors:

- Dimension columns are usually not row-level index columns.

  Because dimension columns must be columns with low cardinality, they are not the columns that are normally placed at the start of a row-level index. They might not even be included in row-level indexes. This means that the predicates on these columns are index-SARGable (compared after the index record had been read but before the data row is read) at best, and probably only data-SARGable (compared after the data is read). With the dimension block indexes, however, the dimension columns are resolved as index start-stop keys, which means the blocks are selected before any index-SARGable or data-SARGable processing occurs.

- Dimension block indexes are smaller than their equivalent row-level indexes.

  Row-level index entries contain a key value and the list of individual rows that have that key value. Dimension block index entries contain a key value and a list of blocks (extents) where all the rows in the extent contain that key value.

- Block index scans provide clustered data access and block prefetching.

  Because the data is clustered by the dimension column values, all the rows in the fetched blocks pass the predicates for the dimension columns. This means that, compared to a non-clustered table, a higher percentage of the rows in the fetched blocks are selected. Fewer blocks have to be read to find all qualifying rows. This means less I/O, which translates into improved query performance.

Reduced logging

There is no update of the dimension block indexes on a table insert unless there is no space available in a block with the necessary dimension column values. This results in fewer log entries than when the table has row-level indexes, which are updated on every insert.

When performing a mass delete on an MDC table by specifying a WHERE clause containing only dimension columns (resulting in the deletion of entire cells), less data is logged than on a non-MDC table because only a few bytes in the blocks of each deleted cell are updated. Logging individual deleted rows in this case is unnecessary.

Reduced table maintenance

Because blocks in MDC tables contain only rows with identical values for the dimension columns, every row inserted is guaranteed to be clustered appropriately. This means that, unlike a table with a clustering index, the clustering effect remains as strong when the table has been heavily updated as when it was first loaded. Table reorganizations to recluster the data are totally unnecessary.

Reduced application dependence on index structures

Because the dimensions in an MDC table are indexed separately, the application does not need to reference them in a specific order for optimum usage. When the table has a multi-column clustering index, in comparison, the application needs to reference the columns at the beginning of the index to use it optimally.

2.3.2 Usage considerations

If you think your database application can benefit from using MDC on some of the tables, be sure that you consider:

- Is the application primarily for OLTP (transaction processing) or DSS (data warehouse/decision support)?

  Although MDC is suitable for both environments, the DSS environment probably realizes greater performance gains, because the volume of data processed is generally larger and queries are expected to be long-running. MDC is particularly effective in that environment, because it improves data filtering and can reduce the amount of I/O that is required to satisfy queries.

  The OLTP environment, alternatively, normally is characterized by intensive update activity and rapid query response. MDC can improve query response times, but it can impact update activity when the value in a dimension column is changed. In that case, the row must be removed from the cell in which it

currently resides and placed in a suitable cell. This might involve creating a new cell and updating the dimension indexes.

- Is sufficient database space available?

  An MDC table takes more space than the same table without MDC. In addition, new (dimension) indexes are created.

- Is adequate design time available?

  To design an MDC table properly, you must analyze the SQL that is used to query the table. Improper design leads to tables with large percentages of wasted space, resulting in much larger space requirements.

2.4 Combining usage

Database partitioning, table partitioning, and MDC are compatible and complementary to each other. When combining two or all three partitioning technologies in your database design, you can take advantage of each technology. You can create a data-partitioned table in a Database Partitioning Feature (DPF) environment to benefit from the parallelism. MQTs work well on table-partitioned tables. In conjunction with table partitioning, DB2 has made improvements to SET INTEGRITY so that maintenance of MQTs can be accomplished without interfering with applications accessing the base tables concurrently. Utilities, such as LOAD and REORG, work on partitioned tables. The considerations of each technology discussed in this chapter apply when combining the usage of these DB2 partitioning technologies.

One monolithic table

When the data is stored in a single large table, many business intelligence style queries read most of the table as shown in Figure 2-7 on page 30. In this illustration, the query is looking for “blue triangles.” Even if an appropriate index is created in the table, each I/O can pick up many unrelated records that happen to reside on the same page. Because DPF is not used, only one CPU is utilized for much of the processing done by a query.

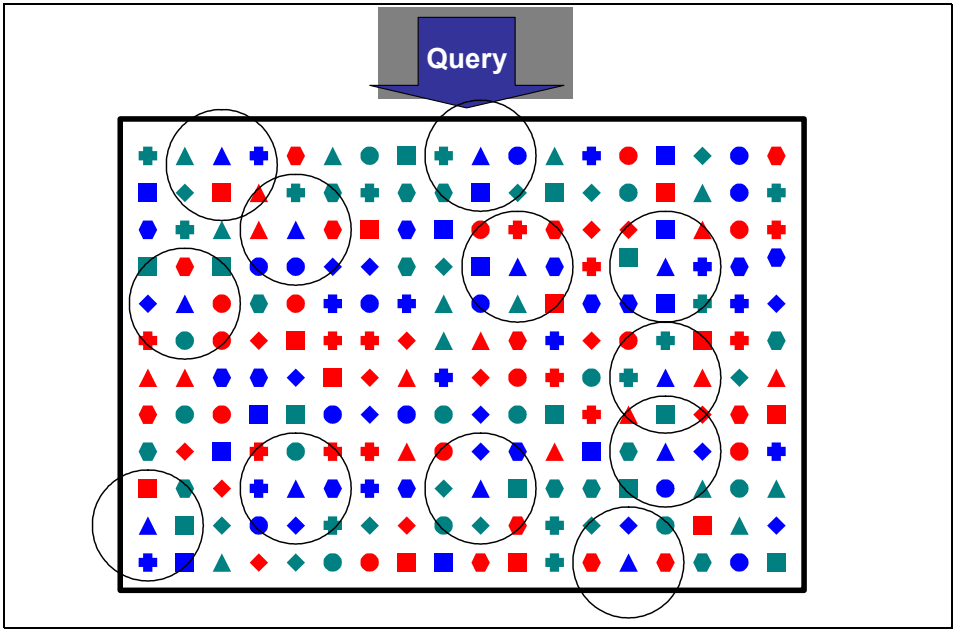


Figure 2-7 Data stored in one monolithic table

Using the database partitioning feature

With DPF, data is distributed by hash into different database partitions. You can take the advantage of the query parallelism provided by DB2 DPF. This work now can be attacked on all nodes in parallel. See Figure 2-8 on page 31.

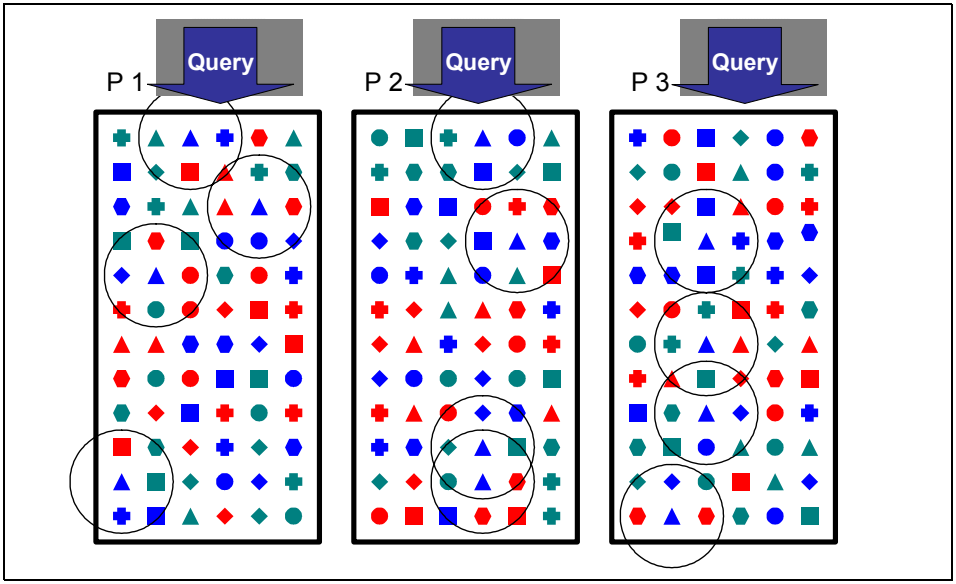


Figure 2-8 Data distributed by hash

Using database and table partitioning

With table partitioning, all data in the same user-defined range is consolidated in the same data partition. The database can read just the appropriate partition. Figure 2-9 on page 32 illustrates only three ranges, but real tables can have dozens or hundreds of data partitions. Table partitioning yields a significant savings in I/O for many business intelligence style queries.

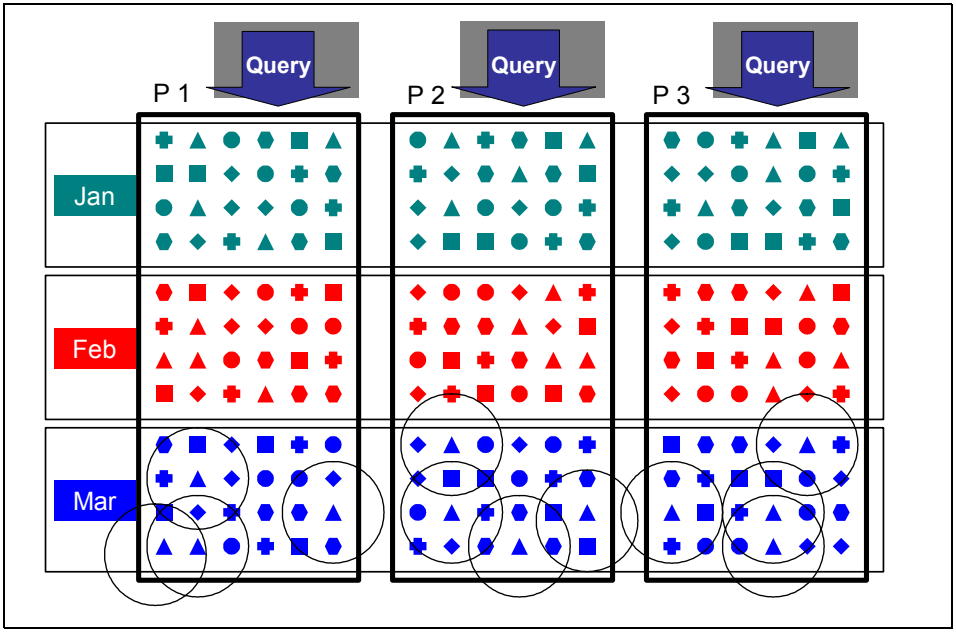


Figure 2-9 Using both database and table partitioning

Using database partitioning, table partitioning, and MDC

You can use MDC to further cluster the data by additional attributes. By combining these technologies, even less I/O is performed to retrieve the records of interest. Plus, table partitioning enables easy roll-in and roll-out of data. See Figure 2-10 on page 33.

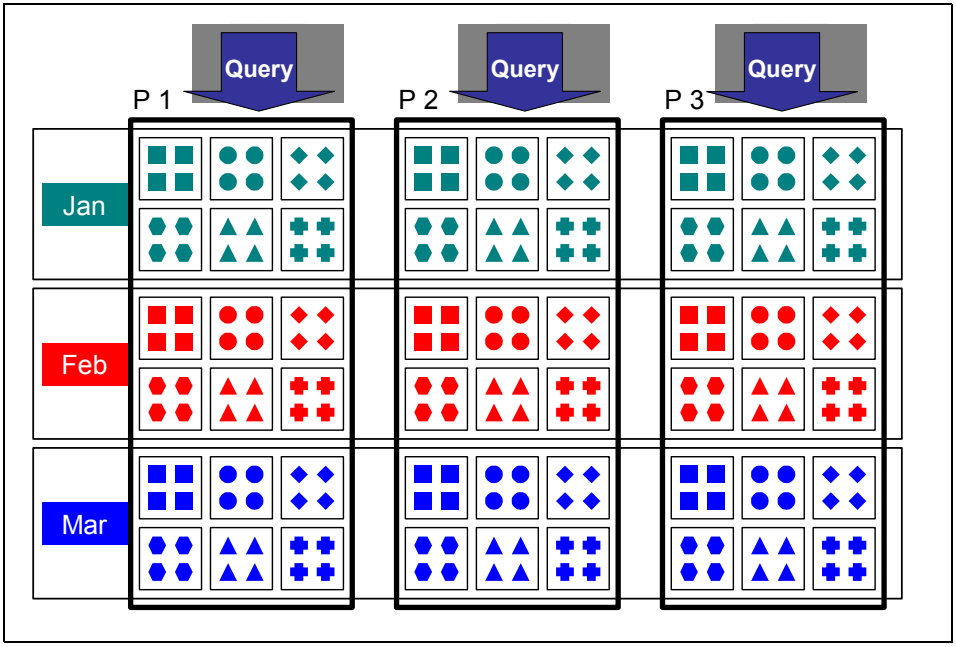


Figure 2-10 Using database partitioning, table partitioning, and MDC

3

Database partitioning

In this chapter, we describe the Database Partitioning Feature (DPF) of DB2. We begin by discussing the hardware, software, and memory requirements. We follow with a discussion about planning and implementing a database partitioning environment. Finally, we discuss the administrative aspects of a partitioned environment and best practices.

In this chapter, we discuss the following subjects:

- Requirements
- Planning considerations
- Implementing DPF on UNIX and Linux
- Implementing DPF on Windows
- Administration and management
- Using Materialized Query Tables to speed up performance in a DPF environment
- Best practices

© Copyright IBM Corp. 2007. All rights reserved.

3.1 Requirements

This section describes the operating systems and hardware supported by DB2 9 together with the minimum memory requirements for running DB2.

3.1.1 Supported operating systems and hardware

In Table 3-1, we summarize the supported operating systems and hardware requirements to install and run DB2 9.

Table 3-1 Supported operating systems and hardware platforms for DB2 9

Operating System	Operating System Version	Hardware
AIX	5.2 with 64-bit AIX kernel	eServer™ pSeries® IBM System p™
	5.3 with 64-bit AIX kernel	
HP-UX	11iv2 (11.23.0505)	PA-RISC (PA-8x00)-based HP 9000, Series 700, and Series 800 systems Itanium®-based HP Integrity Series systems
Linux	Novell Open Enterprise Server 9 with 2.6.5 kernel	x86
	Red Hat Enterprise Linux (RHEL) 4 with 2.6.9 kernel	x86 x86_64 IA64 PPC 64 (POWER™) s390x (zSeries®)
	SUSE Linux Enterprise Server (SLES) 9 with 2.6.9 kernel	
Solaris™	9 with 64-bit kernel and patches 111711-12 and 111712-12	UltraSPARC
	10	UltraSPARC or x86-64 (EM64T or AMD64)

Operating System	Operating System Version	Hardware
Windows	XP Professional Edition with Service Pack (SP) 1 or later	All systems based on Intel® or AMD processors that are capable of running the supported Windows operating systems (32-bit, x64, and Itanium-based systems)
	XP Professional x64 Edition with SP1 or later	
	2003 Standard Edition (32-bit and 64-bit) with SP1 or later	
	2003 Enterprise Edition (32-bit and 64-bit) with SP1 or later	
	2003 Datacenter Edition (32-bit and 64-bit) with SP1 or later	
	Vista supported with DB2 9.1 Fix pack 2 or later	

3.1.2 Minimum memory requirements

At a minimum, a DB2 database system requires 256 MB of RAM. For a system running just DB2 and the DB2 GUI tools, a minimum of 512 MB of RAM is required. However, one GB of RAM is recommended for improved performance. These requirements do not include any additional memory requirements for other software that is running on the system.

When determining memory requirements, be aware of the following:

- DB2 products that run on HP-UX Version 11i V2 (B.11.23) for Itanium-based systems require 512 MB of RAM at a minimum.
- For DB2 client support, these memory requirements are for a base of five concurrent client connections. You need an additional 16 MB of RAM per five client connections.
- Memory requirements are affected by the size and complexity of your database system, as well as by the extent of database activity and the number of clients accessing your system.

3.2 Planning considerations

In this section, we discuss several of the important decisions that you must make when planning a partitioned database environment. Many of the decisions made

during the planning stages require an understanding of the machine configuration in the environment, storage that is available, raw data sizes, and the expected workload.

3.2.1 Deciding on the number of database partitions

DB2 supports up to a maximum of 1000 partitions. When trying to determine the number of database partitions to use, consider:

- The total amount of data under each partition

  This must not be too high to improve the parallelism of partition-wide operations, such as backups. Many factors are involved when deciding how much data to have per database partition, such as the complexity of the queries, response time expectations, and other application characteristics.

- The amount of CPU, memory, and disk available

  A minimum of one CPU needs to be available per database partition. If the environment is expected to support a large number of concurrent queries, consider having more CPUs and memory per database partition. In general, consider more CPUs and memory if you plan to run 20-30 concurrent queries.

- Workload type

  The type of workload of your database has to be considered when you define the physical database layout. For example, if the database is part of a Web application, typically, its behavior is similar to online transaction processing (OLTP). There is a great deal of concurrent activity, such as large numbers of active sessions, simple SQL statements to process, and single row updates. Also, there is a requirement to perform these tasks in seconds. If the database is part of a Decision Support System (DSS), there are small numbers of large and complex query statements, fewer transactions updating records as compared to transactions reading records, and many sequential I/O operations.

- The amount of transaction logging that takes place per partition

  Transaction logs are created per partition. Log management increases as the number of partitions increase, for example, archiving and retrieving log files. For performance reasons, consider placing transaction log files on your fastest disks that are independent of your data.

- The largest table size under each partition

  This must not be too high. There is an architectural limit per partition of 64 GB for 4 K page size; 128 GB for 8 K page size; 256 GB for 16 K page size; and 512 GB for 32 K page size. If you determine that the architectural limits might be a problem, consider adding more database partitions to the environment during the planning phase.

- The time required for recovery

  In general, the more data under each database partition, the longer the recovery times. In a partitioned database environment, it is possible to recover some database partitions independently of others.

- The time available for batch jobs and maintenance

  Queries and utilities in DB2 can take advantage of parallelism. In general, as the number of database partitions increase, so does the utility parallelism.

3.2.2 Logical and physical database partitions

When planning to implement a multi-partition database system, consider the types of database partitions that you use. In general, the decision between logical partitions and physical partitions is based on the hardware that is available or will be available for the database. If you plan to use a large symmetric multiprocessor (SMP) machine, consider implementing a shared everything configuration and using logical database partitions. If multiple physical machines are available, such as a massively parallel processing (MPP) configuration, consider using a shared nothing architecture and physical partitions. If you plan to use multiple physical SMP machines, consider using both physical and logical partitions.

3.2.3 Partition groups

A *partition group* is a collection of one or more database partitions. In a single partition environment, a partition group is not of concern. Table spaces for a database are created in a partition group. By default, there are three partition groups created when a database is created:

- IBMCATGROUP
- IBMDEFAULTGROUP
- IBMTEMPGROUP

You create the IBMCATGROUP partition group when you issue the CREATE DATABASE statement and it contains the system catalog table space (SYSCATSPACE) together with the system catalog tables. The IBMCATGROUP partition group spans only the catalog partition and cannot be altered or dropped.

The IBMDEFAULTGROUP partition group spans all the database partitions, and it is the default group for the CREATE TABLESPACE statement. By default, the USERSPACE1 table space is created in the IBMDEFAULTGROUP partition group. The IBMDEFAULTGROUP partition group is a system object and cannot be altered or dropped.

The IBMTEMPGROUP partition group spans all the database partitions and contains the TEMPSPACE1 table space. The IBMTEMPGROUP partition group is a system object and cannot be altered or dropped.

In general, when designing the database, place large tables in partition groups that span all or most of the partitions in order to take advantage of the underlying hardware and the distribution of the data on each partition. Place small tables in a partition group that spans one database partition, except when you want to take advantage of collocation with a larger table, which requires both tables to be in the same partition group.

Consider separating OLPT-type workload data into partition groups that span one partition. Place DSS-type workload data across multiple partitions.

3.2.4 Distribution maps and distribution keys

Each partition group has a *distribution map* that is created when the partition group is created. The distribution map is used by the database manager to find the data.

A *distribution key* is a column (or a group of columns) from a table that the database manager uses to determine how data is distributed between the partitions, that is, the database partition where the data is stored. The distribution key is specified with the CREATE TABLE statement.

When a row is inserted into a table, DB2 uses a hashing algorithm on the distribution key to determine on which partition to store the row of data by using the distribution map.

Choosing a good distribution key is important for the even distribution of data across partitions and maximizing the use of collocated joins. Collocation between joining tables means that you must have the matching rows of the joining tables on the same database partition. This avoids the database manager from shipping rows between partitions.

Selecting distribution keys

To choose a good distribution key candidate, consider the following rules:

- Frequently joined columns.
- Columns that have a high proportion of different values to ensure an even distribution of rows across all database partitions in the database partition group.
- Integer columns are more efficient than character columns, which are more efficient than decimal.

- Equijoin columns. An *equijoin* is a join operation in which the join condition has the form expression = expression.
- Use the smallest number of columns possible.

Having an inappropriate distribution key can cause uneven data distribution. This can cause the database manager to ship large amounts of rows between partitions.

3.2.5 Table spaces and containers

Table spaces are created in partition groups. Tables are created in table spaces. If a table needs to span multiple partitions, it must be created in a table space that spans multiple database partitions. In general, small tables must be created in table spaces that span one partition. A table with approximately 100 000 rows or about 20 MB in size is considered to be small. Medium-sized tables must be placed in table spaces that do not span too many partitions.

Containers store the data from table spaces onto disk. If there are many available disks on the system, consider creating each table space container on a single disk; otherwise, spread the containers across all the disks. In general, the more disks per table space, the better the performance.

If a Database-Managed Space (DMS) table space is defined by using device containers, it does not use the operating system file system caching. File containers for DMS table spaces use file system caching. On AIX, in either case, it can be beneficial to tune down the file system caching to avoid double buffering of data in the file system cache and the buffer pool.

The general rules to optimize table scans are:

- Without RAID:
  - Extent size needs to be any of the available sizes: 32, 64, 128 and so on. It is best to be in at least the 128K or 256K range.
  - Prefetch size must be the number of containers (that is, the number of disks) multiplied by the extent size.
- With RAID:
  - Extent size must be a multiple of the RAID stripe size.
  - Prefetch size needs to be the number of disk multiplied by the RAID stripe size and a multiple of the extent size.

    This can be limiting if the number of disks is a prime number. In this case, your only choices are extent size = RAID stripe size or extent size = (number of disks x RAID stripe size), which can easily be too big.

3.2.6 Sizing the tables

Knowing the size of your tables is important to determine how many database partitions you might need. For example, for a table space with a 4 KB page size, the maximum size of a table is 64 GB per database partition. To allow for larger tables, more database partitions can be added.

We explain how to size tables by using an example of a large partitioned database environment. We used a two p690 (32-way/128 GB memory) environment to calculate the table sizes.

Figure 3-1 details the estimated table sizes that we used in our example to illustrate sizing tables.

Table Name	Row Count	Row length	Total Size (GB)	Size / SMP (GB)	Size / DP (GB)
lineitem	5999989709	146.3	817.5	408.7	6.38
orders	1500000000	128	178.8	89.4	1.39
customer	150000000	195	27.2	13.6	0.21
part	200000000	163.8	30.5	15.25	0.24
supplier	10000000	170.7	1.6	0.8	0.01
partsupp	800000000	163.8	122	61	0.95
nation	25	n/a	0		
region	5	n/a	0		
TOTAL	8659989739		1177.6	588.8	9.2

Figure 3-1 Estimated table sizes

The descriptions of the columns in Figure 3-1are:

- Row count: The number of rows in the table
- Row length: The sum of the byte counts of all columns in the table
- Total size: Total size = Row count x Row length
- Size/SMP: Total size/number of physical machines

  In this example, we used two p690 machines.
- Size/data partition (DP): (Size/ SMP)/number of database partitions per machine

Based on the table size information, we used 32 database partitions on each machine for a total of 64 database partitions.

3.2.7 Buffer pools

In a DPF-enabled environment, buffer pools are created per partition. By default, if no partition groups are specified during the CREATE BUFFERPOOL statement, the buffer pool is created on all database partitions. Buffer pools can be created using partition groups to identify on which partitions to create the buffer pools.

When creating table spaces by using the CREATE TABLESPACE statement, you can specify a buffer pool to be used by the tables in the table space. The database partition group must be defined for the buffer pool. If no buffer pool is specified during the CREATE TABLESPACE statement, the default IBMDEFAULTBP is used.

Buffer pool allocation can have a profound effect on the performance of the database. When assigning buffer pools to table spaces, consider placing table spaces that have sequential I/O patterns in separate buffer pools from those table spaces that have random I/O patterns.

Consider the following general guidelines for creating buffer pools:

- For DSS workloads, create a buffer pool dedicated to the temporary table spaces. I/O in this environment tends to be mostly sequential and benefits DSS workloads.
- Create a buffer pool for table spaces in which the I/O is largely random, that is, there is not much prefetching or asynchronous I/O activity.
- Create a dedicated buffer pool for OLTP workloads where the tables are frequently read.
- Consider a buffer pool for indexes of tables that are frequently accessed via indexes.
- System catalog tables usually have random I/O patterns, therefore, consider creating a dedicated buffer pool for this table space.
- A large table that is frequently scanned completely can potentially flush out the buffer pool every time that it is scanned. Consider using a separate table space for this table and assign a small buffer pool to it, which saves memory that you can use for other buffer pools.

3.2.8 Catalog partition

The *catalog partition* in a DPF environment is the partition where the system catalog tables are created. The catalog partition is created on the partition where the CREATE DATABASE statement is issued. The catalog partition resides in

IBMCATGROUP partition group, which contains the SYSCATSPACE table space. The IBMCATGROUP partition group spans one partition.

By default, the SYSCATSPACE table space is created as a system-managed (SMS) table space. It can be created as a DMS table space when the CREATE DATABASE statement is issued.

Note: The IBMCATGROUP partition group is a system-created object and cannot be altered by using the ALTER DATABASE PARTITION GROUP statement.

3.2.9 Coordinator partition

The *coordinator partition* is the partition where the application runs. If the application is remote, the coordinator partition is the partition where the application connects. The coordinator partition can be any partition in the database. The application can use the SET CURRENT CLIENT statement to specify to which database partition to connect.

3.2.10 Data placement and table join strategies

In a DPF-enabled environment, consider where data is placed on each database partition. The location of the data on each partition influences the table join strategy that the DB2 optimizer considers and uses. The following join strategies are used by DB2 optimizer in a partitioned database environment:

- Collocated joins

  In this join strategy, there is no data movement between partitions. Always try to use the maximum number of collocated joins so that the database manager can avoid shipping data between partitions. A collocated join occurs if two collocated tables are joined using all of the columns in the partitioning key. Tables are collocated when they satisfy the following requirements:

  – The tables must be in same database partition group.

  – The distribution keys from both tables must have the same number of columns.

- Directed joins

  In this join strategy, data from a table is directed to another partition to complete the join. This is the next best join strategy that the query optimizer considers if collocated joins cannot be used.

- Broadcast joins

  In this join strategy, each row of one table is broadcast to all partitions of the other table to complete the join. If collocated or directed joins cannot be used by the query optimizer, broadcast joins are considered by the query optimizer.

3.3 Implementing DPF on UNIX and Linux

In this section, we describe the steps in our examples of implementing a DPF system in our test environment. Our test environment consisted of a 4-way System p (p690) running AIX 5.3 and DB2 9.1 with fix pack 2. The design of the database partitions, partition groups, and table spaces of our test system is graphically illustrated in Figure 3-2.

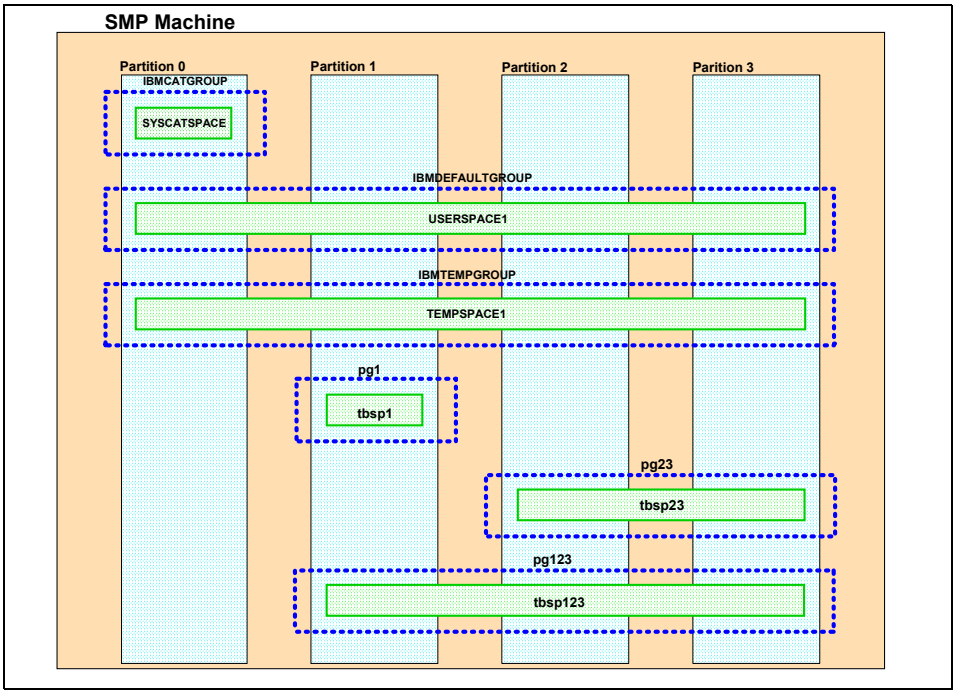


Figure 3-2 Illustration of our test system

We have a single SMP machine with four logical database partitions. We create six database partition groups:

- IBMCATGROUP, IBMDEFAULTGROUP, and IBMTEMPGROUP are DB2-created partition groups when the database is created.
- The partition groups PG1, PG23, and PG123 are user-defined.

We also have six table spaces:

- Table spaces SYSCATSPACE, USERSPACE1, and TEMPSPACE1 are created by default by DB2 when the CREATE DATABASE statement is issued.
- The table spaces TBSP1, TBSP23, and TBSP123 are user-defined.

These are the steps to set up a partitioned database environment:

1. Install DB2 with the database partitioning feature enabled on all the physical servers.
2. Create an instance.
3. Define database partitions by updating the db2nodes.cfg.
4. Set up inter-partition communications.
5. Create a database on the catalog partition.
6. Create database partition groups.
7. Create table spaces.
8. Create database objects: tables, indexes, and so forth.
9. Load data.

We discuss these steps and the related topics in details in the following sections.

3.3.1 Creating instances and databases

The first step in setting up a partitioned database environment is to install DB2. You must install DB2 on each physical machine that is going to participate in the database partition environment.

In our test database environment, we installed DB2 using the db2_install utility, which is only available on UNIX and Linux. On Windows, DB2 can be installed using the db2setup utility. Obtain more information about the installation methods for DB2 on UNIX, Linux, and Windows at the DB2 Information Center:

http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.udb.uprun.doc/doc/c0008711.htm

After we successfully installed DB2 on our AIX machine, we proceeded to create a DB2 instance. In order to create an instance, we need a user ID for the instance owner. In our case, we created a user called `db2inst1`, which has a home directory called /home/db2inst1/.

Note: The /home/<*instance owner*> directory must be a shared directory when setting up multiple physical machines. Each physical machine shares this directory by mounting this directory using Network File System (NFS).

In our test environment, we created an instance called `db2inst1` by using the **`db2icrt`** command. You can obtain more information about the **`db2icrt`** command at the Web site:

http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.udb.admin.doc/doc/r0002057.htm

In our test environment, we created the `db2inst1` instance as the root user as shown in Example 3-1.

Example 3-1 Creating an instance by using db2icrt

```
/opt/IBM/db2/V9.1/instance/db2icrt -u db2inst1 db2inst1
DBI1070I Program db2icrt completed successfully.
```

3.3.2 Defining database partitions

The DB2 nodes configuration file (db2nodes.cfg) is used to define the database partitions that participate in a DB2 instance. Before a partitioned database is created, the partitions need to be defined in the db2nodes.cfg file. There is one db2nodes.cfg file for each instance. The db2nodes.cfg file is located in the instance owner's home directory. It contains one entry for each server participating in the instance. Ensure that the instance is stopped before updating the file using a text editor. The db2nodes.cfg file can be updated to add or remove a server in the DB2 instance. All databases that are created in an instance use the db2nodes.cfg to specify on which hosts and servers the database is created. The format of the db2nodes.cfg file on UNIX is:

```
node number, hostname, logical port, netname, resourcesetname
```

In our test environment, we decided to use four logical partitions. The db2nodes.cfg is updated as shown in Example 3-2.

Example 3-2 The db2nodes.cfg file in our test environment

```
0 Clyde 0
1 Clyde 1
2 Clyde 2
3 Clyde 3
```

In our example in Example 3-2 on page 47, we have defined four partitions on the same host, which indicates that these are logical database partitions. The netname and resourcesetname are optional. If we wanted two physical partitions, the db2nodes.cfg file looks similar to Example 3-3.

Example 3-3 The db2nodes.cfg file for two physical database partitions

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

Note: After you create a database, do not update the db2nodes.cfg file manually to add or remove servers. Manually updating this file might result in unexpected errors. Use the ADD/DROP DBPARTITIONNUM statement instead.

You can obtain more information about the contents of the db2nodes.cfg file at:

http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.udb.uprun.doc/doc/r0006351.htm

3.3.3 Setting up inter-partition communications

In a DPF-enabled environment, each database partition needs to communicate with all other database partitions. DB2 uses the DB2 Fast Communication Manager to communicate with other database partitions. To support this communication, you need to update or create the operating system services file, hosts file, and .rhost files with the new configuration information.

Services file

During the instance creation, a number of ports, which are equal to the number of logical nodes that the instance is capable of supporting, are reserved in the services file.

The ports that are reserved in the services file are used by the DB2 Fast Communication Manager. The reserved ports have 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.

When you install the instance-owning database partition server on the primary computer, DB2 sets up communication ports. The default range is four ports. The DB2 Setup wizard attempts to reserve an identical port range when database partition servers are installed on participating computers. If the port has been used, the Setup wizard uses the first available port range after 60000. However, if the port numbers in the participating servers do not match the ports in the Instance-owning server, you receive communication errors when you start DB2.

On UNIX, the services file is located in the /etc. directory. On Windows, the services file is located in <*drive*>:\WINDOWS\system32\drivers\etc directory. An extract of the services file for our instance in our test environment is shown in Example 3-4.

Example 3-4 Extract of a services file

```
DB2_db2inst1    60004/tcp
DB2_db2inst1_1  60005/tcp
DB2_db2inst1_2  60006/tcp
DB2_db2inst1_END        60009/tcp
```

Hosts file

Another important consideration is the correct definition of the hosts file. The hosts file is located in the same directory as the services file. In the hosts file, you must have an entry defining the IP address, server name, and domain name (in a Windows environment) for each participating server. This ensures the correct configuration of the additional physical partitions at setup time. Example 3-5 shows a sample hosts file with two entries: one entry corresponds to the partition-owning server and the other entry corresponds to an additional physical partition.

Example 3-5 Sample hosts file

```
# Internet Address      Hostname        # Comments
# 192.9.200.1           net0sample      # ethernet name/address
# 128.100.0.1           token0sample    # token ring name/address
# 10.2.0.2              x25sample       # x.25 name/address

127.0.0.1               loopback localhost      # loopback (lo0)
9.43.86.56     Clyde.itsosj.sanjose.ibm.com
...
```

.rhosts files

In a DPF environment, each database partition server must have the authority to perform remote commands on all the other database partition servers participating in an instance. You grant this authority by creating and updating the

.rhosts file in the instance owner's home directory. The .rhosts file has the following format:

```
hostname   instance_owner_user_name
```

In our test environment we created our .rhosts file as shown in Example 3-6.

Example 3-6 Contents of our .rhosts file

```
Clyde db2inst1
```

3.3.4 Creating database

After you create and configure the instance for multiple database partitions, we can proceed to create a database. In a DPF environment, you need to issue the CREATE DATABASE command on the partition where the catalog partition is located, because this determines the catalog partition.

We create our test database, `TESTDB`, as shown in Example 3-7.

Example 3-7 Creating a database

```
db2 CREATE DATABASE testdb
```

You can obtain more information about the CREATE DATABASE statement at:

http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.udb.admin.doc/doc/r0001941.htm

Identifying the catalog partition

In a DPF environment, statements, such as the ROLLFORWARD DATABASE statement, must be issued from the catalog partition. You can locate the catalog partition by using the LIST DATABASE DIRECTORY command as shown in Example 3-8.

Example 3-8 Determining the catalog partition

```
db2 LIST DATABASE DIRECTORY

Database 1 entry:

 Database alias                       = TESTDB
 Database name                        = TESTDB
 Local database directory             = /home/db2inst1
 Database release level               = b.00
 Comment                              =
```

```
 Directory entry type                  = Indirect
 Catalog database partition number     = 0
 Alternate server hostname             =
 Alternate server port number          =
```

The catalog partition can be created on any database partition and does not have to be database partition 0.

3.3.5 Switching partitions

In a DPF environment, you need to execute several commands on specific database partitions. For example, the LIST TABLESPACES statement is partition-specific. In order to switch to a particular partition, follow the sequence of steps shown in Example 3-9.

Example 3-9 Switching partitions

```
export DB2NODE=2
db2 terminate
DB20000I  The TERMINATE command completed successfully.
```

Example 3-9 demonstrates switching to partition 2 in our test environment. To verify that we are on partition 2, we use the VALUES statement with the CURRENT DBPARTITIONNUM special register as shown in Example 3-10.

Example 3-10 Determining the current database partition

```
db2 connect to testdb

db2 "values (current dbpartitionnum)"

1
-----------
          2

  1 record(s) selected.
```

The DB2NODE environment variable is used to specify the coordinator database partition. Before it can be effective, you must issue the TERMINATE statement. The TERMINATE statement terminates the attachment of the DB2 back-end process.

You can obtain more information about the back-end process and its relationship to the Command Line Processor (CLP) at:

http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.udb.admin.doc/doc/r0010412.htm

When issuing DB2 commands to the database using the CLP, be careful to ensure that you issue the command on the intended database partition.

3.3.6 Adding database partitions

Database partitions can be added to a DPF environment by using the following commands:

- **db2start... ADD DBPARTITIONNUM**
- ADD DBPARTITIONNUM

The **db2start... ADD DBPARTITIONNUM** command starts DB2 and adds a new partition to an existing instance and database. For example, to add another partition to our test environment by using **db2start**, we issued the **db2start** command as in Example 3-11.

Example 3-11 Adding a database partition using the db2start command

```
db2start DBPARTITIONNUM 4 ADD DBPARTITIONNUM HOSTNAME Clyde PORT 4

04/04/2007 09:00:16     4   0   SQL6075W  The Start Database Manager
operation successfully added the node. The node is not active until all
nodes are stopped and started again.

SQL6075W  The Start Database Manager operation successfully added the
node. The node is not active until all nodes are stopped and started
again.
```

Upon successful completion of the command, the new db2nodes.cfg is as shown in Example 3-12.

Example 3-12 The db2nodes.cfg after successfully adding a database partition

```
0 Clyde 0
1 Clyde 1
2 Clyde 2
3 Clyde 3
4 Clyde 4
```

By default, the TEMPSPACE1 table space is created on the new database partition as shown in Example 3-13.

Example 3-13 The LIST TABLESPACES statement

```
db2 LIST TABLESPACES

           Tablespaces for Current Database

 Tablespace ID                        = 1
 Name                                 = TEMPSPACE1
 Type                                 = System managed space
 Contents                             = System Temporary data
 State                                = 0x0000
   Detailed explanation:
     Normal

DB21011I  In a partitioned database server environment, only the table
spaces
on the current node are listed.
```

Alternatively, we can add a new partition as shown in Example 3-14.

Example 3-14 Using the ADD DBPARTITIONNUM command to add a partition

```
db2 ADD DBPARTITIONNUM
```

The ADD DBPARTITIONNUM command creates a new empty partition in each database for the instance. In order for the ADD DBPARTITIONNUM statement to succeed, the db2nodes.cfg must have the new partition defined and the statement must be executed on the newly defined partition. Unlike the **db2start DBPARTITIONNUM** command, which adds an entry to the db2nodes.cfg file, the ADD DBPARTITIONNUM statement assumes that the partition is already defined in the db2nodes.cfg.

3.3.7 Removing database partitions

A database partition can only be removed if the database partition is not in use, that is, there is no data on the partition. Prior to removing the partition, verify that the partition can be removed by running the DROP PARTITIONNUM VERIFY statement. If the partition is in use, use the REDISTRIBUTE DATABASE PARTITION GROUP statement to redistribute the data to other partitions before removing the partition.

In our example, we want to drop database partition number 4. Take the steps shown in Example 3-15 to verify if the partition can be removed.

Example 3-15 Verifying if a database partition can be dropped

```
export DB2NODE=4
db2 TERMINATE
db2 DROP DBPARTITIONNUM VERIFY
SQL6034W  Node "4" is not being used by any databases.
```

We can now proceed to drop the partition by using the **db2stop... DROP DBPARTITIONNUM...** command as shown in Example 3-16.

Example 3-16 Dropping a database partition

```
db2stop DROP DBPARTITIONNUM 4

SQL6076W  Warning! This command will remove all database files on the
node for this instance.  Before continuing, ensure that there is no
user data on this node by running the DROP NODE VERIFY command.
Do you want to continue ? (y/n)
```

On successful completion of the **db2stop DROP DBPARTITIONNUM** command, the definition for partition 4 is removed from the db2nodes.cfg file, as well as the configuration files, and so forth.

In our test environment, if we try to drop partition number 3, we get the message shown in Example 3-17.

Example 3-17 Verifying if a database partition can be dropped

```
export DB2NODE=3
db2 terminate

DB20000I  The TERMINATE command completed successfully.

db2 DROP DBPARTITIONNUM VERIFY

SQL6035W  Node "3" is being used by database "TESTDB".
```

We have to redistribute the data from partition 3 first, before we can drop it.

A partition group can also be dropped by using the ALTER DATABASE PARTITION GROUP statement.

3.3.8 Creating database partition groups

After you have define and create the database partitions, the next step in building a partitioned database is creating the database partition groups. In our test environment, we created three database partition groups as shown in Example 3-18.

Example 3-18 Creating database partition groups

```
db2 "CREATE DATABASE PARTITION GROUP pg123 ON DBPARTITIONNUMS (1 to 3)"
DB20000I  The SQL command completed successfully.

db2 "CREATE DATABASE PARTITION GROUP pg1 ON DBPARTITIONNUM (1)"
DB20000I  The SQL command completed successfully.

db2 "CREATE DATABASE PARTITION GROUP pg23 ON DBPARTITIONNUMS (2,3)"
DB20000I  The SQL command completed successfully.
```

Partition group `PG123` spans database partitions `1`, `2`, and `3`. Partition group `PG1` spans only partition `1`. Partition group `PG23` spans partitions `2` and `3`.

3.3.9 Viewing partition groups

You view existing partition groups in a database by using the LIST DATABASE PARTITION GROUPS command. The SHOW DETAIL option shows the partition map ID, partition number, and if the partition is in use as shown in Example 3-19.

Example 3-19 Viewing database partition groups

```
db2 LIST DATABASE PARTITION GROUPS SHOW DETAIL

DATABASE PARTITION GROUP     PMAP_ID DATABASE PARTITION NUMBER IN_USE
---------------------------- ------- -------------------------------
IBMCATGROUP                        0                             0 Y
IBMDEFAULTGROUP                    1                             0 Y
IBMDEFAULTGROUP                    1                             1 Y
IBMDEFAULTGROUP                    1                             2 Y
IBMDEFAULTGROUP                    1                             3 Y
PG1                                4                             1 Y
PG123                              3                             1 Y
PG123                              3                             2 Y
PG123                              3                             3 Y
PG23                               5                             2 Y
PG23                               5                             3 Y
```

Notice that the IBMTEMPGROUP, which is created by default, does not appear in the listing.

3.3.10 Redistributing partition groups

You redistribute data in a partition group by using the REDISTRIBUTE DATABASE PARTITION GROUP statement. Use this statement to achieve uniform distribution across all partitions, to skew data, or to redistribute data prior to dropping a database partition. Tables in the IBMCATGROUP and the IBMTEMPGROUP cannot be redistributed.

After you define a database partition and add it to a partition group by using the ALTER DATABASE PARTITION GROUP statement, the data can be uniformly distributed with the UNIFORM keyword on the REDISTRIBUTE DATABASE PARTITION GROUP statement. In our test environment, we added database partition `4` and ran the REDISTRIBUTE DATABASE PARTITION GROUP statement as shown in Example 3-20.

Example 3-20 Redistributing a database partition group

```
db2 "REDISTRIBUTE DATABASE PARTITION GROUP pg123 UNIFORM"
```

You can use the REDISTRIBUTE DATABASE PARTITION GROUP statement with the DISTFILE option if the distribution of the distribution key values is skewed and you want uniform distribution. In this case, you must specify a distribution map file and provide it with the DISTFILE option.

Note: The REDISTRIBUTE PARTITION DATABASE GROUP statement creates a messages file in the <*instance home*>/sqllib/redist directory.

In our test environment, we created database partition `4` and assigned it to partition group `PG123`. We then redistributed partition group `PG123` uniformly to include partition `4`.

If we decide to remove data from partition `4`, we first have to redistribute the data by using the TARGETMAP option of the REDISTRIBUTE DATABASE PARTITION GROUP statement. The TARGETMAP file specifies how to distribute the data. In our example, we want to redistribute the data to database partitions `1`, `2`, and `3` and exclude partition `4`. An extract of our TARGETMAP file is shown in Example 3-21 on page 57. Use the target map file to indicate the desired distribution of data across the 4096 hash partitions. In our test environment, we created the target manually by using a text editor.

Example 3-21 Extract of a TARGETMAP file

```
1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
....
....
....
```

We use the TARGETMAP option on the REDISTRIBUTE DATABASE PARTITION GROUP statement as shown in Example 3-21 and specify `pg123.map` as our file. The TARGETMAP file `pg123.map` has 4096 entries in it.

Example 3-22 Redistributing a database partition by using a target map file

```
db2 "REDISTRIBUTE DATABASE PARTITION GROUP pg123 using TARGETMAP
pg123.map"

DB20000I  The REDISTRIBUTE NODEGROUP command completed successfully.
```

Once the REDISTRIBUTE DATABASE PARTITION GROUP statement completes successfully, partition group `PG123` has no data on partition `4`.

DB2 provides a utility called Get Distribution Map (**`db2gpmap`**), which gets the distribution map for the database table or the database partition group from the catalog partitioned database server. In our test environment, we dumped the current distribution map for partition group `PG123` to the output file `pg123.out` as shown in Example 3-23. Option -g specifies the partition group.

Example 3-23 Dumping a distribution map

```
db2gpmap -d testdb -m pg123.out -g pg123

Connect to testdb.
Successfully connected to database.
Retrieving the partition map ID using nodegroup PG123.
The partition map has been sent to pg123.out.
```

An extract of the file `pg123.out` is shown in Example 3-24.

Example 3-24 Extract of a distribution map

```
1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
.....
.....
```

You can obtain more information about the dbgpmap utility at:

http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.udb.admin.doc/doc/r0011837.htm

3.3.11 Altering database partition groups

You can alter database partition groups to add more database partitions to a partition group or to drop one or more partitions from a partition group.

Adding a database partition to a database partition group

You can add database partitions to a partition group by using the ALTER DATABASE PARTITION GROUP... ADD DBPARTITIONNUMS... command. Before a database partition can be added to a partition group, you must define the database partition in the db2nodes.cfg file.

In our test environment, we have just defined database partition `4` and want to add this new database partition to the `PG123` partition group as shown in Example 3-25.

Example 3-25 Adding a database partition

```
$ db2 "ALTER DATABASE PARTITION GROUP pg123 ADD DBPARTITIONNUMS (4)
WITHOUT TABLESPACES"

SQL1759W  Redistribute database partition group is required to change
database partitioning for objects in nodegroup "PG123" to include some
added database partitions or exclude some dropped database partitions.
SQLSTATE=01618
```

Upon successful completion of the ALTER DATABASE PARTITION GROUP... ADD statement, we are given a warning message indicating that a redistribute database partition is required. We can confirm that our new database partition

has been added to the PG123 partition group by listing the database partition groups as shown in Example 3-26.

Example 3-26 Viewing database partition groups

```
db2 LIST DATABASE PARTITION GROUPS SHOW DETAIL

DATABASE PARTITION GROUP     PMAP_ID DATABASE PARTITION NUMBER IN_USE
---------------------------- ------- ------------------------------
IBMCATGROUP                        0                            0 Y
IBMDEFAULTGROUP                    1                            0 Y
IBMDEFAULTGROUP                    1                            1 Y
IBMDEFAULTGROUP                    1                            2 Y
PG1                                4                            1 Y
PG123                              3                            1 Y
PG123                              3                            2 Y
PG123                              3                            3 Y
PG123                              3                            4 T
PG23                               5                            2 Y
```

Notice that the IN_USE column has a “T” for database partition 4, which we added to the PG123 partition group. The “T” indicates that the database partition has been added to the database partition group, but is not yet added to the distribution map.

The containers for the table spaces in the database partition group have not been added on this database partition. Table space containers must be added on the new database partition for each table space in the database partition group. The value is changed to “A” when containers have successfully been added.

In our test database, we added a container to our table space TBSP123 for database partition 4. When the container was added, we ran the REDISTRIBUTE DATABASE PARTITION GROUP statement (Example 3-27) to complete the process to add the partition.

Example 3-27 Redistributing a database partition group

```
db2 "REDISTRIBUTE DATABASE PARTITION GROUP pg123 UNIFORM"
DB20000I  The REDISTRIBUTE NODEGROUP command completed successfully.
```

After the command to redistribute completes successfully, the IN_USE status for partition group pg123 changes to “Y”.

Dropping a database partition from a partition group

If you are going to drop a database partition and the database partition has data on it, you must redistribute the data first before you can drop the database partition.

In our test environment, partition group PG123 has database partitions 1, 2, 3, and 4 assigned to it. Before we can drop database partition 4, we have to redistribute any data that exists on it. We achieve this by specifying the TARGETMAP option of the REDISTRIBUTE DATABASE PARTITION GROUP statement as shown in Example 3-28.

Example 3-28 Redistributing a database partition group

```
$ db2 "REDISTRIBUTE DATABASE PARTITION GROUP pg123 using TARGETMAP
pg123.map"

DB20000I  The REDISTRIBUTE NODEGROUP command completed successfully.
```

After the data has been redistributed successfully, the IN_USE flag on the LIST DATABASE PARTITION GROUP SHOW DETAIL statement for partition group PG123 on database partition 4 is set to “A”.

We can now proceed to drop database partition 4 from database partition group PG123 as shown in Example 3-29.

Example 3-29 Dropping a database partition group

```
db2 "ALTER DATABASE PARTITION GROUP pg123 DROP DBPARTITIONNUM (4)"

SQL20225W  The buffer pool operation (DROP) will not take effect until
the next database startup because the buffer pool is in use.
SQLSTATE=01657
```

In our example, we had a buffer pool assigned to partition group pg123. Upon successful completion of the ALTER DATABASE PARTION GROUP pg123 DROP DBPARTITIONNUM (4) statement, database partition 4 is no longer associated to partition group PG123. If we want, we can drop database partition 4 as shown in Example 3-30.

Example 3-30 Verifying and dropping a database partition number

```
db2 DROP NODE VERIFY

SQL6034W  Node "4" is not being used by any databases.

db2stop DROP DBPARTITIONNUM 4
```

```
SQL6076W  Warning! This command will remove all database files on the
node for this instance.  Before continuing, ensure that there is no
user data on this node by running the DROP NODE VERIFY command.
Do you want to continue ? (y/n)y
```

3.3.12 Dropping a database partition group

You can drop a database partition group by using the DROP DATABASE PARTITION GROUP statement. When a database partition group is dropped, all objects within the database partition group are dropped as shown in Example 3-31. The system partition groups IBMDEFAULT GROUP, IBMTEMPGROUP, and IBMCATGROUP cannot be dropped.

Example 3-31 Dropping a database partition group

```
db2 "DROP DATABASE PARTITION GROUP pg1"

SQL20225W  The buffer pool operation (DROP) will not take effect until
the next database startup because the buffer pool is in use.
SQLSTATE=01657
```

If a REDISTRIBUTE DATABASE PARTITION statement is executing, the database partition cannot be dropped. If a database partition group is partially redistributed, it can be dropped.

3.3.13 Implementing buffer pools

In a partitioned database environment, a default buffer pool is created for each partition unless database partition group is specified. If a partition group is specified, the buffer pool is only created in that partition group. You can view details about buffer pools in a database by querying the SYSCAT.BUFFERPOOLS system catalog table.

Creating buffer pools

You create buffer pools by using the CREATE BUFFERPOOL statement. There must be at least one buffer pool for each page size used in the database. The page size of the buffer pool cannot be altered after the buffer pool is created.

In our test environment, for illustration purposes, we have decided to create four buffer pools with four different page sizes and assign them to partition groups as shown in Example 3-32 on page 62. Each buffer pool is approximately 250 MB in size.

Example 3-32 Creating buffer pools

```
db2 CREATE BUFFERPOOL bp0 IMMEDIATE DATABASE PARTITION GROUP
ibmcatgroup SIZE 62500 PAGESIZE 4K;

db2 CREATE BUFFERPOOL bp1 IMMEDIATE DATABASE PARTITION GROUP pg1 SIZE
31250 PAGESIZE 8K;

db2 CREATE BUFFERPOOL bp23 IMMEDIATE DATABASE PARTITION GROUP pg23 SIZE
15625 PAGESIZE 16K;

db2 CREATE BUFFERPOOL bp123 DEFERRED DATABASE PARTITION GROUP pg123
SIZE 7812 PAGESIZE 32K;
```

Altering a buffer pool

You can alter an existing buffer pool by using the ALTER BUFFERPOOL statement. After a buffer pool is created, you can:

- Alter the buffer pool to change its size
- Add the buffer pool to a new database partition group
- Enable or disable automatic sizing for the buffer pool
- Change the block area of the buffer pool

Note: The page size of a buffer pool cannot be altered. You must create a new buffer pool of the page size that you want.

In our test environment, we changed the size of `BP0` to AUTOMATIC as shown in Example 3-33. The keyword AUTOMATIC enables automatic sizing of the buffer pool based on workload requirements.

Example 3-33 Altering a buffer pool

```
db2 "ALTER BUFFERPOOL bp0 IMMEDIATE SIZE AUTOMATIC"
DB20000I  The SQL command completed successfully.
```

Dropping a buffer pool

You can drop an existing buffer by using the DROP BUFFERPOOL statement. In our test environment, we dropped buffer pool `BP1234` as shown in Example 3-34 on page 63. The default buffer pool that is created, IBMDEFAULTBP, cannot be dropped, because it is a system object.

Example 3-34 Dropping a buffer pool

```
db2 "DROP BUFFERPOOL bp1234"
DB20000I  The SQL command completed successfully.
```

Note: A buffer pool cannot be dropped if it has other objects that depend on it, such as a table space. In addition, there must be at least one buffer pool per page size that is used in the database.

3.3.14 Implementing table spaces

In this section, we demonstrate creating, viewing, and altering table spaces.

Creating a table space

The CREATE TABLESPACE statement defines a new table space within the database and assigns containers to the table space. The table space can be created as a LARGE, REGULAR, SYSTEM TEMPORARY, or USER TEMPORARY table space. In a DPF environment, table spaces are created in partition groups. If no partition group is specified in the CREATE TABLESPACE statement, by default, REGULAR, LARGE, and USER TEMPORARY table spaces are created in the IBMDEFAULTGROUP partition group. SYSTEM TEMPORARY table spaces are created in the IBMTEMPGROUP partition group.

Note: SYSTEM TEMPORARY table spaces can only be created in the IBMTEMPGROUP partition group.

If the type of table space is not specified in the CREATE TABLESPACE statement, a LARGE table space is created.

Table space containers can be System-Managed Space (SMS) or Database-Managed Space (DMS). SMS containers are managed by the operating system and are directories. DMS containers are managed by the database and can have FILE or DEVICE containers. A table space can be defined to use AUTOMATIC storage.

In our test environment, we created three database-managed (DMS) table spaces in different partition groups with a size of 10 GB each and assigned buffer pools to these table spaces as shown in Example 3-35 on page 64. Each table space uses FILE containers.

Example 3-35 Creating table spaces

```
db2 "CREATE REGULAR TABLESPACE tbsp123 IN DATABASE PARTITION GROUP pg123
MANAGED BY DATABASE
USING  (FILE '/database/dbdisk1/testdb/cont123' 10G) ON DBPARTITIONNUM (1)
USING  (FILE '/database/dbdisk2/testdb/cont123' 10G) ON DBPARTITIONNUM (2)
USING  (FILE '/database/dbdisk3/testdb/cont123' 10G) ON DBPARTITIONNUM (3)
BUFFERPOOL bp123"

db2 "CREATE REGULAR TABLESPACE tbsp1 IN DATABASE PARTITION GROUP pg1 MANAGED BY
DATABASE
USING  (FILE '/database/dbdisk1/testdb/cont1' 10G) ON DBPARTITIONNUM (1)
BUFFERPOOL bp1"

db2 "CREATE REGULAR TABLESPACE tbsp23 IN DATABASE PARTITION GROUP pg23 MANAGED
BY DATABASE
USING  (FILE '/database/dbdisk2/testdb/cont23' 10G) ON DBPARTITIONNUM (2)
USING  (FILE '/database/dbdisk3/testdb/cont23' 10G) ON DBPARTITIONNUM (3)
BUFFERPOOL bp23"
```

Viewing table spaces

You can view table spaces in a database by running the LIST TABLESPACES SHOW DETAIL command as shown in Example 3-36.

Example 3-36 Viewing table spaces

```
db2 LIST TABLESPACES SHOW DETAIL

Tablespaces for Current Database

 Tablespace ID                        = 0
 Name                                 = SYSCATSPACE
 Type                                 = Database managed space
 Contents                             = All permanent data. Regular
                                        table space.
 State                                = 0x0000
   Detailed explanation:
     Normal
 Total pages                          = 16384
 Useable pages                        = 16380
 Used pages                           = 11260
 Free pages                           = 5120
 High water mark (pages)              = 11260
 Page size (bytes)                    = 4096
 Extent size (pages)                  = 4
 Prefetch size (pages)                = 4
 Number of containers                 = 1
```

Note: In a partitioned database server environment, only the table spaces on the current node are listed.

Viewing table space containers

You can view containers associated with a table space by running the LIST TABLESPACE CONTAINERS FOR... statement as shown in Example 3-37. The statement requires the table space ID to display the containers.

Example 3-37 Viewing table space containers

```
$ db2 LIST TABLESPACE CONTAINERS FOR 1

            Tablespace Containers for Tablespace 1

 Container ID                         = 0
 Name                                 =
/home/db2inst1/db2inst1/NODE0000/TESTDB/T0000001/C0000000.TMP
 Type                                 = Path
```

Altering a table space

You can use the ALTER TABLESPACE statement to alter characteristics and storage of an existing table space. You can modify characteristics such as the PREFETCHSIZE, BUFFERPOOL, OVERHEAD, TRANSFERRATE, and file system caching.

Regular DMS table spaces can be converted to LARGE table spaces.

Depending on the type of table space, containers can be dropped, added, extended, reduced, or resized.

When containers are added or extended, a rebalance of a table space might occur. *Rebalancing* involves moving table space extents from one location to another in an attempt to keep the data striped within the table space. You can avoid rebalancing of a table space if you use the BEGIN NEW STRIPE SET option of the ALTER TABLESPACE statement.

In our test environment, we added an additional FILE container to the DMS table space `TBSP123` after `TBSP123` was created as shown in Example 3-38 on page 66.

Example 3-38 Altering a table space

```
db2 ALTER TABLESPACE tbsp123 ADD
(FILE '/database/dbdisk1/testdb/cont1234' 10G) ON DBPARTITIONNUM (4)
```

Renaming a table space

You can rename a table space by using the RENAME TABLESPACE statement. In our test environment, we renamed the `USERSPACE1` table space as shown in Example 3-39.

Example 3-39 Renaming a table space

```
$ db2 "RENAME TABLESPACE userspace1 TO userdata"
DB20000I  The SQL command completed successfully.
```

You can issue the RENAME TABLESPACE statement from any database partition in a DPF environment even if the table space is not defined on the partition from where it is issued.

Dropping a table space

You can drop a table space by using the DROP TABLESPACE statement. Dropping a table space drops all the objects defined in a table space. A table space cannot be dropped if it has dependencies in another table space. In our test environment, we dropped table space `TBSP1` as shown in Example 3-40.

Example 3-40 Dropping a table space

```
db2 DROP TABLESPACE tbsp1
DB20000I  The SQL command completed successfully.
```

3.3.15 Implementing tables

In DB2, tables are created in table spaces. In a DPF-enabled environment, if a table space spans multiple database partitions, the tables created in the table space span multiple database partitions. A table cannot be created on just specific partitions if the table space spans multiple partitions. For example, if a table space spans three database partitions, a table cannot be created on just two of those three database partitions.

By default, if a table space name is not specified, a table space is assigned in a round-robin fashion from the list of table spaces that you can see with the LIST TABLESPACES command.

Creating tables

Create tables by using the CREATE TABLE statement. You can store table data, indexes, and long column data in the same table space or separate them by using the IN, INDEXES IN, and LONG IN options. In a DPF-enabled environment, you can specify the distribution key for the table by using the DISTRIBUTE BY clause. One of the tables in our test database is the `LINEITEM` table and it was created in table space `TBSP123` with `L_ORDERKEY` as the distribution key. Example 3-41 shows the DDL that we used to create the `LINEITEM` table.

Example 3-41 DDL to create the lineitem table

```
CREATE TABLE "DB2INST1"."LINEITEM"  (
                  "L_ORDERKEY" INTEGER NOT NULL ,
                  "L_PARTKEY" INTEGER NOT NULL ,
                  "L_SUPPKEY" INTEGER NOT NULL ,
                  "L_LINENUMBER" INTEGER NOT NULL ,
                  "L_QUANTITY" DECIMAL(15,2) NOT NULL ,
                  "L_EXTENDEDPRICE" DECIMAL(15,2) NOT NULL ,
                  "L_DISCOUNT" DECIMAL(15,2) NOT NULL ,
                  "L_TAX" DECIMAL(15,2) NOT NULL ,
                  "L_RETURNFLAG" CHAR(1) NOT NULL ,
                  "L_LINESTATUS" CHAR(1) NOT NULL ,
                  "L_SHIPDATE" DATE NOT NULL ,
                  "L_COMMITDATE" DATE NOT NULL ,
                  "L_RECEIPTDATE" DATE NOT NULL ,
                  "L_SHIPINSTRUCT" CHAR(25) NOT NULL ,
                  "L_SHIPMODE" CHAR(10) NOT NULL ,
                  "L_COMMENT" VARCHAR(44) NOT NULL )
                 DISTRIBUTE BY HASH("L_ORDERKEY")
                   IN "TBSP123" ;
```

You can obtain more details about the CREATE TABLE statement at:

http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.udb.admin.doc/doc/r0000927.htm

Altering tables

You can alter tables by using the ALTER TABLE statement.

Several characteristics of a table that you can alter by using the ALTER TABLE statement are:

- ADD or ALTER columns
- ADD, ATTACH, or DETACH table partitions
- ALTER foreign keys or check constraints

- DROP keys, columns, constraints, or distribution keys
- ADD security policy

You can obtain more information about the ALTER TABLE statement at:

http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.udb.admin.doc/doc/r0000888.htm

Renaming tables

You can rename tables by using the RENAME TABLE statement. We can rename the `LINEITEM` table in our test database as shown in Example 3-42.

Example 3-42 Renaming a table using the RENAME TABLE statement

```
db2 "RENAME TABLE lineitem TO lineitems_renamed"
DB20000I  The SQL command completed successfully.
```

You can obtain more information about the RENAME statement at:

http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.udb.admin.doc/doc/r0000980.htm

Dropping tables

You can drop tables by using the DROP TABLE statement. We can drop the name table in our test database as shown in Example 3-43.

Example 3-43 Dropping a table

```
$ db2 "DROP TABLE nation"
DB20000I  The SQL command completed successfully.
```

Viewing tables

View tables in a database by using the LIST TABLES command. In a DPF-enabled environment, issue the command from any database partition. In our test example, we can view all the tables for the `db2inst1` schema as shown in Example 3-44.

Example 3-44 Viewing a table with the LIST TABLES command

```
$ db2 LIST TABLES

Table/View                      Schema          Type  Creation time
------------------------------- --------------- ----- --------------
LINEITEM                        DB2INST1        T
2007-04-10-11.12.27.550965
```

```
NATION                          DB2INST1        T
2007-04-09-16.14.34.273300
ORDERS                          DB2INST1        T
2007-04-09-16.14.55.457627
PART                            DB2INST1        T
2007-04-09-16.14.45.529928
PARTSUPP                        DB2INST1        T
2007-04-09-16.14.50.492779
REGION                          DB2INST1        T
2007-04-09-16.14.43.237286
REGION_MQTR                     DB2INST1        S
2007-04-18-10.38.51.713419
SUPPLIER                        DB2INST1        T
2007-04-09-16.14.47.736387

8 record(s) selected.
```

You can obtain more details about the LIST TABLES command at:

http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.udb.admin.doc/doc/r0001967.htm

Viewing the current distribution of data in a table

To view the current data distribution, you can run the following SQL statement. This statement shows you the number of rows per partition that you have in a table.

```
SELECT     DBPARTITIONNUM(distribution key), COUNT( * )
FROM       schema.table
GROUP BY   DBPARTITIONNUM(distribution key)
ORDER BY   DBPARTITIONNUM(distribution key)
```

Substitute the appropriate values for *schema.table* and the *distribution key* column. In our test environment, we can determine the distribution of data in the `LINEITEM` table as shown in Example 3-45 on page 70. The first column is the database partition number and the second column is the number of rows for the `LINEITEM` table on a database partition. In our example, the `LINEITEM` table is fairly evenly distributed.

Example 3-45 Viewing the current data distribution of a table using SQL

```
db2 "SELECT dbpartitionnum(l_orderkey), COUNT(*) FROM db2inst1.lineitem
GROUP BY dbpartitionnum(l_orderkey) ORDER BY
dbpartitionnum(l_orderkey)"

1           2
----------- -----------
          1     3995488
          2     4001348
          3     4001160

  3 record(s) selected.
```

Viewing the distribution map of a table

You can view the current distribution map of a table by using the **db2gpmap** utility. DB2 provides a utility called Get Distribution Map (**db2gpmap**), which gets the distribution map for the database table or the database partition group from the catalog partitioned database server. Option -t specifies the table name. In our test environment, we dumped the current distribution map for the `lineitem` table to the output file called `lineitem.out` as shown in Example 3-46.

Example 3-46 Viewing a distribution map

```
$ db2gpmap -d testdb -m lineitem.map -t lineitem

Connect to testdb.
Successfully connected to database.
Retrieving the partition map ID using table LINEITEM.
The partition map has been sent to lineitem.map.

An extract of the lineitem.map file:

1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
....
....
....
```

3.4 Implementing DPF on Windows

In this section, we look at the DB2 9 Windows installation and at several of the commands for implementing DPF that are specific to the Windows platform.

3.4.1 Installing DB2 Enterprise 9 on Windows

The first step in setting up a partitioned environment on Windows is to ensure that your participating servers meet the minimum requirements and are part of a Windows domain. We recommend the following steps prior to installation:

1. Verify that each computer meets the necessary operating system, memory, and disk requirements.
2. Ensure that all computers belong to the same Windows domain.
3. Ensure that all computers have consistent time and date settings.
4. Verify that all computers can communicate with each other via TCP/IP.
5. Add a domain user account to the local Administrator group on each computer.
6. Optionally, create DB2 user accounts for setup.

For more in-depth information about the Windows installation requirements, see *Quick Beginnings for DB2 Servers*, GC10-4246, and *Getting Started with DB2 Installation and Administration on Linux and Windows*, GC10-4247. Information is also available in the DB2 9 Information Center at:

http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.udb.uprun.doc/doc/r0025127.htm

Installing the instance owning partition

Here are the steps to install the instance owning partition:

1. The first step is to log on to the server that will be the instance owning server and start the installation wizard. You can log on with a local administrator ID or a domain ID that has local administrative rights. However, this user must have the “Access this computer from the network” advanced user right in order to successfully complete the installation. Run setup.exe to start the installation launchpad as shown in Figure 3-3 on page 72.

Figure 3-3 Installation launchpad

2. Run through the installation, selecting options for typical or custom install, response file creation, and installation path. The wizard prompts you to specify the user ID and password for the DB2 Administration server (DAS). The DAS user performs administrative and configuration tasks locally or remotely on this server and other partitioned database servers. It is important to have this user under a domain users group to grant access to all participating database partitions. The account used by the DAS must have the following advanced user rights:

   – Act as part of the operating system.
   – Debug programs.
   – Create token object.
   – Lock pages in memory.
   – Log on as a service.
   – Increase quotas.
   – Replace a process level token.

   The DAS account is granted these user rights if the account already exists or if the installation process creates the account.

Note: You must ensure that you install DB2 on the same drive on each participating server. For example, do not install DB2 on the C: drive of the instance owning database server, on the D: drive of a database partition server, or on the J: drive of another database partition server. If you install DB2 on the C: drive of the instance owning database server, install DB2 on the C: drive of any participating database partition servers.

3. In the “Set up a DB2 instance” window, specify whether to create a new default DB2 instance or join an existing partitioned environment. Because this server is our instance owning partition server, we select **Create the default DB2 instance** as shown in Figure 3-4.

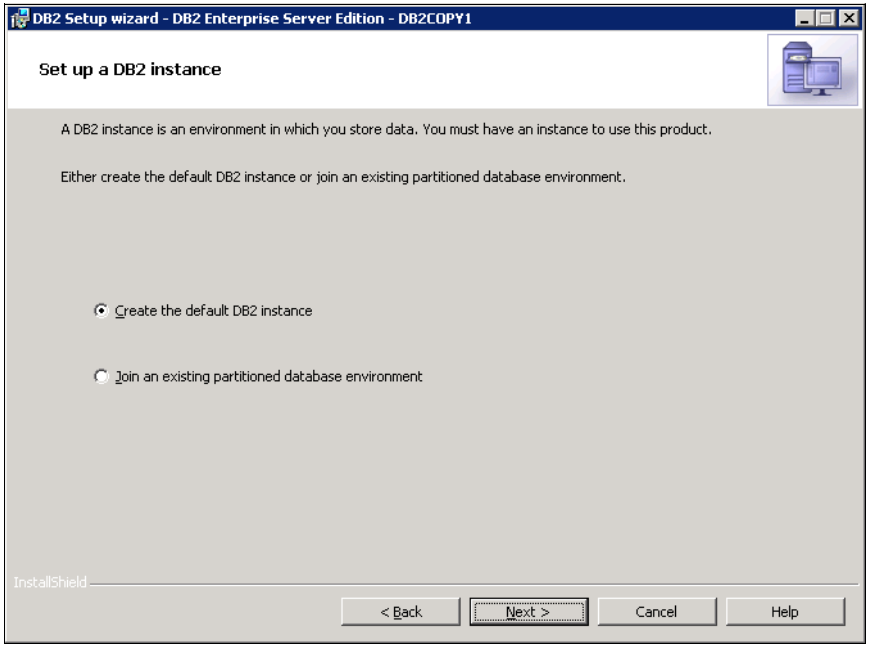


Figure 3-4 Default instance selection

4. In the “Set up partitioning options for the default DB2 instance” window, select the type of instance to create. Because we are setting up a multi-partition environment, we select **Multiple-partition instance** from the options. See Figure 3-5 on page 74. Here, we can also specify the number of logical partitions.

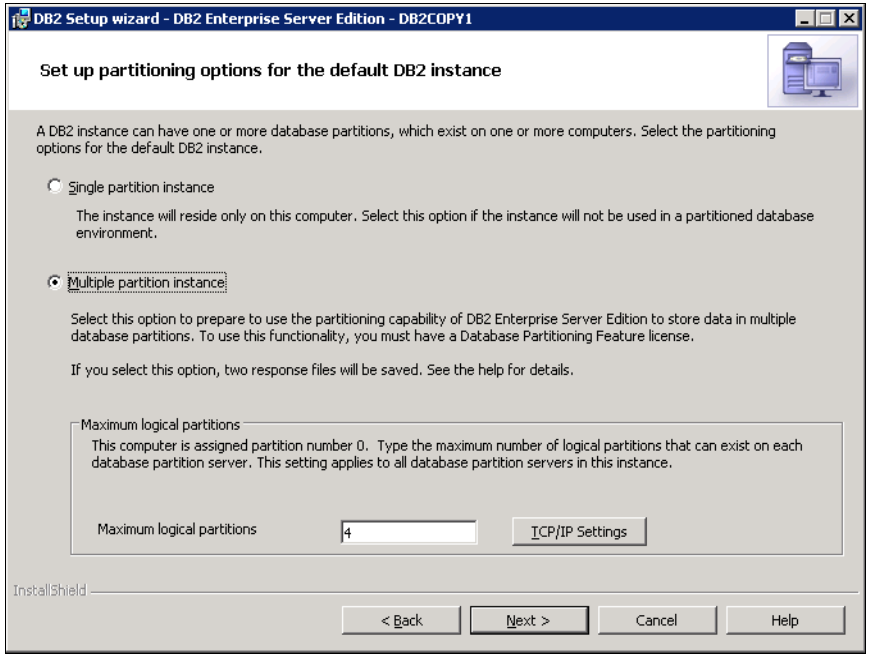


Figure 3-5 Instance type selection

5. In the “Configure DB2 instances” step (Figure 3-6 on page 75), specify the communications options for the instance. Leaving these as defaults is sufficient.

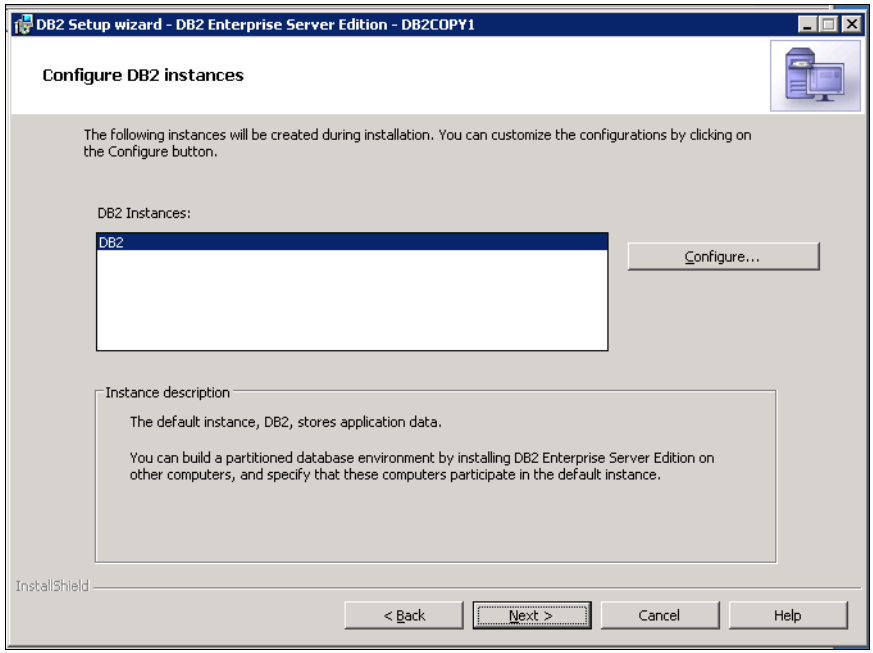


Figure 3-6 Instance communications options

6. In the “Set user information for the default DB2 instance” window (Figure 3-7 on page 76), specify an account that the instance uses to start. You can define a user before starting the installation, or you can have the DB2 Setup wizard create a new domain user for you. If you want to create a new domain user by using the DB2 Setup wizard, the account used to perform the installation must have the authority to create domain users. The instance user domain account must belong to the local Administrators group on all the participating servers and is granted the following user rights:
   - Act as part of the operating system.
   - Debug programs.
   - Create token object.
   - Increase quotas.
   - Log on as a service.
   - Replace a process level token.
   - Lock pages on memory.

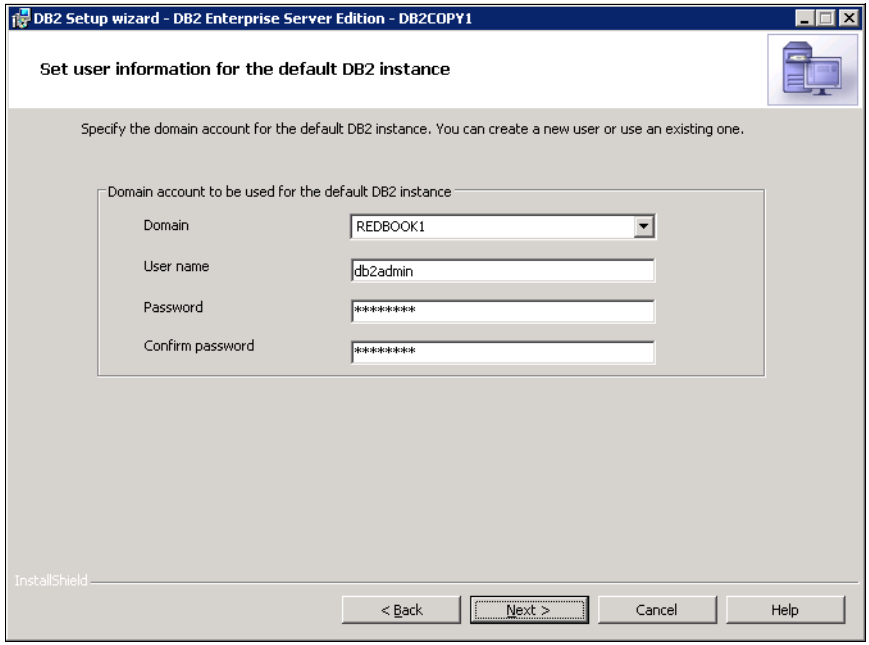


Figure 3-7 Instance owning user ID selection

7. The next two windows give you options for setting up the tools catalog and notification options for e-mail and pager notifications.
8. Next, we are required to specify the operating system security options. This allows the DB2 binaries to be secured by using NTFS permissions. We elected to enable OS security.

   After this step is complete, you see a summary and the installation completes.

   The installation process updates the services file and the db2nodes.cfg file.

   Now that we have our instance owning partition setup, we can proceed to set up an additional physical partition.

Installing additional partitions

The steps to install additional physical partitions are:

1. Start the installation launchpad on the Windows server that will act as an additional physical partition. To do this, run **`setup.exe`**. The installation initially uses the same steps as the instance-owning partition server; however, we select the **Join an existing partitioned database environment** option rather than **Create the default DB2 instance** as shown in Figure 3-8 on page 77. This joins this partition server to an existing partition server environment.

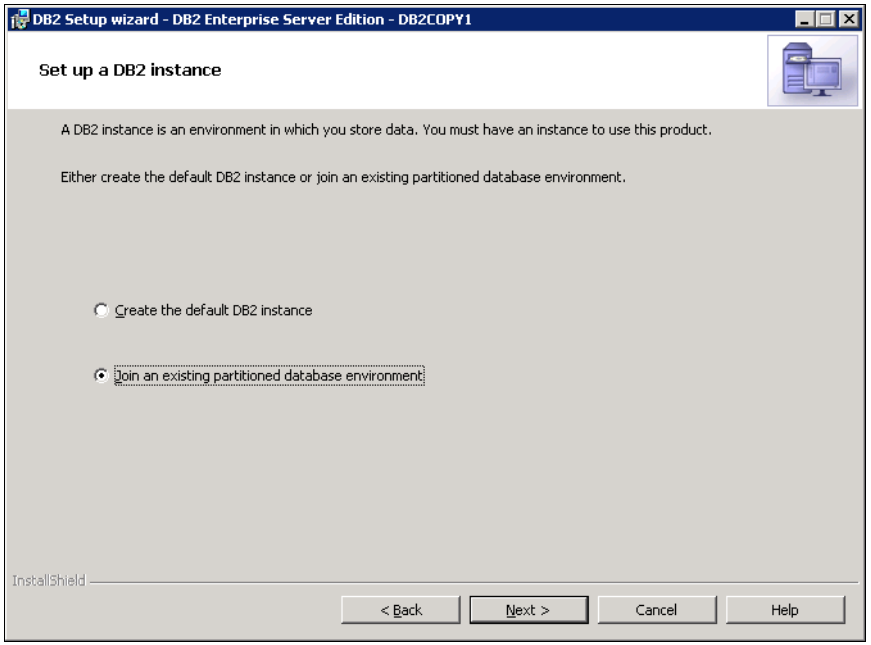


Figure 3-8 Join an existing partitioned environment

2. The next window allows us to add the new partition to an existing partition server environment. The first step is to select the button next to the instance-owning server field. This allows you to select the partition server environment that you want to join as shown in Figure 3-9 on page 78. Select the instance owning server from the displayed list of servers in the domain and select **OK**. In the “Add a new database partition server” window, specify the user account that the new partition will use. It needs to be the same domain account used when the instance-owning partition server was set up. Figure 3-10 on page 78 shows a summary of the options that we have selected.

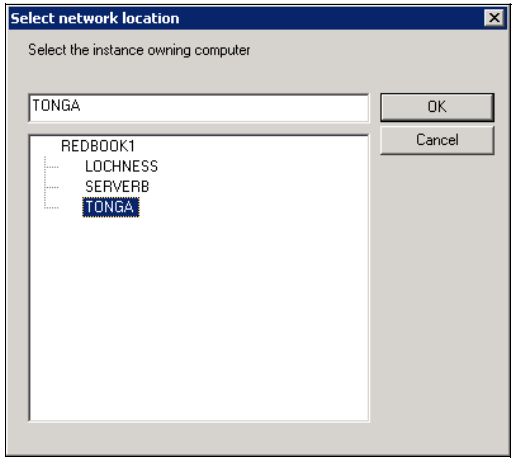


Figure 3-9 Instance owning partition selection

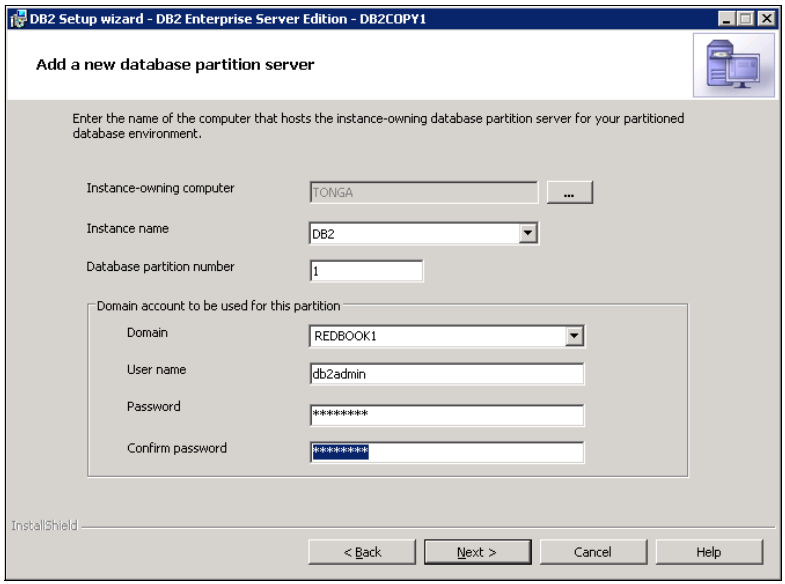


Figure 3-10 Add a new partition server

3. The next window gives us the option of setting up the operating system security.
4. The final step, when the installation completes, is to start the new partition. From the instance owning partition, issue a **db2stop**. You notice that only one partition is stopped (the partition on the instance owning server). This is

because the new partition is not available until DB2 has been restarted as shown in Example 3-47.

Example 3-47 Stopping and starting db2

```
C:\>db2stop
04/05/2007 10:50:13 0 0 SQL1064N  DB2STOP processing was successful.
SQL1064N  DB2STOP processing was successful.
C:\>db2start
04/05/2007 10:50:19 0 0   SQL1063N  DB2START processing was successful.
04/05/2007 10:50:37 1 0   SQL1063N  DB2START processing was successful.
SQL1063N  DB2START processing was successful.
```

Using db2icrt to create a partitioned instance

To create a partitionable instance on a Windows server where DB2 has already been installed, you can use the **db2icrt** command. You must specify the user ID and Fast Communication Manager (FCM) port range to create a instance that can be partitioned. Example 3-48 shows the command to create an instance called DPF.

Example 3-48 Create a partitionable instance

```
db2icrt -s ese -u db2admin,eliza -r 60001,60005 DPF
```

Where:

/s Is the instance type. In this case, an Enterprise Server instance.

/u Is the instance owning user ID and password.

/r Is the port range used by the FCM. This facilitates inter-partition communications.

3.4.2 Working with partitioned databases

Here, we look at several of the commands specific to the Windows platform that allow you to create and work with database partitions.

db2ncrt

The **db2ncrt** command allows you to add new logical or physical partitions. The command creates the necessary Windows service for the new partition and updates the db2nodes.cfg file on the instance owning machine. After the **db2ncrt** command is issued, you must recycle the instance with the **db2stop** and **db2start** commands in order for the new partition to become active. When you have issued the **db2stop**, the system only stops the current configured instance,

at which point the db2nodes.cfg file on the instance owner is updated with information for DB2 to communicate with the new partition.

> **Note:** Only use **db2ncrt** if no databases exist in the instance. If you already have an instance in which a database has already been created, you must always use the **db2start add dbpartitionnum** command to create additional partitions and redistribute your data.

For example, you have a server (server_a) with a single partition instance configured (DB2). You want to add an additional physical partition server (server_b). On server_b, you execute the command in Example 3-49.

Example 3-49 db2ncrt command example

```
db2ncrt /n:1 /u:DOMAIN1\db2admin,eliza /i:DB2 /m:server_b /p:0
/o:server_a
```

Where:

- /n Is the partition number for server_b.
- /n Is the partition number for server_b.
- /u Is the domain DB2 administrator username and password on server_b.
- /i Is the instance name.
- /m Is the name of the server where the partitioned instance is added.
- /p Is the logical port used for the database partition server. If it is the first partition on this server, it must start with port number 0. The port number cannot exceed the port range reserved for FCM communications in the x:\windows\system32\drivers\etc\ services. For example in our case, a range of 4 ports is reserved for the current instance; therefore, the maximum port number is 3.
- /o Is the instance owning server.

db2nchg

The **db2nchg** command allows you to change the configuration of a database partition. Options include: selecting a different logical port number or a different network name for the database partition, changing the TCP/IP host name of the machine, and moving the database partition from one machine to another. Only use this command when the database partition is stopped.

db2ndrop

The **db2ndrop** command drops a partition from an instance that has no databases. As with the **db2nchg** command, only use **db2ndrop** if the database partition is stopped.

If we want to drop partition 1, we issue the command shown in Example 3-50.

Example 3-50 Using db2ndrop

```
C:\>db2ndrop /n:1
SQL2808W  Node "1" for instance "DB2" has been deleted.
```

Note: If you use **db2ndrop** to remove a partition when the database still exists in the instance, you lose the data on that partition. If you want to drop a database partition from a partition server environment, we recommend using the **db2stop drop dbpartition** command in conjunction with data redistribution. See 3.3.10, “Redistributing partition groups” on page 56 for further details.

3.4.3 DB2 Remote Command Service

The DB2 Remote Command Service (**db2rcmd.exe**) automatically handles all inter-partition administrative communications. Ensure that this service is running at all times in your partitioned environment and has been configured to start with the domain user ID that was used to start the partitioned instance.

3.5 Administration and management

In this section, we discuss the partitioned database administration and management.

3.5.1 DB2 utilities

In this section, we discuss the available DB2 utilities for managing a partitioned database environment.

db2_all

In a DPF-enabled environment, the **db2_all** utility is provided to issue commands remotely to all database partitions in an instance.

In our test environment, if we want to serially update a data configuration on all four database partitions, we issue the **db2_all** command as shown in Example 3-51.

Example 3-51 Using db2_all serially

```
$ db2_all "db2 UPDATE DB CFG FOR TESTDB USING LOGRETAIN ON"

DB20000I  The UPDATE DATABASE CONFIGURATION command completed
successfully.
Clyde: db2 UPDATE DB CFG ... completed ok

DB20000I  The UPDATE DATABASE CONFIGURATION command completed
successfully.
Clyde: db2 UPDATE DB CFG ... completed ok

DB20000I  The UPDATE DATABASE CONFIGURATION command completed
successfully.
Clyde: db2 UPDATE DB CFG ... completed ok

DB20000I  The UPDATE DATABASE CONFIGURATION command completed
successfully.
Clyde: db2 UPDATE DB CFG ... completed ok
```

If we want to update the database configuration parameter concurrently, we place a semicolon (;) in front of the **db2** command as shown in Example 3-52.

Example 3-52 Using db2_all in parallel

```
db2_all ";db2 UPDATE DB CFG FOR TESTDB USING LOGRETAIN ON"
```

To run a command on just a particular database partition, use the “<<+ <“ option with **db2_all**. In our test environment, we can back up just partition 0 as shown in Example 3-53.

Example 3-53 Using db2_all against a specific partition

```
db2_all "<<+0< db2 BACKUP DB testdb to /home/db2inst1/BACKUPS"
```

To run a command on all partitions except one partition, use the “<<- <“ option with **db2_all**. In our test environment, we can back up all partitions except partition 0 as shown in Example 3-54.

Example 3-54 Using db2_all to omit a partition

```
db2_all "<<-0< db2 BACKUP DB testdb to /home/db2inst1/BACKUPS"
```

When using **db2_all**, more than one DB2 statement can be issued by separating each statement with a semicolon (;). For example, to list all the table spaces on all database partitions, a database connection is needed for the LIST TABLESPACES command to succeed as shown in Example 3-55.

Example 3-55 Listing table spaces on all partitions

```
db2_all "db2 CONNECT TO testdb ;db2 LIST TABLESPACES SHOW DETAIL"
```

If the two commands in our example are not issued together, an error message is returned by DB2 on each partition as shown in Example 3-56.

Example 3-56 Listing table spaces

```
db2_all "db2 LIST TABLESPACES SHOW DETAIL"

SQL1024N  A database connection does not exist.  SQLSTATE=08003
Clyde: db2 LIST TABLESPACES ... completed rc=4
...
...
```

Database backup

Database backups are taken to protect against the possibility of losing data due to hardware or software failures or both. A well-rehearsed recovery strategy must be in place. DB2 provides the backup utility for taking backups.

You can take DB2 backups offline or online. An offline backup is also known as a *cold backup*. Offline backups can be taken when there are no users or applications connected to the database. An offline backup requires that the database is unavailable to users during the backup. Online backups can be taken while there is activity against the database and do not require that the database is unavailable to users and applications.

DB2 backups can be full database backups or table space level backups. These backups can be taken as incremental or delta backups. The database transaction logs can also be backed up in the backup image.

DB2 supports taking a backup to disk, Tivoli® Storage Manager (TSM), or an X/Open Backup Services Application Programmer's Interface (XBSA). You can optionally compress DB2 backups by using a compression library to reduce the size of the backup images.

In a DPF-enabled environment, database backups are taken at a database partition level, that is, each database partition has to be backed up individually. Offline backups require the catalog partition to be backed up first.

In our test environment, we have four database partitions. For a full database backup, we have to back up four database partitions. We start by taking a backup of the catalog partition first, because this is an *offline* backup as shown in Example 3-57. After this completes successfully, we can back up the rest of the partitions.

Example 3-57 Backing up a database

```
db2_all "<<+0< db2 BACKUP DB testdb to /home/db2inst1/BACKUPS"

Backup successful. The timestamp for this backup image is :
20070409100349

Clyde: db2 BACKUP DB testdb ... completed ok


db2_all "|<<-0< db2 BACKUP DB testdb ONLINE to /home/db2inst1/BACKUPS"
rah: omitting logical node 0

Backup successful. The timestamp for this backup image is :
20070409100510

Clyde: db2 BACKUP DB testdb ... completed ok

...
...
```

Example 3-58 illustrates an *online* backup of table space `TBSP1`.

Example 3-58 Backing up a table space

```
db2 BACKUP DB TESTDB TABLESPACE tbsp1 ONLINE TO /home/db2inst1/BACKUPS
```

Monitoring backup

When the DB2 backup utility is executing, you can monitor it by using the LIST UTILITIES SHOW DETAIL command. In a DPF-enabled environment, this command only returns information about the database partition where it is executed. In our test environment, we can monitor the backup utility as shown in Example 3-59.

Example 3-59 Monitoring backup

```
export DB2NODE=0
db2 terminate

db2 LIST UTILITIES SHOW DETAIL
```

```
ID                                = 5
Type                              = BACKUP
Database Name                     = TESTDB
Partition Number                  = 0
Description                       = offline db
Start Time                        = 04/09/2007 10:41:23.587256
State                             = Executing
Invocation Type                   = User
Throttling:
   Priority                       = Unthrottled
Progress Monitoring:
   Estimated Percentage Complete = 0
      Total Work                  = 47713041 bytes
      Completed Work              = 0 bytes
      Start Time                  = 04/09/2007 10:41:23.598681
```

Viewing the backup history

Whenever the DB2 BACKUP utility is invoked, an entry is made in the DB2 history file to record the event. You can view the history file for backups in a database by using the LIST HISTORY BACKUP ALL command as shown in Example 3-60. In a DPF-enabled environment, the LIST HISTORY command only returns information about the database partition where it is executed.

Example 3-60 Viewing the backup history

```
export DB2NODE=0
db2 terminate

db2 LIST HISTORY BACKUP ALL FOR testdb


List History File for testdb

Number of matching file entries = 7


 Op Obj Timestamp+Sequence Type Dev Earliest Log Current Log  Backup ID
 -- --- ------------------ ---- --- ------------ ------------ ---------
  B  D  20070409094628001   F    N  S0000000.LOG S0000000.LOG

----------------------------------------------------------------------------
  Contains 1 tablespace(s):

  00001 TBSP1
```

```
----------------------------------------------------------------------
    Comment: DB2 BACKUP TESTDB OFFLINE
 Start Time: 20070409094628
   End Time: 20070409094629
     Status: A

----------------------------------------------------------------------
  EID: 13 Location: /dev/null
```

Verifying the backup images

DB2 provides the DB2 check backup utility, **db2ckbkp**, which you can use to determine the integrity of a backup image. It can also be used to display the metadata stored in a backup image header. You can only use the **db2ckbkp** utility on backup images stored on disk or tape. If the backup image is stored on TSM, it has to be retrieved to disk first before the utility can be executed.

In our test environment, the integrity of the backup image on partition 1 can be checked as shown in Example 3-61.

Example 3-61 Verifying a backup image

```
$ db2ckbkp TESTDB.0.db2inst1.NODE0001.CATN0000.20070409104130.001

[1] Buffers processed:  ###

Image Verification Complete - successful.
```

In a DPF environment, run the **db2ckbkp** utility on all the backup images, because there is a backup image for each partition that was backed up.

RESTORE

Use the DB2 RESTORE utility to recover previously DB2-backed up databases or table spaces after a problem, such as a media or storage failure, power interruption, or application failure.

The DB2 RESTORE utility can be used to restore databases, table spaces, the history file, or database transaction logs that have been stored in the backup image. DB2 restore also supports rebuilding a database using table space backup images.

DB2 supports restoring from disk, Tivoli Storage Manager (TSM), or an X/Open Backup Services Application Programmer's Interface (XBSA).

In a DPF-enabled environment, the RESTORE utility has to be performed for each partition to recover a database or table space. The **db2_all** utility can be used to restore the database partitions.

> **Note:** When performing a restore using the **db2_all** utility, always specify REPLACE EXISTING and WITHOUT PROMPTING. Otherwise, if there is prompting, the operation appears to be stopped or in a loop. The **db2_all** utility does not support user prompting.

In our test environment, we can restore our entire database as shown in Example 3-62.

Example 3-62 Restoring all the database partitions

```
db2_all "<<+0< db2 RESTORE DATABASE testdb FROM /home/db2inst1/BACKUPS
TAKEN AT 20070431234149 INTO testdb REPLACE EXISTING"

db2_all "<<+1< db2 RESTORE DATABASE testdb FROM /home/db2inst1/BACKUPS
TAKEN AT 20070431234427 INTO testdb REPLACE EXISTING"

db2_all "<<+2< db2 RESTORE DATABASE testdb FROM /home/db2inst1/BACKUPS
TAKEN AT 20070431234828 INTO testdb REPLACE EXISTING"

db2_all "<<+3< db2 RESTORE DATABASE testdb FROM /home/db2inst1/BACKUPS
TAKEN AT 20070431235235 INTO testdb REPLACE EXISTING"
```

DB2 restore is an offline process, unless table spaces are restored, in which case, the ONLINE keyword can be used. This allows access to the rest of the database while the specific table spaces are restored.

If a backup that is being restored was taken online, at the end of the restore process, the database needs to roll forward through the transaction logs before the database is available.

Monitoring restore

When the DB2 RESTORE utility executes, you can monitor it by using the LIST UTILITIES SHOW DETAIL command. In a DPF-enabled environment, the LIST UTILITIES command only returns information about the database partition where it is executed. In our test environment, we can monitor the restore utility as shown in Example 3-63 on page 88.

Example 3-63 Monitoring restore

```
db2 LIST UTILITIES SHOW DETAIL

ID                                = 11
Type                              = RESTORE
Database Name                     = TESTDB
Partition Number                  = 0
Description                       = db
Start Time                        = 04/09/2007 12:09:03.058149
State                             = Executing
Invocation Type                   = User
Progress Monitoring:
      Completed Work              = 67129344 bytes
      Start Time                  = 04/09/2007 12:09:03.058157
```

The IMPORT utility

The IMPORT utility inserts data from an external file into a table, typed table hierachy view, or nickname. You can use the IMPORT utility to append data to a table, replace data in a table, or create a table and insert data into it when using the IXF file format.

You can run the IMPORT utility offline by using the ALLOW NO ACCESS keyword or online by using the ALLOW WRITE ACCESS keyword. The ALLOW WRITE ACCESS keyword is not supported with the REPLACE_CREATE, REPLACE, and CREATE keywords of the IMPORT utility.

By default, the IMPORT utility issues a commit at the end of the import. Because the IMPORT utility issues SQL inserts, each row inserted is logged in the transaction logs. Inserting large amounts of data can exhaust the transaction logs. To commit more frequently, use the COMMITCOUNT keyword to specify the frequency of rows to be committed. In general, the more frequent the commits for an online import, the greater the concurrency.

By default, the IMPORT utility inserts one row per INSERT statement. Using the MODIFIED BY COMPOUND=*x* keyword (where 0<*x*<=100) allows multiple rows to be written in each INSERT statement, which can result in better performance.

In a DPF-enabled environment, consider enabling buffered inserts for the IMPORT utility. You achieve this by rebinding the db2uimpm.bnd package with the INSERT BUF option as shown in Example 3-64 on page 89.

Example 3-64 Enabling buffered inserts

```
db2 BIND db2uimpm.bnd INSERT BUF

LINE    MESSAGES FOR db2uimpm.bnd
------------------------------------------------------------------------
        SQL0061W  The binder is in progress.
        SQL0091N  Binding was ended with "0" errors and "0" warnings.
```

In our test environment, we issued an IMPORT as shown in Example 3-65 to insert data into the `LINEITEM` table.

Example 3-65 Using the IMPORT utility

```
db2 "IMPORT FROM lineitem.tbl OF DEL MODIFIED BY COLDEL| COMPOUND=100
COMMITCOUNT 100000 REPLACE INTO lineitem"

SQL3109N  The utility is beginning to load data from file
"lineitem.tbl".

SQL3221W  ...Begin COMMIT WORK. Input Record Count = "100000".
....
```

The EXPORT Utility

You can use the DB2 EXPORT utility to unload data from tables into a varying number of external file formats. You can specify the data to be exported by a SELECT statement. Data that is exported by the EXPORT utility can be imported or loaded by using the IMPORT and LOAD utilities.

In our test environment, we exported the entire `LINEITEM` table as shown in Example 3-66.

Example 3-66 Using the EXPORT utility

```
db2 "EXPORT TO lineitem.del OF DEL SELECT * FROM db2inst1.lineitem"

SQL3104N  The Export utility is beginning to export data to file
"lineitem.del".
```

Note: The IXF file format cannot be used to load or import into a table defined in a partitioned database environment.

The LOAD Utility

The DB2 LOAD utility loads data from a file, named *pipe*, or *tape*. Unlike the IMPORT utility, the LOAD utility does not perform SQL inserts to load the data. Instead, the LOAD utility copies formatted pages directly into the database. In general, the LOAD utility performs better than the IMPORT utility.

The LOAD utility does not fire triggers or perform referential or table constraint checking; it only validates the unique constraints of indexes.

You can run the LOAD utility offline with the ALLOW NO ACCESS keyword or in read only mode by using the ALLOW READ ACCESS keyword. Only data that was available in a table prior to the load is accessible. By default, LOAD does not allow access to the table while it is loaded.

Execute the LOAD utility with the COPY YES/NO or the NONRECOVERABLE option. The COPY YES specifies that a copy of the data loaded is saved. The NONRECOVERABLE option of the LOAD utility specifies that you cannot recover the table being loaded by rollforward during recovery. Example 3-67 illustrates loading data into the `LINEITEM` table by using the LOAD utility.

Example 3-67 Using the LOAD utility

```
db2 "LOAD FROM lineitem.tbl OF DEL MODIFIED BY COLDEL| REPLACE INTO
lineitem STATISTICS YES NONRECOVERABLE"
```

REORG utility

Use the DB2 REORG utility to reorganize a table or an index.

Index reorganization

You can reorganize all indexes defined on a table by rebuilding the index data into unfragmented, physically contiguous pages. You achieve this by using the INDEXES ALL FOR TABLE keyword. You can also reorganize a table by a specific index by using the INDEX keyword. While indexes are reorganized, you can use the REORG options ALLOW NO ACCESS, ALLOW READ ACCESS, or ALLOW WRITE ACCESS to the table on which the indexes are reorganized. You can also use the REORG utility to CONVERT Type 1 indexes to Type 2 indexes. When the REORG utility is run with the CLEANUP ONLY option, a full reorganization is not done. The indexes are not rebuilt and any pages freed up are available for reuse by indexes defined on this table only.

When reorganizing indexes, the amount of sort memory available to sort the index keys has a significant impact on performance.

In our test environment, we reorganized all the indexes on the `LINEITEM` table as shown in Example 3-68 on page 91.

Example 3-68 Using the REORG utility on indexes

```
$ db2 "REORG INDEXES ALL  FOR TABLE lineitem ALLOW WRITE ACCESS"
DB20000I  The REORG command completed successfully.
```

Table reorganization

When a table is reorganized by using the REORG utility, you can use the INDEX keyword to reorganize the table according to that index. If the INDEX option is not specified to the REORG utility and if a clustering index exists on the table, the data is ordered according to the clustering index.

Table reorganization can be offline or online. When an offline reorganization is run, the ALLOW NO ACCESS or the ALLOW READ ACCESS keywords can be specified with the REORG utility. Offline table reorganization is the default and is the fastest method to reorganize a table. Offline reorganization of a table incurs a large space requirement, because the entire copy of the table needs to be rebuilt. An offline reorganization is synchronous.

If there is insufficient space within the table space to hold the table while it is reorganized, a separate temporary table space must be specified.

In order to run the REORG utility online, you must specify the INPLACE keyword. The INPLACE option allows user access to the table during a table reorganization by specifying the ALLOW READ ACCESS or ALLOW WRITE ACCESS keywords. An INPLACE table reorganization is asynchronous.

In our test environment, we ran an online table reorganization on all the database partitions as shown in Example 3-69.

Example 3-69 Using the REORG utility on tables

```
$ db2 "REORG TABLE lineitem INPLACE ALLOW WRITE ACCESS ON ALL
DBPARTITIONNUMS"

DB20000I  The REORG command completed successfully.
DB21024I  This command is asynchronous and may not be effective
immediately.
```

Note: DB2 supplies the **`reorgchk`** command to assist in determining whether tables and indexes need to be reorganized.

Monitoring REORG

You can monitor table reorganizations by using the GET SNAPSHOT FOR TABLES command. In our test environment, we specified the GLOBAL keyword

to get a table snapshot of all database partitions as shown in Example 3-70 on page 92.

Example 3-70 Monitoring the reorg utility

```
db2 "GET SNAPSHOT FOR TABLES on testdb GLOBAL"

Table Reorg Information:
   Node number       = 2
   Reorg Type        =
        Reclaiming
        Inplace Table Reorg
        Allow Write Access
   Reorg Index       = 0
   Reorg Tablespace  = 2
   Start Time        = 04/11/2007 10:21:13.214769
   Reorg Phase       =
   Max Phase         =
   Phase Start Time  =
   Status            = Started
   Current Counter   = 43841
   Max Counter       = 138819
   Completion        = 0
   End Time          =
```

The RUNSTATS utility

Use the RUNSTATS utility for collecting statistics about the data stored in tables and indexes. These statistics are stored in the system catalog tables. When generating access plans, the DB2 query optimizer uses these statistics. Keeping statistics current is crucial in generating optimal access plans. If tables are frequently inserted into, updated, or deleted, the RUNSTATS utility needs to be run more frequently to keep the statistics current.

You can collect statistics on tables, indexes, or both. Collect distribution statistics for a table by specifying the WITH DISTRIBUTION keyword. You can collect detailed index statistics by using the FOR DETAILED INDEXES ALL keyword.

You can run the RUNSTATS utility with the ALLOW WRITE ACCESS or the ALLOW READ ACCESS keywords, which allows access to the table while statistics are collected on the table.

Use the WITH DISTRIBUTION option when the data is known to have nonuniform data distribution and the workload is capable of exploiting nonuniform data distribution statistics.

Collect the DETAILED index statistics when the table has multiple unclustered indexes with varying degrees of clustering or the degree of clustering in an index is nonuniform among the key values.

In our test environment, we collected detailed and distribution statistics for the `LINEITEM` table with write access as shown in Example 3-71.

Example 3-71 Collecting detailed statistics

```
db2 "RUNSTATS ON TABLE db2inst1.lineitem WITH DISTRIBUTION AND DETAILED
INDEXES ALL ALLOW WRITE ACCESS"

DB20000I  The RUNSTATS command completed successfully.
```

Statistics in a partitioned environment

The DB2 optimizer uses statistics to estimate the size of the intermediate results and the costs in order to choose an optimum access plan. In a partitioned environment, the collection of statistics is done on a single partition. It is difficult to extrapolate certain statistics to the table level. If the table is very small compared to the number of partitions, the statistics might not reflect the whole table. It is possible that there are no rows on the partition where the statistics are collected. Small tables, particularly those with significant skew in the partition key values, might give the optimizer better information to use during planning if the tables are placed in a single partition.

Monitoring RUNSTATS

Monitor the RUNSTATS utility by using the LIST UTILITIES command as shown in Example 3-72.

Example 3-72 Monitoring the RUNSTATS utility

```
db2 LIST UTILITIES SHOW DETAIL

ID                                = 13
Type                              = RUNSTATS
Database Name                     = TESTDB
Partition Number                  = 1
Description                       = DB2INST1.LINEITEM
Start Time                        = 04/11/2007 11:25:29.175498
State                             = Executing
Invocation Type                   = User
Throttling:
   Priority                       = Unthrottled
```

3.5.2 Monitoring

In this section, we discuss monitoring a partitioned database environment by using DB2 tools.

Snapshot monitoring

You can use the DB2 snapshot monitor to capture information about the database and any connected applications at a specific point-in-time. The snapshot information is stored in internal buffers within DB2. Capture snapshots by using the GET SNAPSHOT command. The snapshot monitor collects a “snapshot” or current activities in the database and does not provide historical information. Snapshot information that is already collected can be cleared by using the RESET MONITOR command. Table 3-2 is a summary of the scope, information provided, and use of snapshot monitoring.

Table 3-2 Scope, information provided, and use of snapshot monitors

Snapshot level	Information provided	Usage
Instance	High-level view of the instance	► In DPF, FCM information ► Service level information
Database	High-level view of the database	► Buffer pool efficiency ► Sort heap sizing ► Lock contention ► Deadlock occurrence
Table space	Details about each table space, such as containers, used pages, and free pages.	► DMS space utilization ► Determine most active table spaces
Table	Details about the most active tables, such as rows read, rows written, and overflows	► Determine most active tables during peak periods in terms of scans and writes ► Reorganization status of table
Buffer pool	► Details of all of the defined buffer pools ► Logical and physical reads and writes ► Asynchronous reads and writes	► Calculating buffer pool efficiency for each buffer pool that is defined

Snapshot level	Information provided	Usage
Lock	Current locks held in the database	► Identifying holders of locks and applications waiting ► Identifying the objects locked, for example, tables or indexes ► Identifying the types of locks held, for example, exclusive or share locks
Statement	► SQL statement text of dynamic SQL ► Number of statement compilations and executions ► Rows read and written by the statement ► Buffer pool usage per statement	► Package cache efficiency ► Most active statements ► Indexing opportunities for statements doing large table scans
Application	Details about each application connected to the database, for example, application handle, status, and so forth	► Determine what an application is doing ► Determine most active applications ► Elapsed times of applications

Before most snapshot data can be collected at any of the levels, the default database monitor switches need to be turned on at the instance level. Even without the monitor switches on, you can collect certain monitor data by default.

Note: The DFT_MON_TIMESTAMP monitor switch is turned on by default.

In our test environment, we turned on all the monitor switches as shown in Example 3-73.

Example 3-73 Turning on all the monitor switches

```
db2 UPDATE DBM CFG USING DFT_MON_BUFPOOL ON
db2 UPDATE DBM CFG USING DFT_MON_LOCK ON
db2 UPDATE DBM CFG USING DFT_MON_SORT ON
db2 UPDATE DBM CFG USING DFT_MON_STMT ON
db2 UPDATE DBM CFG USING DFT_MON_UOW ON
db2 UPDATE DBM CFG USING DFT_MON_TABLE ON
```

In a DPF-enabled environment, you can use the GLOBAL keyword to collect aggregated snapshot data from all database partitions. The instance level snapshot in a DPF environment provides additional information, such as FCM information. You can capture a global database manager snapshot as shown in Example 3-74 on page 96.

Example 3-74 Collecting a global database manager snapshot

```
db2 GET SNAPSHOT FOR DATABASE MANAGER GLOBAL

            Database Manager Snapshot

Node type                                       = Enterprise Server
Edition with local and remote clients
Instance name                                   = db2inst1
Number of database partitions in DB2 instance   = 4
Database manager status                         = Active

Product name                                    = DB2 v9.1.0.2
Service level                                   = s070210 (U810940)
...
...
Node FCM information corresponds to             = 3
Free FCM buffers                                = 25060
Free FCM buffers low water mark                 = 25060
Free FCM channels                               = 12518
Free FCM channels low water mark                = 12518
Number of FCM nodes                             = 4
...
...
```

You can use the AT DBPARTITIONNUM keyword to collect data at a particular database partition as shown in Example 3-75.

Example 3-75 Collecting a snapshot at a particular database partition

```
db2 GET SNAPSHOT FOR LOCKS ON testdb AT DBPARTITIONNUM 3

Database Lock Snapshot

Database name                              = TESTDB
Database path                              =
/home/db2inst1/db2inst1/NODE0003/SQL00001/
Input database alias                       = TESTDB
Locks held                                 = 0
Applications currently connected           = 1
```

```
Agents currently waiting on locks          = 0
Snapshot timestamp                         = 04/11/2007 15:26:40.407832

Application handle                         = 63
Application ID                             = *N0.db2inst1.070411200328
Sequence number                            = 00001
Application name                           = db2bp
CONNECT Authorization ID                   = DB2INST1
Application status                         = Connect Completed
Status change time                         = 04/11/2007 15:03:40.932614
Application code page                      = 819
Locks held                                 = 0
Total wait time (ms)                       = 0
```

You reset snapshot data by using the RESET MONITOR command. In a DPF-enabled environment, the monitor can be reset at a particular database partition with the AT DBPARTITIONNUM keyword or on all database partitions with the GLOBAL keyword as shown in Example 3-76.

Example 3-76 Resetting the monitor switches

```
db2 "RESET MONITOR ALL GLOBAL"

DB20000I  The RESET MONITOR command completed successfully.
```

Event monitoring

You can use the DB2 event monitor to collect database monitor information on a continual basis, which differs from snapshots because the snapshots are taken at a certain point-in-time. DB2 event monitors are stored in the system catalog tables. Event monitor data can be collected to pipes, files, or tables. Event monitors collect data when a specific event, for which the event monitor has been set up, occurs.

Event monitors do not have any database configuration switches that need to be turned on before the collection of data. You must define and activate event monitors in order to use them. By default, the DB2DETAILDEADLOCK event monitor is defined. Event monitors that are defined can be viewed by querying the SYSCAT.EVENTMONITORS system catalog table.

To create an event monitor, use the CREATE EVENT MONITOR SQL statement. Event monitors only collect data when they are activated. To activate or deactivate an event monitor, use the SET EVENT MONITOR STATE SQL statement. A state of 0 indicates the event monitor is not activated. A state of 1 indicates the event monitor is activated.

In a DPF-enabled environment, you can define an event monitor at a specific database partition by using the DBPARTITIONNUM keyword, on a partition where the command is issued from by using the LOCAL keyword (this is the default), or on all database partitions by using the GLOBAL keyword.

The event monitor creates binary files. You need to use a utility called **db2evmon** to format the event monitor binary files to text as shown in Example 3-77. The **db2evmon** utility applies to event monitors that write to files or pipes.

Example 3-77 Creating, activating, and formatting the event monitor data

```
db2 CONNECT TO testdb

db2 "CREATE EVENT MONITOR testevmon FOR CONNECTIONS WRITE TO FILE
'/home/db2inst1/EVMON' ON DBPARTITIONNUM 2"

DB20000I  The SQL command completed successfully.

db2 "SET EVENT MONITOR testevmon STATE 1"

DB20000I  The SQL command completed successfully.

export DB2NODE=2
db2 terminate
db2evmon -path /home/db2inst1/EVMON
```

Table 3-3 provides a summary of the scope, the information provided, and the point at which the event data is collected by the event monitor.

Table 3-3 Event monitor scope, information provided, and data collection time

Event monitor	Information provided	Collected
Database	All database level counters	Database deactivation
Connections	All application level counters	End of connection
Table spaces	Counters for the buffer pool, prefetchers, page cleaners, and direct I/O for each table space	Database deactivation
Tables	Rows read and written for each table	Database deactivation
Buffer pools	Counters for the buffer pool, prefetchers, page cleaners, and direct I/O for each buffer pool	Database deactivation

Event monitor	Information provided	Collected
Deadlocks	Applications involved in the deadlocks and locks in contention	Detection of deadlock
Statements	► Statement start and stop time ► CPU consumed ► Dynamic SQL statement text ► In DPF, table queue information	► End of statement (Single partition) ► End of subsection (DPF)
Transactions	► Unit of work (UOW) start and stop ► CPU consumption ► Transaction logs used ► Locking information	End of unit of work

DB2 problem determination (db2pd) and monitoring

The utility, **db2pd**, is a stand-alone utility shipped with the DB2 product. Use **db2pd** for monitoring, as well as problem determination. You can execute **db2pd** in a command line mode or an interactive mode. The utility **db2pd** runs very quickly, because it does not acquire locks or latches and runs outside of the DB2 engine resources. You can even run **db2pd** on an engine that is stopped or appears to be in a loop. Using **db2pd** gives the user a closer view of the DB2 Engine.

Requirements

The requirements to use **db2pd** are:

- You must execute the utility on the same physical machine as the instance.
- You can use **db2_all**, **rsh**, and so forth to execute remotely.
- The user must have SYSADM authority.
- The user must be the instance owner (for UNIX and Linux only).

You can execute **db2pd** to collect data at an instance level or a database level. Instance scope options report information at the instance level. Database scope options report information at the database level. Using the -alldbs option, several databases can be active in one instance. All **db2pd** options have a full descriptive word, for example, -applications, which can be shortened to a three letter minimum, such as -app.

Controlling the scope of db2pd

You can use **db2pd** options to control the scope of the data that is collected:

- Use of the -alldbpartitionnums (-alldbp) option attaches to all database partitions on this physical machine.

- Use of the -dbpartitionnum (-dbp) <*num*>[,<*num*>] option attaches to a database partition (overrides the DB2NODE environment variable).
- Use of the -alldatabases (-alldbs) option attaches to all active databases.
- Use of the -database (-db) <*database*>,[<*database*>] attaches to the chosen databases.

The most common options for **db2pd** at the instance and database levels are shown in Table 3-4.

Table 3-4 db2pd scope and options

Scope	Options	Use
Instance	agents	► Detailed agent information, such as type, state, rows read, and rows written
	fcm	► FCM statistics ► Buffer consumption ► Applications involved in FCM usage
	dbmcfg	► Database manager configuration
	sysplex	► Used for DB2 Connect™ products showing remote connections
	utilities	► Utilities that are executing and states
	osinfo	► Detailed operating system (OS) information
Database	applications	► Details about applications, such as application handle, PID, current statement executing, and last statement executed
	transactions	► Transaction information, such as log space used, transaction handle, and so forth
	bufferpools	► Information for all bufferpools and hit ratios with monitor switch enabled
	logs	► Transaction log information, such as current active log, and so forth
	locks	► Detailed lock information
	table spaces	► Table space details and statistics
	dynamic	► Information about dynamic SQL stored in the dynamic cache
	static	► Static SQL package information
	dbcfg	► Displays the database configuration

Scope	Options	Use
	catalogcache	► Information from the catalog cache
	tcbstats	► Detailed table statistics and information
	reorgs	► Table reorg information, such as status
	recovery	► Recovery information, such as crash recovery and roll forward
	storagepaths	► Database storage path
	activestatements	► Information about currently active SQL statements

In our test environment, we submitted the **db2pd** command to list all the applications on database partition `1`, as shown in Example 3-78.

Example 3-78 Using db2pd to list all applications on database partition 1

```
db2pd -dbp 1 -app -db testdb

Database Partition 1 -- Database TESTDB -- Active -- Up 0 days 00:27:00

Applications:
Address            AppHandl [nod-index] NumAgents  CoorPid    Status
0x0780000000199200 162      [000-00162] 1          0          Unknown
```

We can use the -alldbp option of **db2pd** to display information about all database partitions. Example 3-79 shows how to get lock information.

Example 3-79 Using db2pd to list all locks at the database level

```
db2pd -alldbp -locks -db testdb

Database Partition 0 -- Database TESTDB -- Active -- Up 0 days 00:31:33

Locks:
Address            TranHdl    Lockname                   Type
Mode Sts Owner      Dur HoldCount  Att  ReleaseFlg

Database Partition 1 -- Database TESTDB -- Active -- Up 0 days 00:31:19

Locks:
Address            TranHdl    Lockname                   Type
Mode Sts Owner      Dur HoldCount  Att  ReleaseFlg
```

```
Database Partition 2 -- Database TESTDB -- Active -- Up 0 days 00:31:19

Locks:
Address            TranHdl    Lockname                   Type
Mode Sts Owner      Dur HoldCount  Att  ReleaseFlg

Database Partition 3 -- Database TESTDB -- Active -- Up 0 days 00:31:19

Locks:
Address            TranHdl    Lockname                   Type
Mode Sts Owner      Dur HoldCount  Att  ReleaseFlg
```

DB2 EXPLAIN

The DB2 EXPLAIN facility allows you to capture information about the environment and the access plan chosen by the optimizer for static or dynamic SQL statements. You can then use this information to tune the SQL statements, as well as the database manager configuration, to improve the performance of queries.

DB2 EXPLAIN captures:

- Sequence of operations to process the query
- Cost information
- Predicates and selectivity estimates for each predicate
- Statistics for all objects referenced in the SQL statement at the time that the EXPLAIN information is captured

The DB2 EXPLAIN facility provides a number of tools to capture, display, and analyze information about the access plans that the optimizer chooses for SQL statements. The access plan listed in the EXPLAIN output is based on the statistics available at the time of statement compilation. For static SQL, this corresponds to bind and preparation time and might not match the actual runtime statistics.

Access path information is stored in EXPLAIN tables, which you can query to retrieve the information that you want. You can use either the GUI tool Visual Explain or the text-based **db2exfmt** tool to examine the contents of the EXPLAIN tables. We only demonstrate the **db2exfmt** tool in our examples.

EXPLAIN tables can be created by issuing the **db2 -tvf EXPLAIN.DDL** command, or by the DB2 Control Center automatically. The EXPLAIN.DDL file is located in the $HOME/sqllib/misc directory on UNIX and Linux, where $HOME is the home directory of the DB2 instance owner and is located in C:\Program Files\IBM\SQLLIB\misc on Windows.

SQL statements need to be explained in order to populate the EXPLAIN tables. After the EXPLAIN tables have been populated, the following command is used to generate **db2exfmt** output for a SQL statement:

```
db2exfmt -d <dbname> -l -s <schema> -o <output file>
```

Analyzing query access plans in a partitioned environment

This section looks at characteristics of a plan that are relevant to a partitioned environment. In order to explain several of the concepts that we have discussed in the previous sections, we use an example based on the TPC-H schema shown in Example 3-80. This query has a join of six tables with an aggregation. The **db2exfmt** tool output that formats the information from the EXPLAIN tables is shown in Example 3-80, too. The number in parentheses beneath the operator is the operator number. The number above the operator is the cardinality relevant to the operation.

Example 3-80 Analyzing a query access plan

```
SELECT     nation, o_year, sum(amount) AS sum_profit
FROM       (
           SELECT     n_name AS nation, year(o_orderdate) AS o_year,
                      l_extendedprice * (1 - l_discount) - ps_supplycost *
                      l_quantity AS amount
           FROM       tpcd.part, tpcd.supplier, tpcd.lineitem, tpcd.partsupp,
                      tpcd.orders, tpcd.nation
           WHERE      s_suppkey = l_suppkey
                      AND ps_suppkey = l_suppkey
                      AND ps_partkey = l_partkey
                      AND p_partkey = l_partkey
                      AND o_orderkey = l_orderkey
                      AND s_nationkey = n_nationkey
                      AND p_name LIKE '%coral%'
           ) AS profit
GROUP BY   nation, o_year
ORDER BY   nation, o_year DESC

Access Plan:
-----------

        Total Cost:                     2.99008e+07
        Query Degree:                   1

                            Rows
                           RETURN
                           (   1)
                             |
                            225
                           GRPBY
```

```
                                 (   2)
                                   |
                                 36000
                                MDTQ
                                (   3)
                                   |
                                  225
                                GRPBY
                                (   4)
                                   |
                                  225
                                TBSCAN
                                (   5)
                                   |
                                  225
                                SORT
                                (   6)
                                   |
                             2.2704e+07
                                HSJOIN
                                (   7)
                     /-------------+-------------\
              2.2704e+07                        633503
                DTQ                             HSJOIN
                (   8)                          (  18)
                  |                         /------+-----\
              2.2704e+07                633503              25
                HSJOIN                  TBSCAN            BTQ
                (   9)                  (  19)            (  20)
              /----+----\                 |                 |
       5.07707e+07    2.45672e+07       633503              25
         TBSCAN         DTQ         TABLE: TPCD           FETCH
         (  10)         (  11)     SUPPLIER               (  21)
           |              |                              /----+----\
       5.07707e+07    2.45672e+07                     25            25
     TABLE: TPCD        HSJOIN                      IXSCAN     TABLE: TPCD
    PARTSUPP            (  12)                      (  22)    NATION
                   /-------+-------\                  |
            9.5221e+07           2.45672e+07         25
              TBSCAN               HSJOIN      INDEX: TPCD
              (  13)               (  14)     N_NK
                |                 /---+---\
            9.5221e+07     3.80893e+08  1.30986e+08
          TABLE: TPCD        TBSCAN       BTQ
         ORDERS              (  15)       (  16)
                                |            |
                            3.80893e+08    818664
                         TABLE: TPCD      TBSCAN
                        LINEITEM          (  17)
```

```
                                        |
                                  1.26927e+07
                                TABLE: TPCD
                               PART
```

The two biggest tables, LINEITEM and ORDERS, are partitioned on ORDERKEY. The query joins these tables on the ORDERKEY. The join (HSJOIN(12)) is collocated and this is probably good considering the size of data joined.

The PART table is partitioned on PARTKEY, but the LINEITEM table is partitioned on ORDERKEY. Even though you can direct the LINEITEM table to the PART table partitions, the optimizer does not want to move the rows of the LINEITEM table. Instead, after filtering some of the PART table rows, the optimizer chooses to broadcast (BTQ(16)) the PART table to be joined to the LINEITEM table. This join is done before the join to the ORDERS table, because it helps reduce the size of the LINEITEM table. Note that the partitioning of the result of this join is the same as that of the LINEITEM table. Note also that the cardinality has been increased from 818664 to 1.30986a+08 after broadcasting the rows to the 160 partitions of the LINEITEM table.

The join between PARTSUPP and LINEITEM is through the columns PARTKEY and SUPPKEY. These columns are also the partitioning keys by which the PARTSUPP table is partitioned. The optimizer chooses a directed table queue (DTQ(11)) to push each of the result rows containing the LINEITEM data to the corresponding partitions of the PARTSUPP table.

The NATION table is joined to the SUPPLIER table on the NATIONKEY. Because the SUPPLIER is partitioned on the SUPPKEY, this join (HSJOIN(18)) uses a broadcast table queue (BTQ(20). The resulting partitioning of this join is still the same as the SUPPLIER was partitioned with SUPPKEY as the partitioning column.

The final join (HSJOIN(7)) is chosen with a directed table queue (DTQ(8)) to send each row of the result of the join between LINEITEM, PART, ORDERS, and PARTSUPP to the corresponding partitions of the result of SUPPLIER and NATION. This is because the join predicate is on the SUPPKEY.

Finally at the top of the plan, after all the joins, we have a partial sort on the N_NATION and O_YEAR columns on each partition. This also helps the intermediate aggregation (GRPBY(4)) that collapses the size of the result on each partition. The final aggregation is done through the GRPBY(2) operator. The order required by the query is maintained by merging the rows from each partition through the merging directed table queue (MDTQ(3)) sent to the coordinator partition before the result is returned to the user.

3.5.3 Rebalancer

To maintain data striping across containers, DB2 might determine that a rebalance is necessary and kick off the rebalancer process. The rebalancer runs asynchronously in the background, and data in the table space is still accessible to applications during this process.

You can add space to a DMS table space by using the ADD, ADD TO STRIPE SET, EXTEND, and RESIZE options of the ALTER TABLESPACE statement. BEGIN NEW STRIPE SET is another option for adding space, but because it does not result in a rebalance, we do not include it in this discussion.

You can remove space from a DMS table space by using the DROP, REDUCE, and RESIZE options of the ALTER TABLESPACE statement.

Table space map

Each DMS table space has a “map” that describes how containers are positioned within the logical address space of the table space. This table space map is used to convert table space-relative page numbers to physical disk locations. During the rebalance, extents are physically moved from one location on disk to another location and the internal table space map is updated to point to the new spot.

All of the examples in this section use either a pictorial representation, a list of ranges (table space map), or a combination of both to describe container configurations.

In the pictorial representation:

- Each container is shown as a vertical bar of extents and the order of the extents is determined by the container IDs (0 to N, with 0 on the left).
- Extent numbers might be listed in several of the examples, but they are not necessary after you learn the basics.
- A *stripe* is a horizontal cross section of the containers, and a stripe is one extent thick. There is always at least one stripe set in a table space and it is stripe set #0.
- Consecutive stripes that have the identical container configurations (that is, the same number of containers and the exact same container IDs within it) make up a range. There is always at least one range in a table space, because you must have at least one container.

The following fields exist for each range:

- *Stripe set* is the number of the stripe set in which the range exists.
- *Stripe set offset* is the stripe in which the range’s stripe set starts.
- *Maximum extent* is the highest extent mapped by the range.

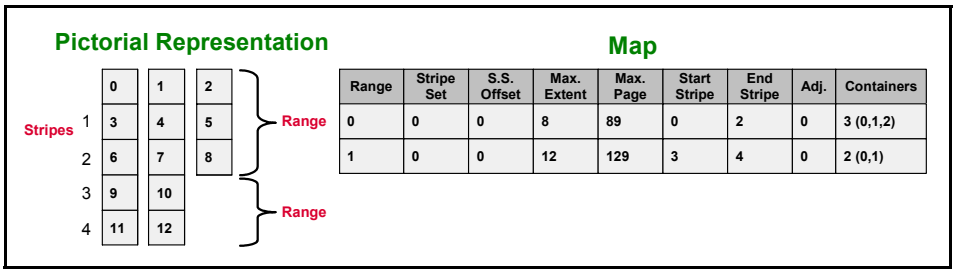
- *Maximum page* is the highest page mapped by the range.
- *Start stripe* is the stripe in which the range starts.
- *Stop stripe* is the stripe in which the range ends.
- *Adjustment* is a value used during a rebalance (more about this value later).
- *Containers* is the list of containers that participate in the range.

In Example 3-81, we create a table space that we use to explain rebalancing concepts.

Example 3-81 Creating a table space with three containers

```
CREATE TABLESPACE ts1 MANAGED BY DATABASE USING
  (FILE 'cont0' 51, FILE 'cont1' 51, FILE 'cont2' 31) EXTENTSIZE 10
```

After removing the page or pages from each container for the tag and splitting the remaining pages into extents, the three containers have five, five, and three extents, respectively (13 total). The resulting container configuration can be represented by either a pictorial representation or a map as shown in Figure 3-11.



Range	Stripe Set	S.S. Offset	Max. Extent	Max. Page	Start Stripe	End Stripe	Adj.	Containers
0	0	0	8	89	0	2	0	3 (0,1,2)
1	0	0	12	129	3	4	0	2 (0,1)

Figure 3-11 Pictorial representation of containers and a table space map

In Figure 3-11, there are three containers with sizes of 51, 51, and 31 pages respectively. Each container has a container tag, and by default, this uses a single page, leaving 50, 50, and 30 pages. The extent size is 10, and therefore, there are 5, 5, and 3 extents.

Container IDs are assigned in the order in which they are listed in the CREATE TABLESPACE statement.

In Figure 3-11, the three containers are shown as vertical bars of 5, 5, and 3 boxes (representing the number of extents in them).

When a table space is created, it is placed into this map so that striping starts by using all of the containers (that is, all start in stripe 0).

You can see by the extent numbers in the boxes how DB2 stripes across these containers. It starts in the first stripe and uses an extent in each of the containers within the stripe. After it has striped across one stripe, it moves onto the next one.

In this example, striping continues across all of the containers until container 2 is full. When this happens, the striping is done across the remaining containers.

Because stripes 0 - 2 share the same containers, it is considered a range. Stripes 3 - 4 share the same containers so they are also a range.

There are 9 extents in the first range, (0 - 8), which means that there are 90 pages in the first range, (0 - 89). These maximum values are stored in the map along with the starting stripe (0), end stripe (2), and the containers (3 of them: 0, 1, and 2).

In the second range, there are 4 extents, 9 - 12 (pages 90 - 129). The starting stripe is 3, the end stripe is 4, and there are 2 containers (0 and 1).

The adjustment value is an indication of how far into the range that the extents start. It is non-0 only during a rebalance.

Table space map: Adding containers

When a table space is created, the starting points for all of the containers are in stripe 0 (regardless of their sizes). This is not true when new containers are added to an existing table space.

When a table space is created, its map is created and all of the given containers are lined up so that they all start in stripe 0. This means that data is striped evenly across all of the table space containers until the individual containers start filling up.

When containers are added to an existing table space, they might be added so that they do not start in stripe 0. Where they start in the map is determined by DB2 and is based on the size of the containers added.

If the container is large enough so that it can start in stripe 0 and end at or beyond the last stripe in the map, that is how it is positioned.

If it is not large enough to do this, it is placed so that it ends in the last stripe of the map. This means that it does not start in stripe 0. This is done to minimize or avoid rebalancing. We discuss more information about determining when a rebalance is needed in the “Rebalancing” on page 114.

Using the table space created in Example 3-81 on page 107, Figure 3-12 on page 109 shows three examples of adding one new container to it.

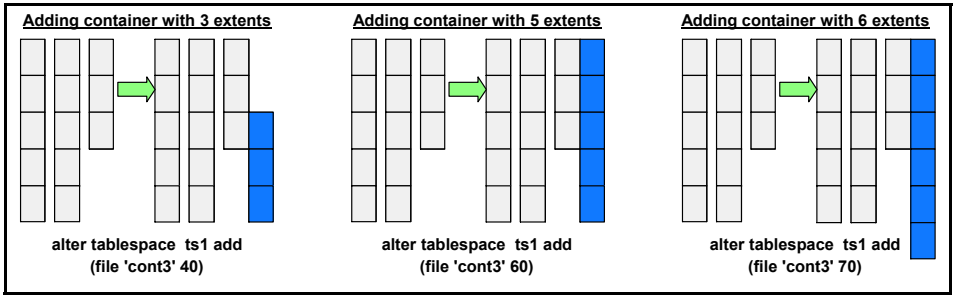


Figure 3-12 Adding containers to a table space

In the first example in Figure 3-12, a container with 3 extents is added (note that the container is actually 4 extents in size, but one extent is used to hold the container tag). If the container was placed so that it started in stripe 0, it ends in stripe 2. Stripe 2 is not the last stripe in the map, so DB2 decides not to place it this way. Instead, it is placed so that it starts in stripe 2 and ends in the last stripe (stripe 4).

In the other two examples in Figure 3-12, containers are added with 5 and 6 extents. In these cases, they can be added so that they start in stripe 0 and stop in the last stripe (4, as in the second example) or beyond it (5, as in the third example).

Table space map: Extending containers

When extending containers, the new extents are added to the end of the container in the map as shown in Figure 3-13 on page 110.

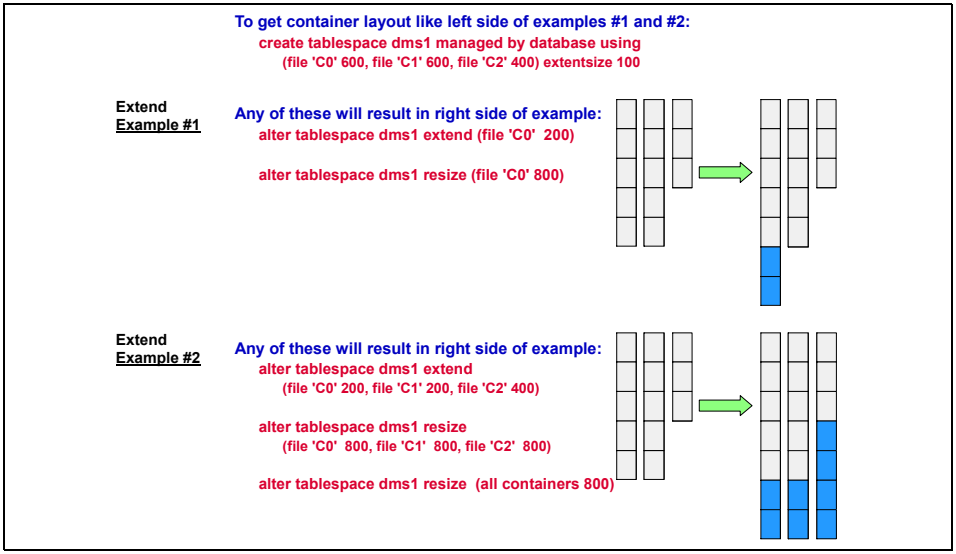


Figure 3-13 Extending containers

You can extend existing table space containers by using the EXTEND or RESIZE options on the ALTER TABLESPACE statement.

When new space is added to an existing container, the map is changed so that the space is just tacked on to the end of the container. In other words, the start stripe for that particular container does not change but the end stripe for it does. The end stripe increases by the number of extents added to it. Note that this can still cause a rebalance to occur.

We provide all of the scenarios and examples in this section only to show you how DB2 extends containers. The examples do not mean that these are good configurations (in fact, they are not good configurations and are quite impractical).

For examples #1 and #2 in Figure 3-13, the table space is created with three file containers and the extent size is 100. The containers are 600, 600, and 400 pages in size but after removing an extent of pages from each for the tag, this leaves 500, 500, and 300 pages (5, 5, and 3 extents).

Example #1 in Figure 3-13 increases the size of the first container by two extents (200 pages). You can do this by using EXTEND and specifying the size difference, or you can do this by using RESIZE and specifying the new size (800 in this case, 700 pages plus 100 for the tag).

Example #2 in Figure 3-13 on page 110 increases the size of each of the containers to seven extents. Using EXTEND, each container can be listed along with the number of pages that it needs to grow. Using RESIZE, each container can be listed along with the new size (800, which is 700 pages plus 100 for the tag). Because all of the containers in the table space are resized to the same size (800 pages), you can use the ALL CONTAINERS clause instead of listing them individually.

Table space map: Reducing containers

When reducing containers, the extents at the end of the container are removed as shown in Figure 3-14.

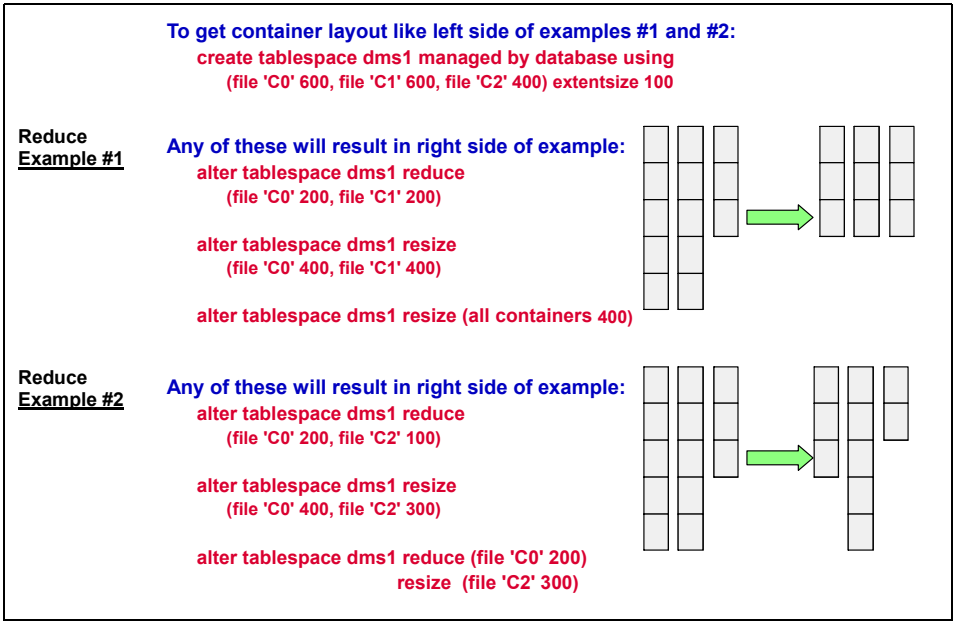


Figure 3-14 Reducing containers

When space is removed from an existing container, the map is changed so that the space is removed from the end of the container. In other words, the start stripe for that particular container does not change, but the end stripe for that container does. The end stripe decreases by the number of extents removed from it. Note that this can still cause a rebalance to occur.

We use all of the scenarios and examples in this section only for the purpose of showing how DB2 reduces containers. These examples are not meant to imply that these are good configurations, in fact, they are not good configurations and are quite impractical.

For examples #1 and #2 in Figure 3-14 on page 111, the table space is created with three file containers and the extent size is 100. The containers are 600, 600, and 400 pages in size but after removing an extent of pages from each for the container tag, this leaves 500, 500, and 300 pages (5, 5, and 3 extents).

Example #1 in Figure 3-14 on page 111 decreases the size of the first two containers by two extents (200 pages). You can do this by using REDUCE and specifying the size difference or by using RESIZE and specifying the new size (400 in this case, 300 pages plus one for the tag).

Example #2 in Figure 3-14 on page 111 decreases the size of the first container by two extents (200 pages) and decreases the size of the second container by one extent (100). As shown, you can do this by using the REDUCE option, RESIZE option, or a combination of both.

Table space map: Dropping containers

When dropping containers, the containers are dropped from the map as shown in Figure 3-15.

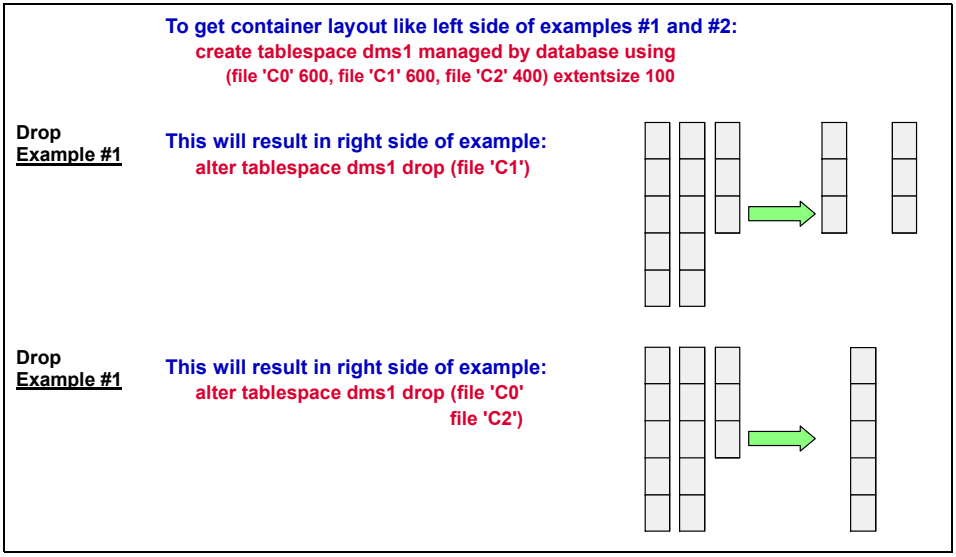


Figure 3-15 Dropping containers

When dropping one or more containers, a rebalance likely happens (this depends on the current container configuration and the amount of data in the table space, we discuss this more in upcoming sections). The rebalance that occurs when space is removed from a table space is called a *reverse rebalance*.

After containers have been dropped and the reverse rebalance completes, the remaining containers are renumbered so that their container IDs are contiguous, starting at 0.

For example, if there are five containers in a table space, they have container IDs in the range of 0 - 4. If containers 0, 1, and 3 were dropped, this leaves containers 2 and 4. After the reverse rebalance completes, the actual files and devices associated with containers 0, 1, and 3 are released, and then containers 2 and 4 are renumbered to 0 and 1.

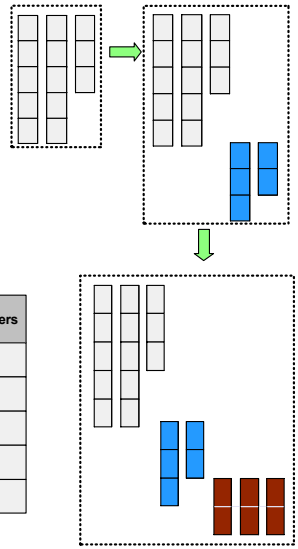
Table space map: Stripe sets

When a new stripe set is created in a table space, the new containers that are added are appended to the existing map. Each of the new containers is positioned so that it starts in the same stripe, regardless of its size as shown in Figure 3-16.

create tablespace ts1 managed by database using
(file 'C0' 30, file 'C1' 30, file 'C2' 20) extentsize 5

alter tablespace ts1 begin new stripe set
(file 'C3' 20, file 'C4' 15)

alter tablespace ts1 begin new stripe set
(file 'C5' 15, file 'C6' 15, file, 'C7' 15)



Range	Stripe Set	S.S. Offset	Max. Extent	Max. Page	Start Stripe	End Stripe	Adj.	Containers
0	0	0	8	44	0	2	0	3 (0,1,2)
1	0	0	12	64	3	4	0	2 (0,1)
2	1	5	16	84	5	6	0	2 (3,4)
3	1	5	17	89	7	7	0	1 (3)
4	2	8	23	119	8	9	0	3 (5,6,7)

Figure 3-16 Stripe sets

When creating a new stripe set using the BEGIN NEW STRIPE SET option of the ALTER TABLESPACE statement, the new containers are added to the map so that they do not share the same stripe with any other containers. In this way, data rebalancing is not necessary and the rebalancer does not start.

In this example, a table space is created with three containers that are 5, 5, and 3 extents respectively (remember that an extent of pages (5 in this example) in each container is used to hold the container tag).

When the first new stripe set is created (stripe set 1), the containers that are added (C3 and C4) are positioned in the map so that their stripe start value (5) is one greater than the stripe end value of the last range in the map (4). After doing this, the last stripe in the map is stripe 7.

When the second new stripe set is created (stripe set 2), the containers that are added (C5, C6, and C7) are positioned in the map so that their stripe start value (8) is one greater than the last stripe in the map (7).

As objects in the table space grow, space is consumed so that stripe 0 is used first, followed by stripe 1, stripe 2, and so forth. This means that a stripe set is only used after all of the space in the previous stripe sets has been consumed.

Rebalancing

When you add or remove space from a table space, a rebalance might occur (depending on the current map and the amount of data in the table space). When space is added and a rebalance is necessary, a forward rebalance is started (extents move starting from the beginning of the table space).

When space is removed, a rebalance is necessary, and a reverse rebalance is started (extents move starting with extents at the end of the table space).

When new space is added to a table space, or existing space is removed, a new map is created. At this point, all of the extents still reside in physical locations on the disk determined by the existing current map. It is the job of the rebalancer to physically move extents from the original location (based on the current map) to the new location (based on the new map) as shown in Figure 3-17.

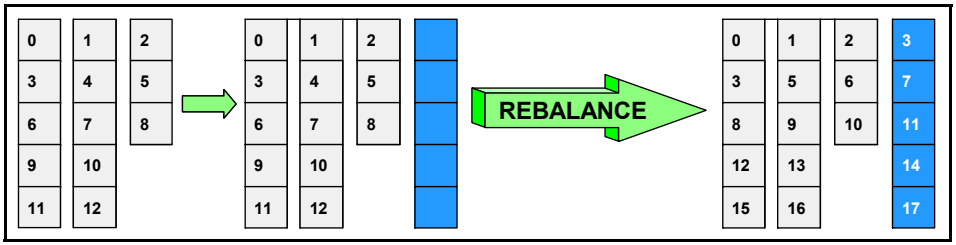


Figure 3-17 Table space map after a rebalance

Forward rebalance

A *forward rebalance* starts with the first extent in the table space (extent 0), moving one extent at a time until the extent holding the high-water mark has been moved.

As each extent gets moved, the current map is updated so that it knows to look for the extent in the new spot rather than the old one. As a result, the current map begins to look like the new map.

Because the high-water mark is the stopping point for the rebalancer, by the time that all of the extents have been moved, the current map and the new map are identical up to the stripe holding the high-water mark.

Because there are no actual data extents above the high-water mark, nothing else needs to move and the current map is made to look completely like the new map.

Data is still accessible while a rebalance is in progress. Other than the performance impact of the rebalance, objects can be dropped, created, populated, and queried as if nothing were happening.

Even empty extents are moved during a rebalance, because the rebalancer does not know whether extents are empty or in use. This is done for a couple of reasons. First, to know about free and in use extents, Space Map Pages (SMPs) must be “locked” in the buffer pool, and changes to SMPs cannot occur while the SMPs are scanned. This means that objects cannot be created or dropped while the corresponding SMPs are locked. Second, new space is usually added to a table space when there is little or no free space in it. Therefore, there need to be few free extents and the overhead of moving them unnecessarily is quite small relative to the work done for the rest of the rebalance.

For example, a table space has two containers that are each four extents in size. A third container of the same size is then added. The high-water mark is within the 6th extent (extent #5) as shown in Figure 3-18 on page 116.

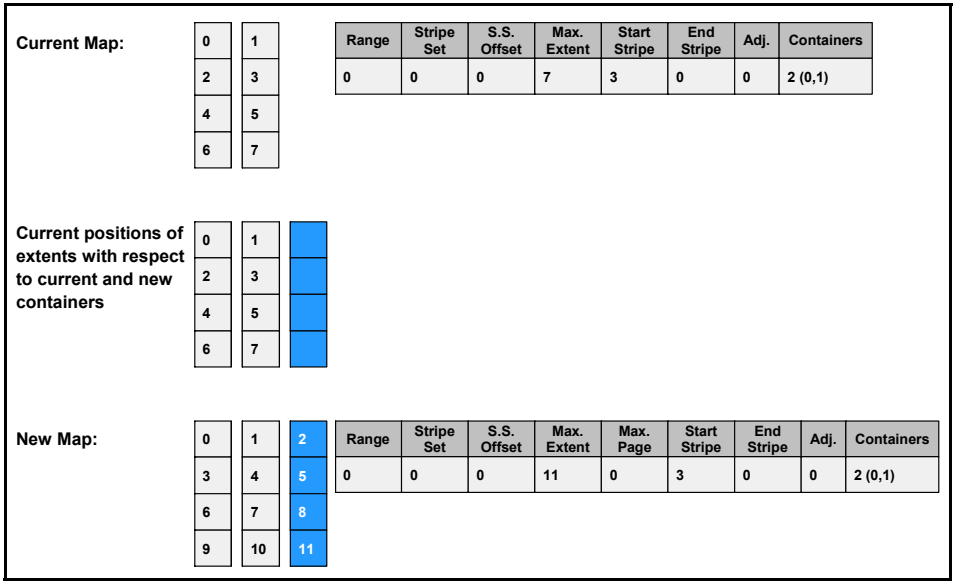


Figure 3-18 Forward rebalance

The current map is the table space map that describes the container configuration before adding the new container.

The new map is what the current map looks like after the rebalance is complete.

Note the following in this example in Figure 3-18:

- The container tag is not mentioned or taken into consideration. The sizes of the containers are given in “usable extents” (in other words, the number of extents that DB2 can use to actually hold data).
- The extent size has not been given (because it does not add anything to the example). Therefore, the Max. Page field is not shown.

Reverse rebalance

If the space that is removed (whether it is full containers dropped or extents removed as part of a reduction) all is *after* the high-water mark extent in the table space map, a reverse rebalance is unnecessary. Because none of the extents removed contain any data, nothing needs to be moved out of the extents into the remaining extents.

A reverse rebalance starts with the high-water mark extent, moving one extent at a time until the first extent in the table space (extent 0) has been moved.

As each extent gets moved, the current map is updated so that it knows to look for the extent in the new location rather than the old location. As a result, the current map begins to look more and more similar to the new map.

Data is still accessible while a rebalance is in progress. Other than the performance impact of the rebalance, objects can be dropped, created, populated, and queried as if nothing were happening.

Even empty extents are moved during a rebalance, because the rebalancer does not know whether extents are empty or in use. This is done for the following reasons. First, to know about free and in use extents, SMP pages must be “locked” in the buffer pool and changes to them cannot occur while the SMPs are scanned. This means that objects cannot be created or dropped while the corresponding SMPs are locked. Second, new space is usually added to a table space when there is little or no free space in it. Therefore, there must be few free extents and the overhead of moving the free extents unnecessarily is quite small relative to the work done for the rest of the rebalance.

Space availability

If the map is altered in such a way that all of the space comes after the high-water mark, a rebalance is unnecessary and all of the space is available immediately for use as shown in Figure 3-19.

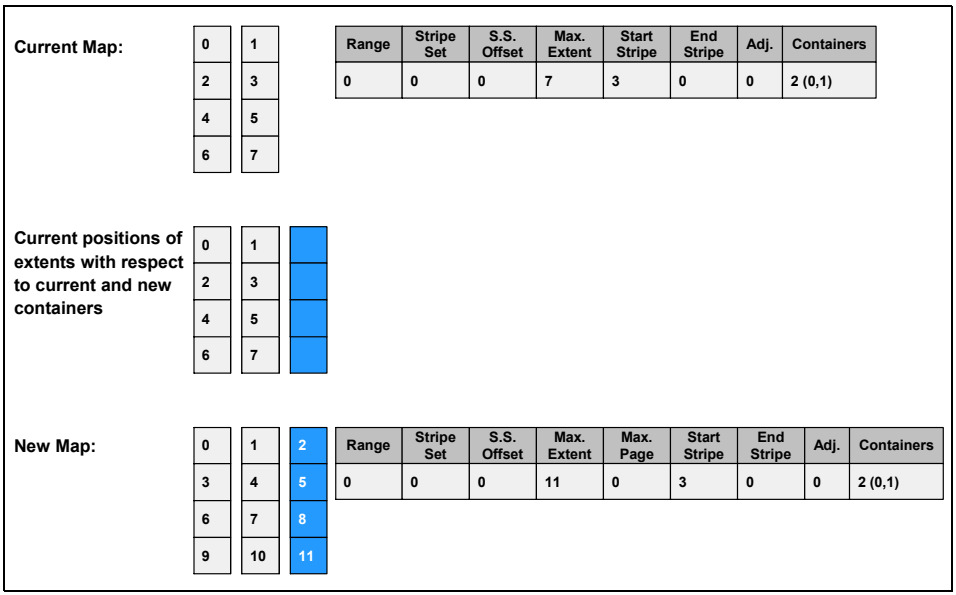


Figure 3-19 Space after the table space high-water mark

If the map is altered so that part of the space comes after the high-water mark as shown in Figure 3-20, the space in the stripes above the high-water mark becomes available for use. The rest of the space is unavailable until the rebalance is finished.

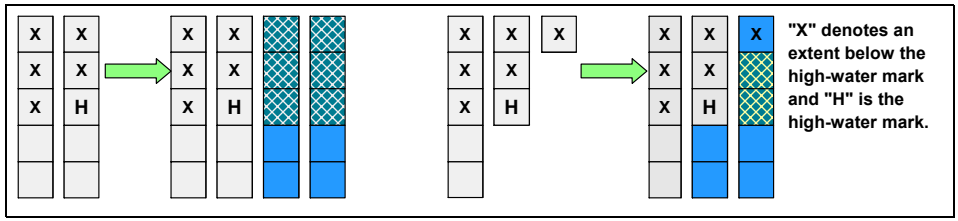


Figure 3-20 Space after the high-water mark is available for use

By the same logic, if all of the space is added to the map so that the space is in the high-water stripe or lower, none of the space becomes available until after the rebalance is complete, as shown in Figure 3-21.

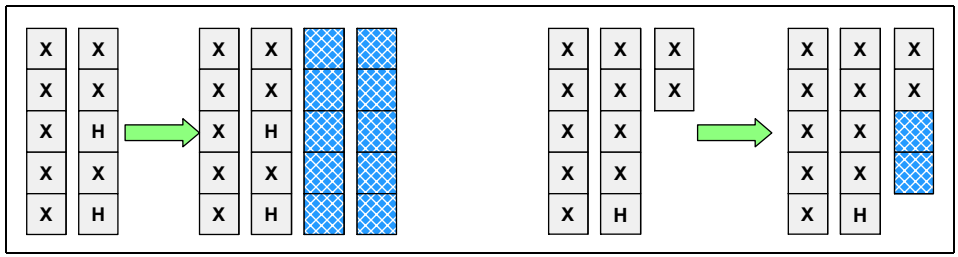


Figure 3-21 Space is not available until rebalanced

Tips for creating table spaces and adding space

We recommend that you:

- Create table spaces with equally sized containers.
- Create a table space large enough for your current needs, as well as your potential future growth (you might need to balance the cost of currently unused space with the ease of administration).
- If possible, extend containers instead of adding them (less chance of rebalancing).
- When adding new containers, consider growth and try to plan for the future by possibly adding more containers than immediately needed (to save on rebalances).
- If you are satisfied with the striping that occurs across the existing containers, create a new stripe set, which guarantees that a rebalance does not occur.

3.6 Using Materialized Query Tables to speed up performance in a DPF environment

Materialized Query Tables (MQTs) offer a way to speed up response times on well-known and repeatable queries against large volumes of data, where results are often expressed as summaries or aggregates. This section describes the MQT.

3.6.1 An overview of MQTs

An MQT is a real (materialized) table built from the result set of a query. Like any other table, an MQT can have indexes. Use RUNSTATS to collect and store table statistics. MQTs primarily exist for performance. Properly selected, MQTs perform a set of calculations that can then be used over and over by subsequent queries, but without going through the same operations again and again. The qualifying rows do not need to be fetched again, saving input time. Also, the calculations do not need to be redone, which saves CPU time.

3.6.2 When to consider a MQT

The design of good materialized query tables requires adequate up-front planning and analysis. You need to be familiar with the query workload to identify patterns for accessing tables and frequently performed aggregation and summarization. When deciding whether to create a materialized query table, consider:

- Does the MQT significantly increase performance?
- Do many queries benefit? Do the most frequent, most critical, or most expensive and longest running queries benefit?
- Does the MQT offer resource savings: communication, I/O, and CPU?
- Is the loss of disk space that contains the MQT and its indexes a worthwhile trade for the performance gained?
- What is the cost of updating or refreshing the MQT?
- What are the patterns for accessing groups of tables, for aggregation, and for grouping requirements?
- How current does the data in the MQT need to be? Does it need to be up to the minute?
- For MQTs that are maintained in real time, are automatic updates too slow?
- In a partitioned environment, does collocation of data provide any benefit?

- What is the logging requirement when large MQTs are refreshed?
- Does the MQT need to be system-maintained or user-maintained?

3.6.3 When to use the MQT

This section discusses the two portions of the SQL compiler that determine whether to use an MQT. First, the query rewrite portion must be able to determine that an MQT exists and that it can be used to satisfy the current query. Second, the optimizer examines the costs of using the MQT and makes the final determination to use or not use the MQT.

For the optimizer to consider using an MQT in the access plan, certain conditions must be met:

- For REFRESH DEFERRED MQTs, the CURRENT REFRESH AGE special register must be set to ANY. This setting informs the optimizer that any version of the MQT can be used to determine the answer to the query. REFRESH IMMEDIATE MQTs are always current and are candidates for optimization regardless of the setting of the CURENT REFRESH AGE register.
- The MQT must not be in REFRESH PENDING mode or CHECK PENDING NO ACCESS state.
- The optimization level must be equal to 2, or greater than or equal to 5. The default value for the query optimization level is 5. It can be changed by updating the database configuration parameter DFT_QUERYOPT. You can also set the value as shown in Example 3-82.

  Example 3-82 Set the current query optimization level

  ```
  -- Valid values are 0, 1, 2, 3, 5, 7, and 9
  SET CURRENT QUERY OPTIMATION 7
  ```

- The special register CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION must be set for the appropriate type of MQT to be considered. Acceptable settings are ALL, NONE, SYSTEM, FEDERATED_TOOL, or USER. If set to SYSTEM, user-maintained MQTs are not used for optimizing. Example 3-83 shows how to set this register to the value SYSTEM.

  Example 3-83 Set the current maintained table types register

  ```
  SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION = SYSTEM
  ```

- The isolation level currently in effect must be equal to or lower than the optimization level that was in effect when the MQT was created.

3.6.4 Intra-database replicated tables and partitioning

In the partitioned database environment, the performance of joins increases dramatically when the rows of the separate tables in the join are collocated, and we can avoid shipping data between partitions.

Figure 3-22 describes an environment with a collocated join between `CUST` and `ORDERS` tables, which have been partitioned on `CUST_ID` column.

However, when the `REGION` table is also involved in the join, then a collocated join is not possible, because the `REGION` table does not have a `CUST_ID` column, and therefore, cannot be partitioned by `CUST_ID`. DB2 can choose to perform a directed join in this particular case, but the performance of directed joins is less efficient than a collocated join, because the movement of rows is inline with query execution.

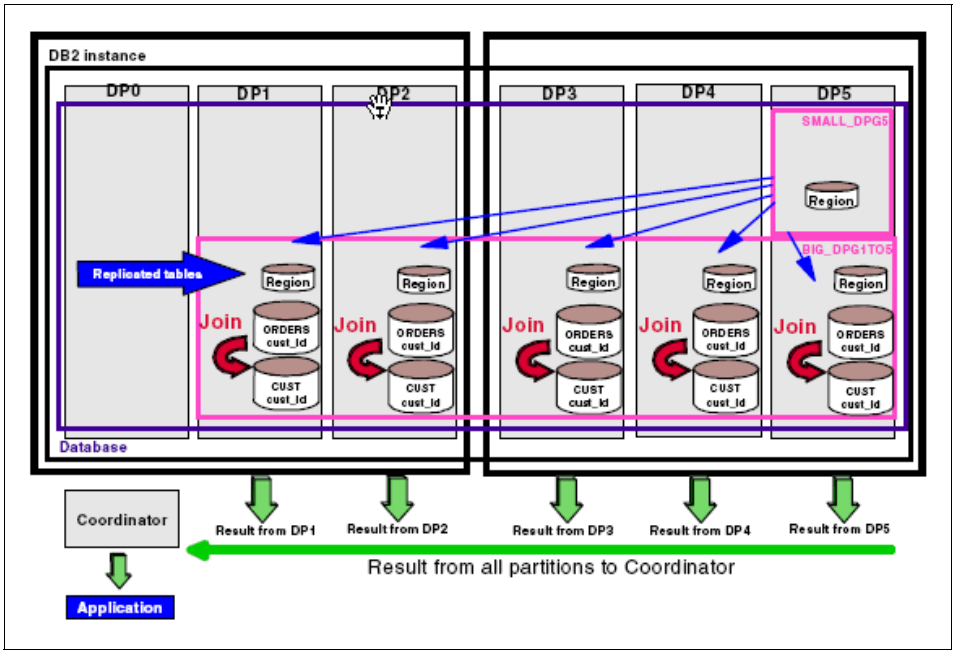


Figure 3-22 Replicated MQT

You can use MQTs to replicate tables to other database partitions to enable collocated joins to occur even though all the tables are not joined on the partitioned key. In Figure 3-22, `REGION` is replicated to the other partitions using the MQT infrastructure in order to enable collocated joins for superior performance.

Example 3-84 shows how to create a replicated table. Table REGION is stored in a single-partition table space. In order to facilitate a collocated join with a table in the multipartition table space TBSP123, we create the replicated MQT REGION_MQTR. With the REPLICATED clause, we requested that it is duplicated on all the partitions in the partition group over which table space TBSP123 is defined.

Example 3-84 Creating a replicated MQT

```
CREATE TABLE db2inst1.region_mqtr AS
   (SELECT * from db2inst1.region)
   DATA INITIALLY DEFERRED REFRESH IMMEDIATE
   REPLICATED IN tbsp123
DB20000I  The SQL command completed successfully.

REFRESH TABLE db2inst1.region_mqtr
DB20000I The SQL command completed successfully.

REFRESH TABLE db1inst1.region_mqtr INCREMENTAL
DB20000I The SQL command completed successfully.
```

3.7 Best practices

In this section, we look at several of our best practices recommendations for setting up your DPF environment.

3.7.1 Selecting the number of partitions

Selecting the optimum number of partitions for your partitioned database can be difficult. However, we recommend having as few partitions per server as possible. Whenever possible, have at least one partition per CPU, although having two to four CPUs per partition typically works best. We also recommend using 64-bit instances, which allow greater addressability, enabling you to have larger buffer pools.

Coordinator partition

In a DPF environment, any partition can be a coordinator partition. In fact more than one partition can service requests from your users or applications. However, we recommend that you dedicate one partition in your environment to be the coordinator partition and that this partition is only used as the coordinator partition and does not hold any data. When setting up your application connection, we recommend that you use the SET CLIENT command and specify

the coordinator node in the command. Example 3-85 shows the usage of the command to connect to a specific partition.

Example 3-85 SET CLIENT command

```
$ db2 set client connect_dbpartitionnum 3
DB20000I  The SET CLIENT command completed successfully.

$ db2 connect to testdb

   Database Connection Information

 Database server        = DB2/AIX64 9.1.2
 SQL authorization ID   = DB2INST1
 Local database alias   = TESTDB

$ db2 "values(current dbpartitionnum)"

1
-----------
          3
1 record(s) selected.
```

Catalog partition

The *catalog partition* is the database partition that stores the system catalogs for the partitioned database. We recommend separating the catalog partition from the data partitions.

3.7.2 Distribution key selection

Correct distribution key selection is essential to help you ensure that data is distributed evenly across partitions. This in turn can help with workload distribution. You need to take into account the join strategies used, the type of tables that are accessed, and the types of queries issued against the tables.

When choosing the distribution key, try to include columns that are frequently used for joins to increase the number of collocated joins. Include columns that are often used in a GROUP BY clause. Choose a column with a high number of distinct values. If the column has a limited number of distinct values, extremely few hashing numbers are generated. This increases the possibility of data skew and, therefore, a workload imbalance. Having a partitioning key that consists of four or more columns can cause performance issues because of the number of hashing values that are generated. Always choose your own distribution key rather than letting DB2 use the default distribution key if one is not specified.

Consider that integer columns are more efficient than character columns, which in turn are more efficient than using decimal columns. The partitioning key must be a subset of the primary key or unique index.

Remember that after you select a distribution key for a table, you cannot alter the key unless the table is in a table space that is associated with a single-partition database partition group.

Note: Creation of a multiple-partition table that contains only long data types (LONG VARCHAR, LONG VARGRAPHIC, BLOB, CLOB, or DBCLOB) is not supported.

3.7.3 Collocation

Tables that are collocated are stored in the same database partition group that allows the processing of a query within the same logical partition. This avoids unnecessary movement of data over the partitions in your DPF environment. To help ensure table collocation, use the join columns as partitioning keys. The joined tables can then be placed in table spaces that share the same partition group. The joined tables' partitioning keys must have the same number of columns and corresponding data types. Try to place small tables in single-partition database partition groups, except when you want to take advantage of collocation with a larger table. Also, try to avoid extending medium-sized tables across too many database partitions. Performance might suffer if a mid-sized table is spread across too many partitions.

4

Table partitioning

This chapter provides information about the implementation of table partitioning. This applies to UNIX, Linux, and Windows operating systems.

© Copyright IBM Corp. 2007. All rights reserved.

4.1 Planning considerations

This section discusses strategies and the partitioning selection.

4.1.1 Roll-in and roll-out strategies

We advise that you consider the roll-in and roll-out strategies in the database and table design. The roll-in and roll-out strategies determine the table partition granularity and layout. *Roll-in* refers to adding a new partition or attaching a new partition to a partitioned table. *Roll-out* refers to detaching a partition from a partitioned table.

You can use roll-in to:

- Add space for new data.
- Add data that is in an existing table (provided the table characteristics match the partitioned table).
- Re-add a partition that has been previously detached.

You can use roll-out to:

- Remove data from the partitioned table to an archive.
- Remove data from a partitioned table and delete.
- Temporarily remove the partition to a stand-alone table for maintenance, such as a REORG.

Only one partition can be added, attached, or detached in a single ALTER TABLE statement. If you plan to roll-in and roll-out multiple partitions, you must execute multiple ALTER ATTACH and ALTER DETACH statements. It is a good practice to put all the ATTACH or DETACH statements in the same transaction to avoid exclusively locking the entire table multiple times. The SET INTEGRITY executed after the ATTACH transaction can take care of all the new data in a single pass.

Roll-in strategies

Roll-in strategies include the use of the ADD PARTITION and ATTACH PARTITION options of the ALTER TABLE statement.

ADD PARTITION

The strategy for using ADD PARTITION is to create a new partition in the partitioned table and INSERT or LOAD data directly to the partitioned table, provided that it complies with the overall range of the altered table. Using ADD

PARTITION with LOAD minimizes logging, but table access is limited to, at the most, ready only.

We provide examples of the use of ADD PARTITION later in this chapter.

ATTACH PARTITION

Alternatively, you can use the ATTACH PARTITION parameter of ALTER TABLE. This allows you to attach an existing table that you have already loaded with data. This method provides minimal disruption when accessing a table.

When you perform an ATTACH, you must follow with SET INTEGRITY. This incorporates the data into the indexes on the table. At the same time, it also validates that the data is all within the boundaries defined for that range. SET INTEGRITY has been made online in DB2 9 so that this longer running portion of roll-in can take place while applications continue to read and write the existing data in the table.

Be aware that the online SET INTEGRITY is a single statement and runs as a single transaction. If the attached data volume is large, the index maintenance can potentially need a large amount of log space for recovery. An alternative roll-in solution is by using ALTER ADD to add a partition and use LOAD to move the data into the partition.

We provide examples of the use of ATTACH PARTITION later in this chapter.

Roll-out strategies

The roll-out strategy is to use the DETACH PARTITION option of the ALTER TABLE statement.

DETACH PARTITION

Use the DETACH PARTITION option to DETACH a partition to a stand-alone table. You can then perform actions as required, such as REORG, EXPORT data, DROP or prune data, and re-ATTACH.

We provide examples of the use of DETACH PARTITION later in this chapter.

Note that the new table resulting from the detached partition resides in the same table space as the original partition. If exclusive use of that table space is required for the partitioned table space, you have to DROP and re-CREATE the table elsewhere.

4.1.2 Range selection

The value of the partitioning key columns determines how the data is divided into ranges or data partitions. Determining the range is one of the essential elements of table partitioning. *Range selection* defines the key column, how many partitions to have in the table, the volume of data to be rolled-out, and limits to the data that the table holds.

Your particular selection for ranges depends on what you want to achieve by partitioning. If the partitioning is to achieve a date or time roll-in and roll-out, the range column is a date column, a time column, or a combination of both. If you are more interested in a performance solution by separating data in a table to physical objects (disk drives), you can choose a column, such as region, state, or county and define the resulting partitions to the physical database objects by separating the partitions to individual table spaces.

You need to define the range's granularity to match the data roll-out strategy. In reality, most data trickles in, which makes it hard to match the range with roll-in size. You have flexibility in how you utilize table partitioning. The basic principle is partitioning on a column that provides advantages in partition elimination and also makes the data management task (roll-out and roll-in) easier.

Range selection can be done by using automatically generated ranges where an overall range is divided equally, a manually generated range where the individual range for each partition is specified, or a combination of both.

4.1.3 Handling large objects

Large objects can be contained within the data table spaces specified in the minimum specification of the CREATE TABLE statement or you can use the option to separate them to their own table spaces. There are various ways of dealing with large objects in separate table spaces, which we discuss with examples in this chapter.

4.1.4 Indexing partitioned tables

As with indexes on non-partitioned tables, each index contains pointers to rows in all the data partitions of the table. However, an important difference is that each index on a partitioned table is an independent object.

Because an index on a partitioned table can act independently of other indexes, special considerations are needed with respect to which table space to use when creating an index on a partitioned table:

- Even though the table's data partitions might span multiple table spaces, an index on a partitioned table is created in a single table space. The index table space for each index can be specified in the CREATE INDEX statement.
- The use of indexes in a different location than the table is supported on both System-Managed Space (SMS) and Database-Managed Space (DMS) table spaces.
- All table spaces specified for the partitioned table must be in the same database partition group.
- Each index can be placed in its own table space, including large table spaces.
- The index table space type, either DMS or SMS, must be the same as that used for the data partitions.

Benefits of an index on a partitioned table are:

- Performance of online index creation and dropping of indexes is improved.
- You have 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 might be appropriate to ensure better space utilization).
- More efficient concurrent access to the index data for the table.
- You can reorganize an individual index on a range partitioned table.

4.2 Implementing table partitioning

This section describes the implementation of table partitioning using both the Control Center and Command Line Processor (CLP) commands, as well as the relationships of the table spaces, partitions, file systems, and logical drives.

Basic syntax for the CREATE TABLE statement for data partitioned table is:

```
CREATE TABLE table-name (column-definition )
   PARTITION BY RANGE   (partition key columns)
   (
    PARTITION partition-name starting-clause ending-clause
         IN tablespace-name
     ...
    PARTITION partition-name starting-clause ending-clause
         IN tablespace-name
    )
```

You can read the complete details of the syntax in the DB2 Information Center at:

http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.udb.admin.doc/doc/r0000927.htm

4.2.1 Creating a data partitioned table

A simple partitioned table was implemented by using the Control Center and using the DB2 Command Line Processor (CLP). The table definition is shown in Figure 4-1.

```
Columns
CUSTNO                                BIGINT
TRANSACTION_DATE                      DATE
AMOUNT                                DECIMAL(15,2)
CUST_NAME                             CHAR(10)
Range TRANSACTION_DATE
PARTITIONING (partition name, range, table space name(Large,16k)
  INV_PRE00 (< 01/01/2000)              INV_PRE00
  INV_2000  (01/01/2000 - 31/12/2000)   INV_2000
  INV_2001  (01/01/2001 - 31/12/2001)   INV_2001
  INV_2002  (01/01/2002 - 31/12/2002)   INV_2002
  INV_2003  (01/01/2003 - 31/12/2003)   INV_2003
  INV_2004  (01/01/2004 - 31/12/2004)   INV_2004
  INV_2005  (01/01/2005 - 31/12/2005)   INV_2005
  INV_2006  (01/01/2006 - 31/12/2006)   INV_2006
  INV_2007  (01/01/2007 - 31/12/2007)   INV_2007
  INDEXES                               INV_IX
  LONG                                  INV_LNG
  OTHER                                 INV_ALL
```

Figure 4-1 Table definition for the INVOICE table

Using the Control Center to implement a partitioned table

Using the Control Center, these are the steps that we used to create the partitioned table.

To start the Control Center:

Start → All Programs → IBM DB2 → DB2COPY1(Default) → General Administration Tools → Control Center

Creating the table spaces

The steps to create the table spaces are:

1. We created the required table spaces by selecting **Table Space** → **Create** as shown in Figure 4-2.

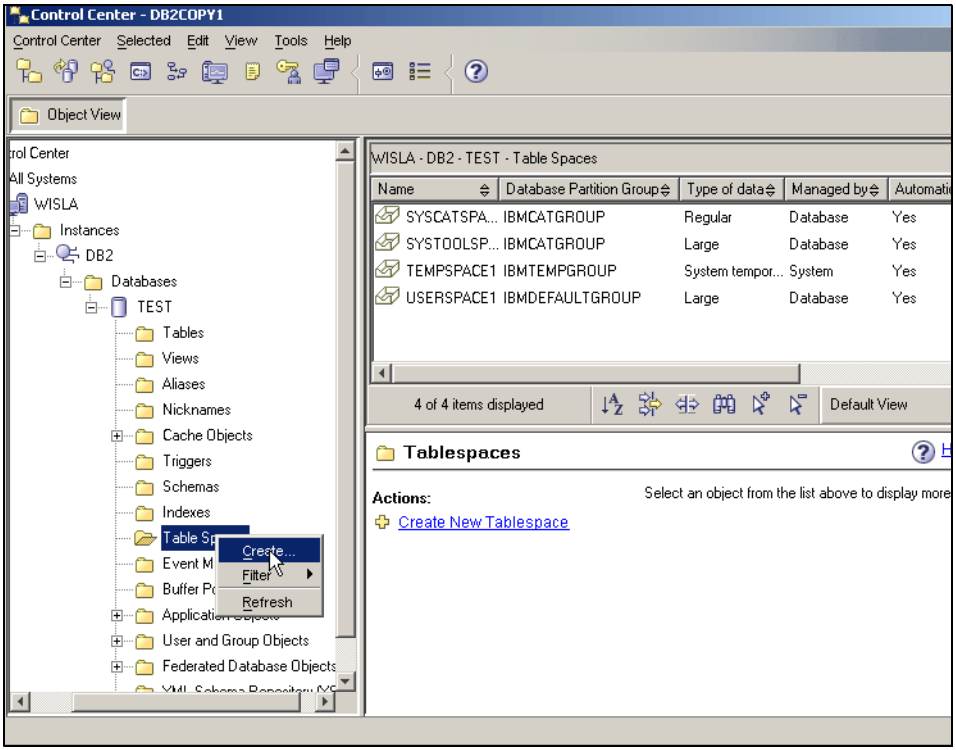


Figure 4-2 Create table spaces for INVOICE table

2. Then, we defined the table space parameters for `INV_PRE00`, which we depict in Figure 4-3 on page 132, Figure 4-4 on page 132, and Figure 4-5 on page 133.

Note: The data table spaces must have the same characteristics, such as regular or large, page size, and extent size.

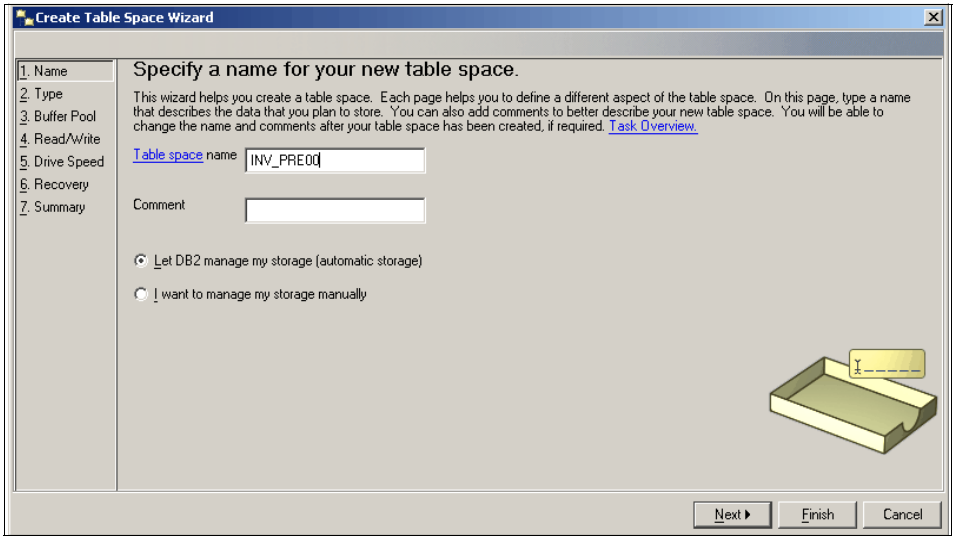


Figure 4-3 Defining the INV_PRE00 table space: Naming the table space

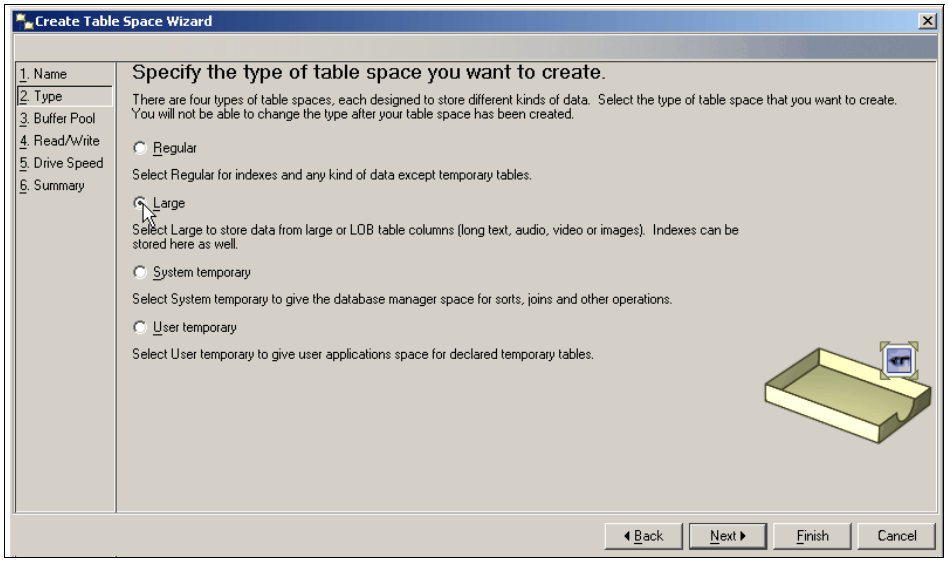


Figure 4-4 Defining the INV_PRE00 table space: Specify the type of table space

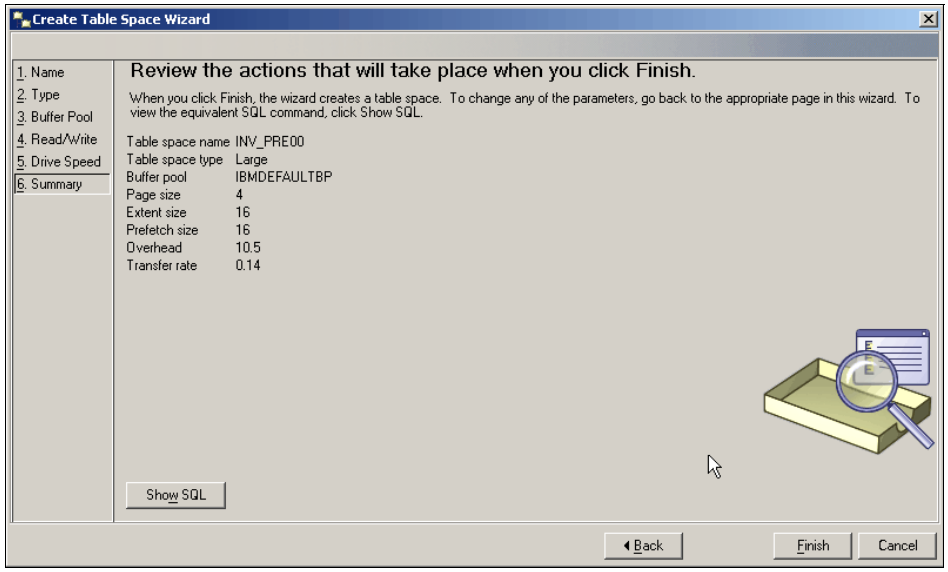


Figure 4-5 Defining the INV_PRE00 table space: Review parameters

3. This was repeated for the remaining table spaces as shown in the table definition in Figure 4-1 on page 130.

Creating the INVOICE table

The steps to create the INVOICE table are:

1. We started the creation of the INVOICE table. The initial Control Center actions are shown in Figure 4-6 on page 134 where we right-click **Table** → **Create**. Then, we defined the schema name and table name, which is shown in Figure 4-7 on page 134.

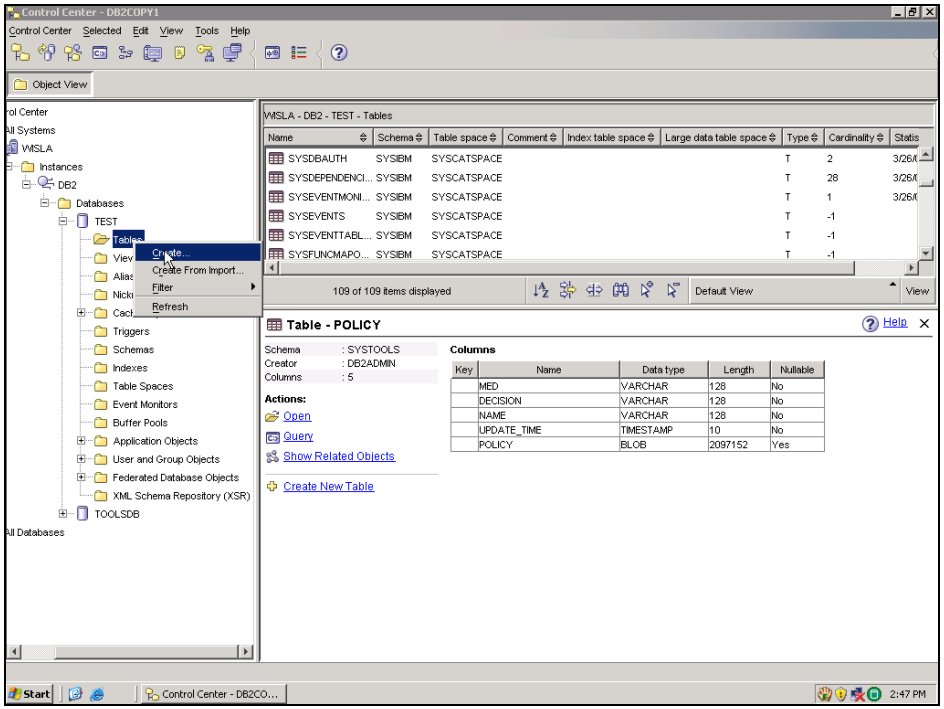


Figure 4-6 Creating the INVOICE table

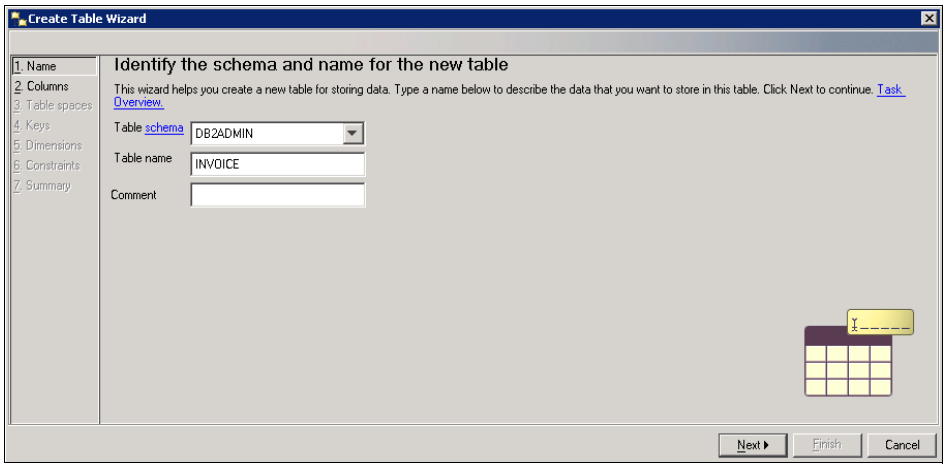


Figure 4-7 Creating the INVOICE table: Naming the table

2. In the Add Column window, we defined the columns as shown in Figure 4-8 on page 135.

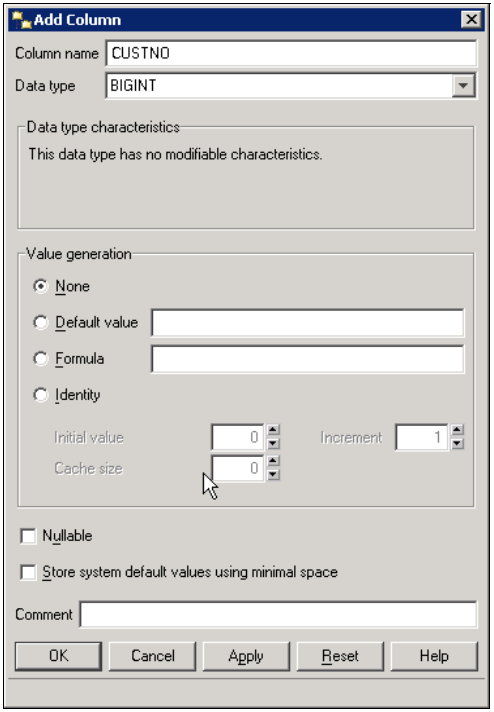


Figure 4-8 Creating the INVOICE table: Step two, defining the columns

3. In the Define data partitions window, we defined `TRANSACTION_DATE` as the partition range as shown in Figure 4-9 on page 136.

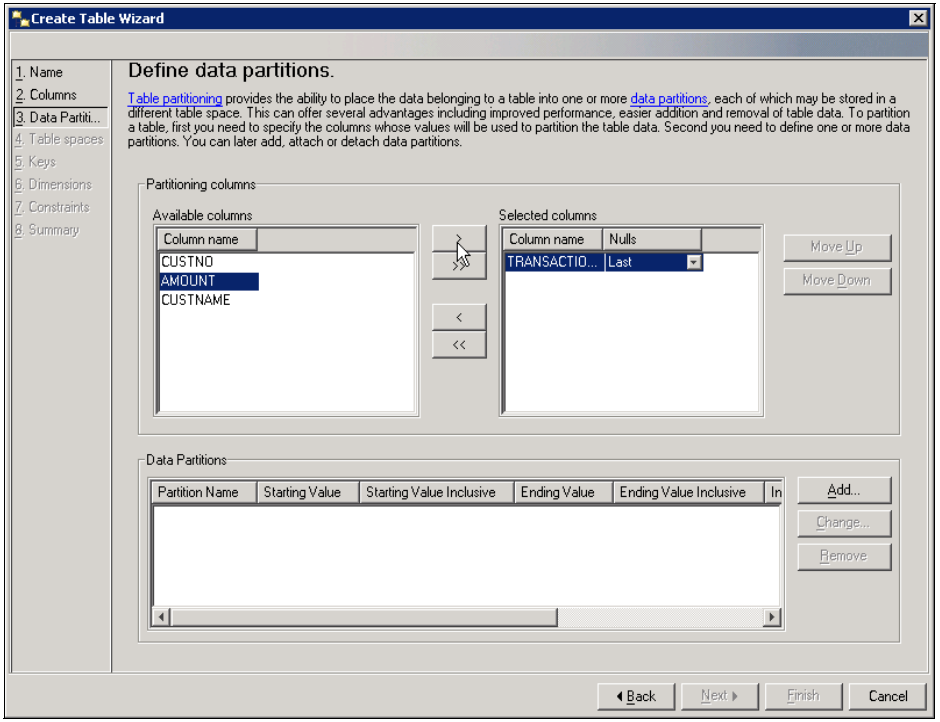


Figure 4-9 Creating the INVOICE table: Defining the range

4. Then, we defined the `INV_PRE00` partition as shown in Figure 4-10 on page 137. You must have already determined the starting and ending values for each partition.

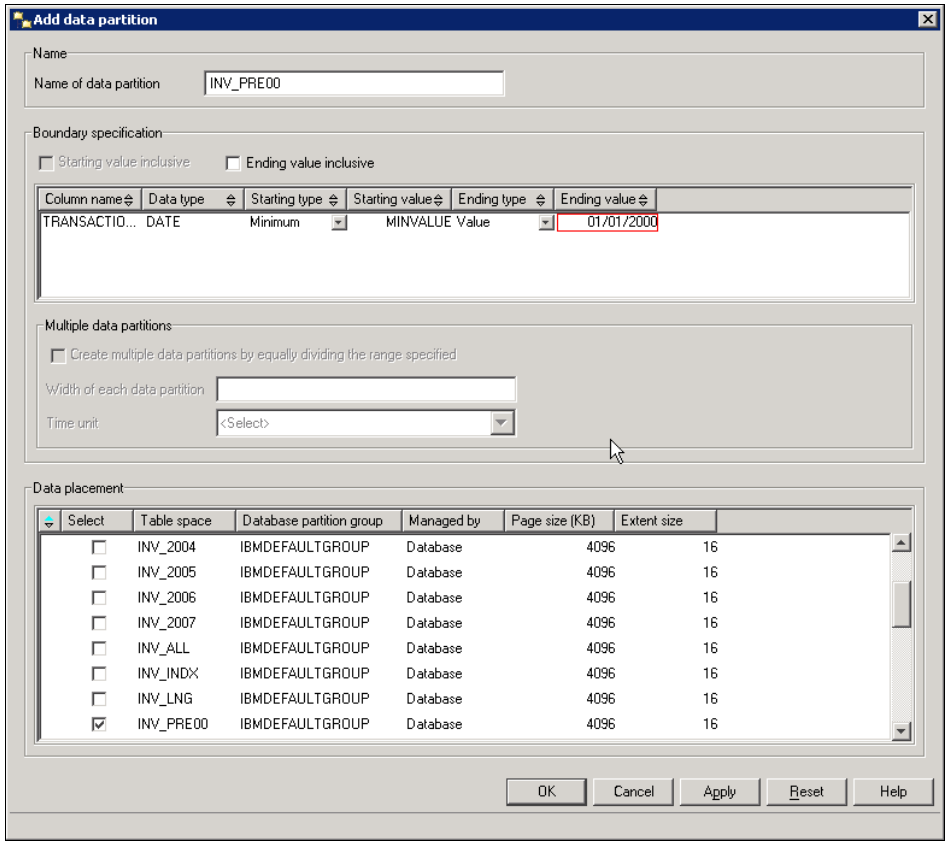


Figure 4-10 Creating the INVOICE table: Defining the INV_PRE00 partition

5. Next, we assigned the table spaces to the partition for data, long data, and indexes, which is shown in Figure 4-11 on page 138. You still have the opportunity at this point to create table spaces, but we suggest that you predefine all the table spaces that you require.

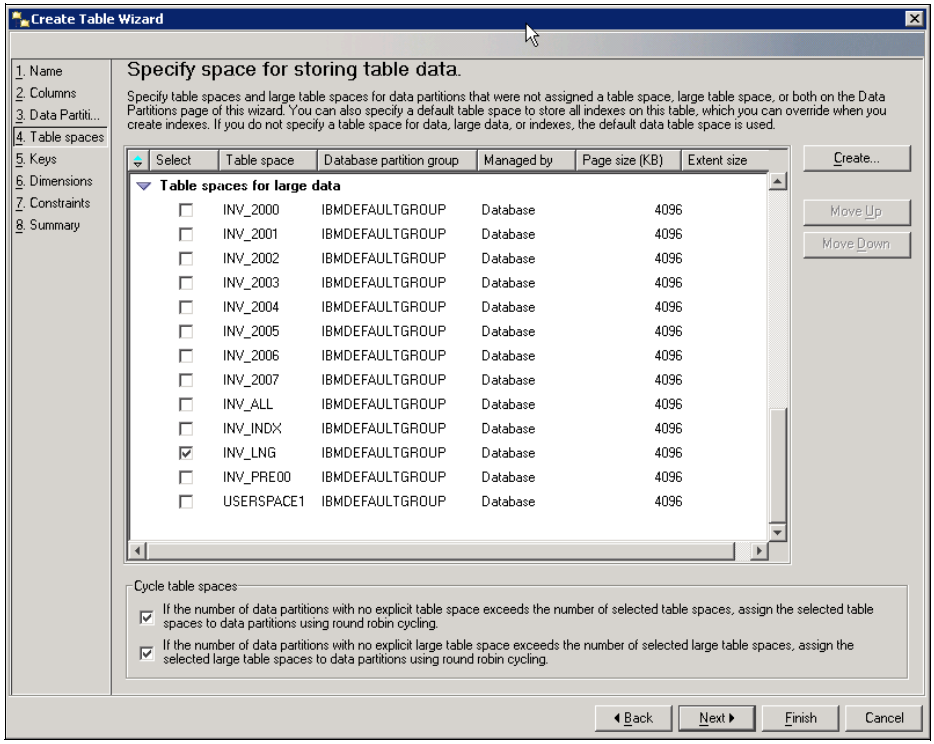


Figure 4-11 Creating the INVOICE table: Specify table space

6. Finally, in Figure 4-12 on page 139, you have the opportunity to review the SQL actions that are performed when you complete the task.

   This completes the setup of the partitioned table by using the Control Center.

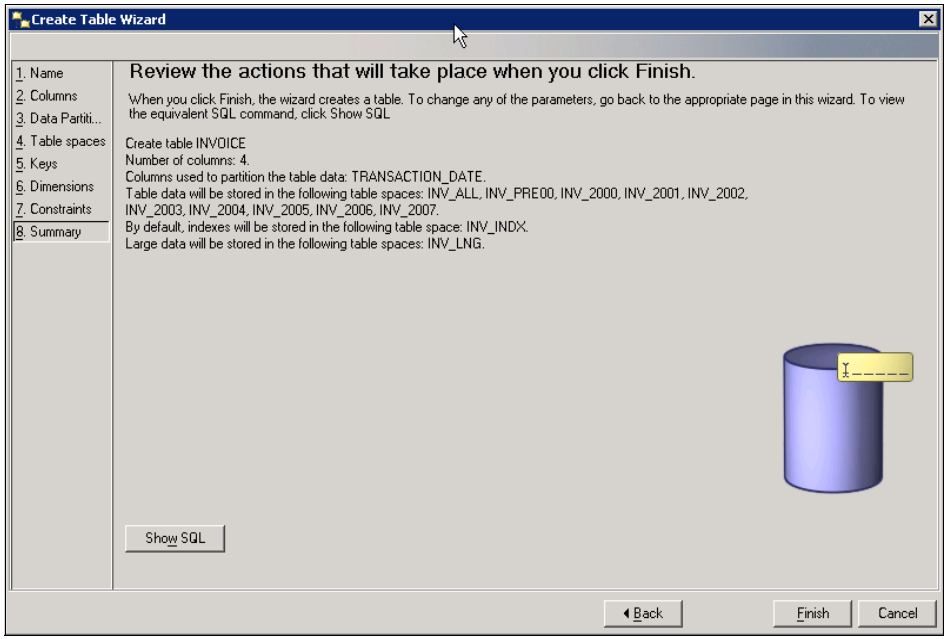


Figure 4-12 Creating the INVOICE table: Step six, reviewing the actions

Using the DB2 Command Line Processor

This section deals with the creation of the same table by using the DB2 Command Line Processor (CLP).

Creating the table spaces for the partitioned table

We performed the same exercise to create the table spaces for the partitioned table by using the SQL statements shown in Example 4-1.

Example 4-1 Create table spaces for INVOICE table

```
CREATE  LARGE  TABLESPACE inv_pre00 PAGESIZE 4 K  MANAGED BY AUTOMATIC
STORAGE EXTENTSIZE 16 OVERHEAD 10.5 PREFETCHSIZE 16 TRANSFERRATE 0.14
BUFFERPOOL  ibmdefaultbp ;

CREATE  LARGE  TABLESPACE inv_2000 PAGESIZE 4 K  MANAGED BY AUTOMATIC
STORAGE EXTENTSIZE 16 OVERHEAD 10.5 PREFETCHSIZE 16 TRANSFERRATE 0.14
BUFFERPOOL  ibmdefaultbp ;

CREATE  LARGE  TABLESPACE inv_2001 PAGESIZE 4 K  MANAGED BY AUTOMATIC
STORAGE EXTENTSIZE 16 OVERHEAD 10.5 PREFETCHSIZE 16 TRANSFERRATE 0.14
BUFFERPOOL  ibmdefaultbp ;
```

```
CREATE  LARGE  TABLESPACE inv_2002 PAGESIZE 4 K  MANAGED BY AUTOMATIC
STORAGE EXTENTSIZE 16 OVERHEAD 10.5 PREFETCHSIZE 16 TRANSFERRATE 0.14
BUFFERPOOL  ibmdefaultbp ;

CREATE  LARGE  TABLESPACE inv_2003 PAGESIZE 4 K  MANAGED BY AUTOMATIC
STORAGE EXTENTSIZE 16 OVERHEAD 10.5 PREFETCHSIZE 16 TRANSFERRATE 0.14
BUFFERPOOL  ibmdefaultbp ;

CREATE  LARGE  TABLESPACE inv_2004 PAGESIZE 4 K  MANAGED BY AUTOMATIC
STORAGE EXTENTSIZE 16 OVERHEAD 10.5 PREFETCHSIZE 16 TRANSFERRATE 0.14
BUFFERPOOL  ibmdefaultbp ;

CREATE  LARGE  TABLESPACE inv_2005 PAGESIZE 4 K  MANAGED BY AUTOMATIC
STORAGE EXTENTSIZE 16 OVERHEAD 10.5 PREFETCHSIZE 16 TRANSFERRATE 0.14
BUFFERPOOL  ibmdefaultbp ;

CREATE  LARGE  TABLESPACE inv_2006 PAGESIZE 4 K  MANAGED BY AUTOMATIC
STORAGE EXTENTSIZE 16 OVERHEAD 10.5 PREFETCHSIZE 16 TRANSFERRATE 0.14
BUFFERPOOL  ibmdefaultbp ;

CREATE  LARGE  TABLESPACE inv_2007 PAGESIZE 4 K  MANAGED BY AUTOMATIC
STORAGE EXTENTSIZE 16 OVERHEAD 10.5 PREFETCHSIZE 16 TRANSFERRATE 0.14
BUFFERPOOL  ibmdefaultbp ;

CREATE  LARGE  TABLESPACE inv_all PAGESIZE 4 K  MANAGED BY AUTOMATIC
STORAGE EXTENTSIZE 16 OVERHEAD 10.5 PREFETCHSIZE 16 TRANSFERRATE 0.14
BUFFERPOOL  ibmdefaultbp ;

CREATE  LARGE  TABLESPACE inv_lng PAGESIZE 4 K  MANAGED BY AUTOMATIC
STORAGE EXTENTSIZE 16 OVERHEAD 10.5 PREFETCHSIZE 16 TRANSFERRATE 0.14
BUFFERPOOL  ibmdefaultbp ;

CREATE  LARGE  TABLESPACE inv_indx PAGESIZE 4 K  MANAGED BY AUTOMATIC
STORAGE EXTENTSIZE 16 OVERHEAD 10.5 PREFETCHSIZE 16 TRANSFERRATE 0.14
BUFFERPOOL  ibmdefaultbp ;
```

Create the table

We created the table by using the statements in Example 4-2 on page 141. Note the use of the inclusive and exclusive parameters. As an alternative, you can specify the ending date as 12/31/20*xx* inclusive.

Example 4-2 Create INVOICE table

```
CREATE TABLE invoice (
  custno BIGINT  NOT NULL ,
  transaction_date DATE  NOT NULL ,
  amount DECIMAL (15, 2)  NOT NULL ,
  custname CHARACTER (10)  NOT NULL   )
  PARTITION BY RANGE (transaction_date)  (
  PARTITION inv_pre00 STARTING FROM (MINVALUE) EXCLUSIVE ENDING AT
    ('01/01/2000') EXCLUSIVE IN inv_pre00 ,
  PARTITION inv_2000 STARTING FROM ('01/01/2000') INCLUSIVE ENDING AT
    ('01/01/2001') EXCLUSIVE IN inv_2000 ,
  PARTITION inv_2001 STARTING FROM ('01/01/2001') INCLUSIVE ENDING AT
    ('01/01/2002') EXCLUSIVE IN inv_2001 ,
  PARTITION inv_2002 STARTING FROM ('01/01/2002') INCLUSIVE ENDING AT
    ('01/01/2003') EXCLUSIVE IN inv_2002 ,
  PARTITION inv_2003 STARTING FROM ('01/01/2003') INCLUSIVE ENDING AT
    ('01/01/2004') EXCLUSIVE IN inv_2003 ,
  PARTITION inv_2004 STARTING FROM ('01/01/2004') INCLUSIVE ENDING AT
    ('01/01/2005') EXCLUSIVE IN inv_2004 ,
  PARTITION inv_2005 STARTING FROM ('01/01/2005') INCLUSIVE ENDING AT
    ('01/01/2006') EXCLUSIVE IN inv_2005 ,
  PARTITION inv_2006 STARTING FROM ('01/01/2006') INCLUSIVE ENDING AT
    ('01/01/2007') EXCLUSIVE IN inv_2006 ,
  PARTITION inv_2007 STARTING FROM ('01/01/2007') INCLUSIVE ENDING AT
    ('01/01/2008') EXCLUSIVE IN inv_2007 )
  IN inv_all INDEX IN inv_indx LONG IN inv_lng;
```

4.2.2 Adding a new partition

To add a new partition, you can use the ADD PARTITION and ATTACH PARTITION options of the ALTER TABLE statement.

ADD PARTITION

Adding a data partition to a partitioned table by using ADD PARTITION adds an *empty* partition to the table. After you add the new partition, you can then insert or load data directly into the partitioned table.

Example 4-3 on page 142 shows the SQL statements to add a new partition for the 2008 year. You can do this in preparation for the change to the new calendar year.

Example 4-3 Add partition to INVOICE table for year 2008

```
CREATE LARGE TABLESPACE inv_2008 PAGESIZE 4 K  MANAGED BY AUTOMATIC
STORAGE
   EXTENTSIZE 16
   OVERHEAD 10.5
   PREFETCHSIZE 16
   TRANSFERRATE 0.14
   BUFFERPOOL  ibmdefaultbp ;

ALTER TABLE invoice
   ADD PARTITION inv_2008
       STARTING FROM ('01/01/2008') INCLUSIVE ENDING AT
         ('01/01/2009') EXCLUSIVE IN inv_2008 ;
```

ATTACH PARTITION

Alternatively, you can add a new partition to a data partitioned table by using the ATTACH PARTITION option. This is an efficient way to roll-in table data to a data partitioned table.

The attaching partition process is:

- Create a non-partitioned compatible table.
- Load the data into the newly created table.
- Attached the new table to the partitioned table.

Note that the table to be attached must be compatible with the target partitioned table. Create the new table by using the same DDL of the source table. If the target table is an existing table, make sure the table definitions match. There is no data movement during ATTACH. Because the new partition is essentially the same physical data object as the stand-alone table, the new partition inherits the table space usage from the original stand-alone table. So, create the stand-alone table in the correct table space before attaching.

Example 4-4 on page 143 illustrates attaching table `INVOICE_DATE_2008Q1` to `INVOICE_DATE`.

Example 4-4 Attaching a partition for first quarter of 2008

```
CREATE TABLE invoice_date_2008q1(
   custno BIGINT    NOT NULL ,
   transaction_date DATE NOT NULL ,
   amount           DECIMAL(15,2) NOT NULL ,
   custname         CHAR(10) NOT NULL ,
   time TIME        NOT NULL ,
   region           INTEGER NOT NULL )
IN inv_tbsp2008_01;

ALTER TABLE invoice_date
    ATTACH PARTITION inv_2008qtr1
      STARTING FROM ('01/01/2008') INCLUSIVE
      ENDING AT ('04/01/2008')
      EXCLUSIVE FROM TABLE invoice_date_2008qtr1;

SET INTEGRITY FOR invoice_date IMMEDIATE CHECKED;
```

4.2.3 Detaching a partition

To roll-out or detach a partition, use the DETACH PARTITION option of the ALTER TABLE statement.

DETACH PARTITION

If you decide to remove all of the pre-2000 data from this table to store it in a separate table, you use the DETACH parameter of the ALTER TABLESPACE statement. To demonstrate this, we have detached the `INV_PRE00` partitions into a table called `INV_PRE00`. The statements that we used are in Example 4-5.

Example 4-5 Detaching partition INV_PRE00 into table INV_PRE00

```
ALTER TABLE invoice DETACH PARTITION inv_pre00 INTO TABLE inv_pre00 ;
```

Note that the new table resulting from the detached partition resides in the same table space as the original partition. If exclusive use of that table space is required for the partitioned table space, you have to DROP and re-CREATE the table elsewhere.

Asynchronous index cleanup

Asynchronous index cleanup (AIC) is a new feature that automatically cleans up the index entries and reclaims index space for the partition that has been detached. It is triggered by the DETACH or after refreshing dependent MQTs. AIC runs as a low priority background process to remove all the index entries

corresponding to data that has been rolled-out. Meanwhile, the entries are present, but invisible to normal queries. The features that make AIC unobtrusive are:

- Periodically checks for waiters and releases locks.
- AIC does not activate a database partition.
- AIC does not keep a database partition active if all applications disconnect.

The detaching process provides higher table availability than the bulk data deletion. Figure 4-13 shows the table availability for DETACH for non-MQT tables. When detaching a partition, there might be slight contention during index cleaning if AIC performs the index cleaning.

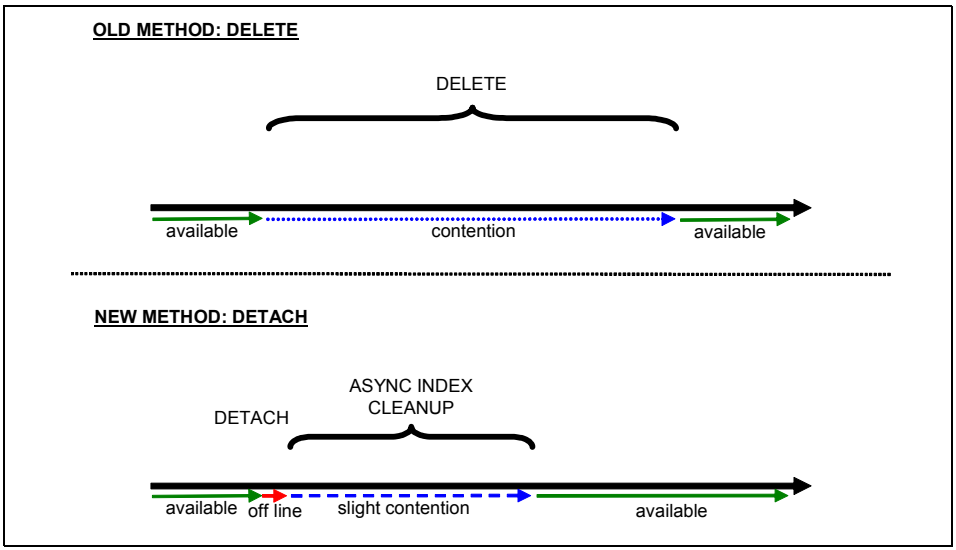


Figure 4-13 Non-MQT table availability for DETACH

Figure 4-14 on page 145 shows the table availability for DETACH for refresh immediate MQTs.

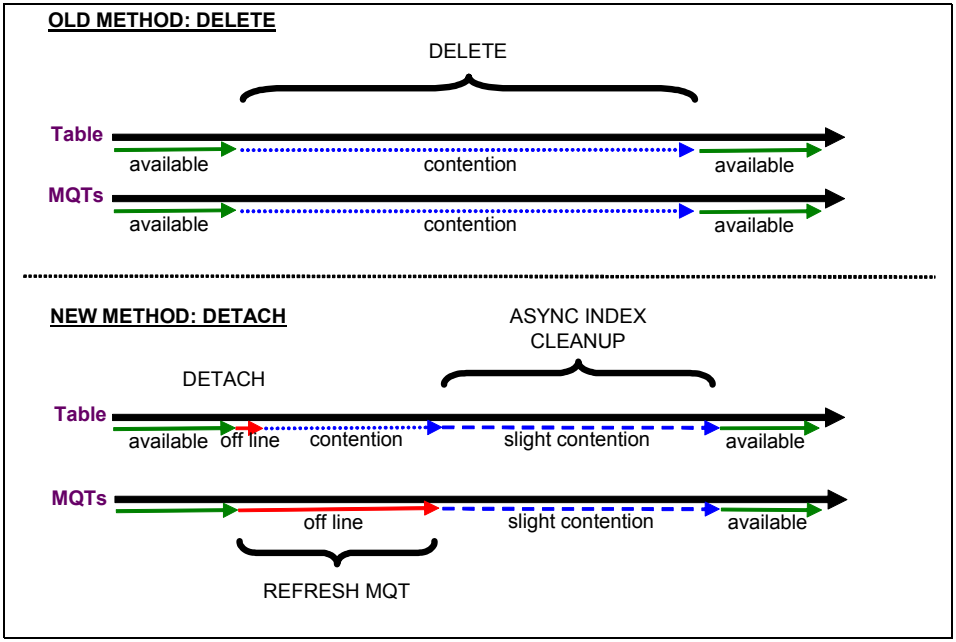


Figure 4-14 Table Availability for DETACH (with refresh Immediate MQTs)

Considerations when you use DETACH to roll-out a range of data:

- Rolled-out data is available in a new, separate table.
- Data disappears from view immediately upon DETACH.
- Delete triggers do not fire for DETACH.
- Rolled-out data can be dropped, archived, or moved to another storage management system.
- Queries are drained and table-locked by DETACH.
- Dependent refresh immediate MQTs go offline and need to be refreshed using SET INTEGRITY.

4.2.4 Re-attaching a partition

There might be a later requirement to add the data for `1999` back into the `INVOICE` table. To do this, we chose to use the ATTACH partition method, which required the steps shown in the statements in Example 4-6 on page 146. If an error was made and the data was outside of the partition definition, an SQL0327 message appeared.

The data for `1999` was separated into a new table in preparation for the ATTACH.

Example 4-6 Attaching a partition to INVOICE with data for 1999

```
CREATE  LARGE  TABLESPACE inv_1999 PAGESIZE 4 K  MANAGED BY AUTOMATIC
STORAGE EXTENTSIZE 16 OVERHEAD 10.5 PREFETCHSIZE 16
TRANSFERRATE 0.14 BUFFERPOOL  ibmdefaultbp ;

CREATE TABLE inv_1999 LIKE inv_pre00 ;
INSERT INTO inv_1999 SELECT * FROM inv_pre00
WHERE transaction_date >= '01/01/1999'
AND transaction_date <= '12/31/1999' ;

ALTER TABLE invoice ATTACH PARTITION inv_1999
STARTING FROM ('01/01/1999') INCLUSIVE
ENDING AT ('01/01/2000') EXCLUSIVE
FROM TABLE inv_1999 ;

SET INTEGRITY FOR invoice IMMEDIATE CHECKED;
```

Alternatively, we can add a partition by using the ADD partition method as in Example 4-7 and then insert the data by using a select from the `INV_PRE00` table. This achieves the same result.

Example 4-7 Adding a partition for 1999 and inserting data

```
CREATE  LARGE  TABLESPACE inv_1999 PAGESIZE 4 K  MANAGED BY AUTOMATIC
STORAGE EXTENTSIZE 16 OVERHEAD 10.5 PREFETCHSIZE 16
TRANSFERRATE 0.14 BUFFERPOOL  ibmdefaultbp ;

ALTER TABLE invoice ADD PARTITION inv_1999
STARTING FROM ('01/01/1999') INCLUSIVE
ENDING AT ('01/01/2000') EXCLUSIVE IN inv_1999 ;

INSERT INTO invoice SELECT * FROM inv_pre00
WHERE transaction_date >= '01/01/1999'
AND transaction_date <= '12/31/1999' ;
```

As a result, we now have pre-2000 data residing in a separate table (`INV_PRE00`), `1999` data included in the `INVOICE` table, and a space for new data for 2008.

4.2.5 RANGE option

Understand the RANGE option is essential for partitioned table design. You must determine the RANGE granularity for the individual application before you

implement the partitioned table. After you select the range, it cannot be changed dynamically.

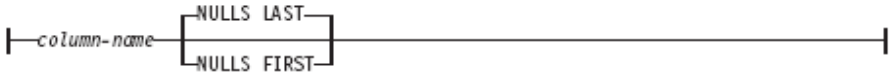
The RANGE option of the CREATE TABLE command defines the ranges of the partitions and, in the case of the automatically generated range, defines the number of partitions.

The RANGE specification is a combination of the RANGE partition expression and the RANGE partition element:

```
PARTITION BY RANGE (partition-expression) (partition-element)
```

RANGE partition expression

The *partition expression* defines the column or columns for the partitioning key. The maximum columns allowed is 16. The syntax of the partition expression is:



For example,

```
PARTITIONED BY RANGE (sales_region)
```

The NULLS LAST and NULLS FIRST specify that the null values compare high or low. Not all the data types supported by the partitioned table can be the partitioning key column data type. For the details of the supported data types, refer to the DB2 Information Center at:

http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp

RANGE partition element

The *partition element* specifies the ranges for the data partitioning key. Figure 4-15 on page 148 shows the syntax of the partition element.

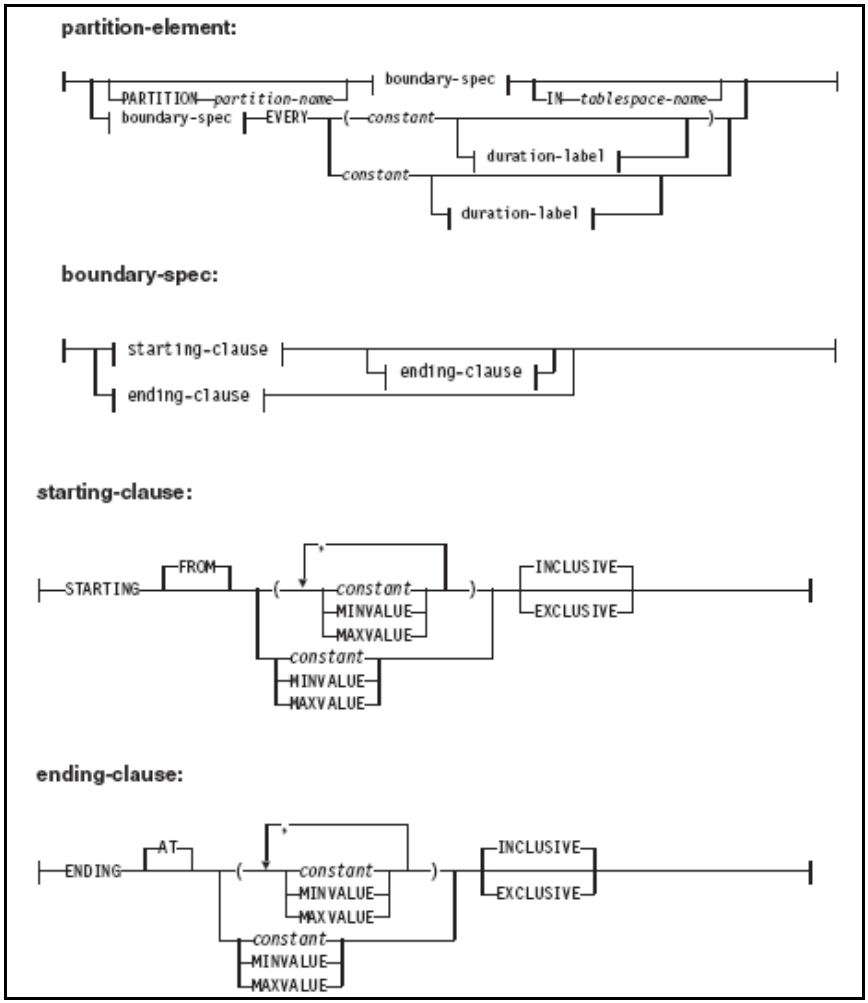


Figure 4-15 Partition element syntax

Each data partition has to have a unique name. If the partition name is not specified, the name defaults to PART*x* where *x* is an integer generated to keep the names unique. The IN tablespace-name clause defines the table space where the rows of the table in the range are stored.

The RANGE granularity can vary by partition as in Example 4-8 on page 149. In this example, we have accumulated the historical (`pre-2007`) data into two partitions, because we are more concerned with current `2007` data. The current `2007` data resides in its own partition, and the `2008` data will also have its own partition. Because of the improved scan performance, the current data and future

data have improved performance because their partitions are in separate table spaces.

Example 4-8 Range granularity

```
PARTITION inv_0 STARTING (MINVALUE) ENDING AT ('12/31/2001') INCLUSIVE
IN inv_tbsp00 ,
PARTITION inv_1 STARTING FROM ('01/01/2002') INCLUSIVE ENDING AT
('12/31/2006') INCLUSIVE IN inv_tbsp01 ,
PARTITION inv_2 STARTING FROM ('01/01/2007') INCLUSIVE ENDING AT
('12/31/2007') INCLUSIVE IN inv_tbsp02 ,
PARTITION inv_3 STARTING FROM ('01/01/2008') INCLUSIVE ENDING AT
('12/31/2008')IN inv_tbsp03;
```

To completely define the range for each data partition, you must specify sufficient boundaries. Consider these guidelines when you define 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. 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. 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, the next greater data partition must specify a STARTING clause.

  Likewise, if you do not specify a STARTING clause, the previous data partition must specify an ENDING clause.

- MINVALUE specifies a value that is smaller than any possible value for the column type that is used.

  You cannot specify MINVALUE and INCLUSIVE or EXCLUSIVE together. Also, MINVALUE precludes the addition of a partition with a lower range.

- MAXVALUE specifies a value that is larger than any possible value for the column type used.

  You cannot specify MAXVALUE and INCLUSIVE or EXCLUSIVE together. Also, MAXVALUE precludes the addition of a partition with a higher range.

- INCLUSIVE indicates that all values equal to the specified value are included in the data partition containing this boundary.

- EXCLUSIVE indicates that all values equal to the specified value are *not* included in the data partition containing this boundary.
- The NULL clause specifies whether null values are 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 data partition is defined, null values are considered 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 appear last in a sorted list of values. FIRST specifies that null values appear first in a sorted list of values.
- When using the long form of the syntax, each data partition must have at least one boundary specified.
- IN indicates the table space where the partition is to be located.

  This table space can be a table space that is used for other partitions or an individual table space for a particular partition.

If you choose a range granularity that is inappropriate, you cannot change the range definitions “in place.”

Your options to change range definitions are:

- Export the data, drop the table, create a table, import data, or load data.
- Detach partitions, add partitions, or import data.
- Detach partitions, manipulate data into new tables that match the required range specifications, and attach the resulting tables as the new partitions.

Automatically generated ranges

A simple way of creating many data partitions quickly and easily is to use *automatic generation*. It is an appropriate method for equal-sized ranges based on dates or numbers. However, there are limitations on the placement of data and the naming of the data partitions. Example 4-9 on page 151 includes the statement to create a partitioned table with automatically generated ranges. Note the use of the EVERY option.

Example 4-9 Automatically generated ranges

```
CREATE TABLE     invoice_auto
   (custno BIGINT NOT NULL,
    transaction_date DATE NOT NULL,
    amount DECIMAL (15, 2) NOT NULL ,
    custname CHARACTER (10) NOT NULL)
PARTITION BY RANGE (transaction_date NULLS LAST)
    (STARTING FROM ('01/01/2000') INCLUSIVE
     ENDING AT ('12/31/2007') INCLUSIVE
     EVERY (1 YEARS)
     ) IN inv_all ;
```

Restrictions when using the automatically generated ranges include:

- MINVALUE and MAXVALUE are not supported in the automatically generated form of the syntax.
- You can only specify one column for the range in the automatically generated form of the syntax.
- Tables that you create by using the automatically generated form of the syntax (containing the EVERY clause) are constrained to use a numeric, date, or time type for the table partitioning key.

Manually generated ranges

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 you define ranges, therefore increasing your data and large object (LOB) placement options. Example 4-10 on page 152 includes the statement for a table with manually created ranges. Note the placement of each data partition in a separate table space.

Example 4-10 Manually generated ranges

```
CREATE TABLE invoice
  (custno BIGINT NOT NULL ,
   transaction_date DATE
   amount DECIMAL (15, 2)  NOT NULL ,
   custname CHARACTER (10)  NOT NULL ,
   time TIME  NOT NULL,
   region INT NOT NULL)
PARTITION BY RANGE(transaction_date)
(PART inv_2000 STARTING('01/01/2000') ENDING('12/31/2000') INCLUSIVE IN inv_2000,
 PART inv_2001 STARTING('01/01/2001') ENDING('12/31/2001') INCLUSIVE IN inv_2001,
 PART inv_2002 STARTING('01/01/2002') ENDING('12/31/2002') INCLUSIVE IN inv_2002,
 PART inv_2003 STARTING('01/01/2003') ENDING('12/31/2003') INCLUSIVE IN INV_2003,
 PART inv_2004 STARTING('01/01/2004') ENDING('12/31/2004') INCLUSIVE IN INV_2004,
 PART inv_2006 STARTING('01/01/2006') ENDING('12/31/2006') INCLUSIVE IN inv_2006,
 PART inv_2007 STARTING('01/01/2007') ENDING('12/31/2007') INCLUSIVE IN inv_2007
);
```

Example 4-11 shows the use of multiple columns in the range definition.

Example 4-11 Range specification with two columns: year and month

```
CREATE TABLE invoice
   (custno INTEGER  NOT NULL ,
    transaction_date DATE  NOT NULL ,
    amount INTEGER  NOT NULL ,
    cust_name CHARACTER (10)  NOT NULL,
    inv_month INT NOT NULL GENERATED ALWAYS AS (MONTH(transaction_date)),
    inv_year  INT NOT NULL GENERATED ALWAYS AS (YEAR(transaction_date)))
PARTITION BY RANGE (inv_year, inv_month)
  (PARTITION prt2004_1 STARTING FROM (2004,1)  INCLUSIVE ENDING AT (2004,3)  INCLUSIVE
                                                                    IN inv_tsd20041,
   PARTITION prt2004_2 STARTING FROM (2004,4)  INCLUSIVE ENDING AT (2004,6)  INCLUSIVE
                                                                    IN inv_tsd20042,
   PARTITION PRT2004_3 STARTING FROM (2004,7)  INCLUSIVE ENDING AT (2004,9)  INCLUSIVE
                                                                    IN INV_TSD20043,
   PARTITION PRT2004_4 STARTING FROM (2004,10) INCLUSIVE ENDING AT (2004,12) INCLUSIVE
                                                                    IN inv_tsd20044,
   PARTITION prt2005_1 STARTING FROM (2005,1)  INCLUSIVE ENDING AT (2005,3)  INCLUSIVE
                                                                    IN inv_tsd20051,
   PARTITION PRT2005_2 STARTING FROM (2005,4)  INCLUSIVE ENDING AT (2005,6)  INCLUSIVE
                                                                    IN INV_TSD20052,
   PARTITION prt2005_3 STARTING FROM (2005,7)  INCLUSIVE ENDING AT (2005,9)  INCLUSIVE
                                                                    IN inv_tsd20053,
   PARTITION prt2005_4 STARTING FROM (2005,10) INCLUSIVE ENDING AT (2005,12) INCLUSIVE
                                                                    IN inv_tsd20054);
```

Example 4-12 shows the use of multiple columns of different data types. Here, the range specification is year and region number.

Example 4-12 Range specification with multiple columns: year and region

```
(CREATE TABLE invoice
  (custno INTEGER  NOT NULL ,
   transaction_date DATE  NOT NULL ,
   amount INTEGER  NOT NULL ,
   cust_name CHARACTER (10)  NOT NULL,
   inv_year INT NOT NULL GENERATED ALWAYS AS (YEAR(transaction_date)),
   region INTEGER)
PARTITION BY RANGE (inv_year,region)
  (PARTITION prt2004_1 STARTING FROM (2004,1) INCLUSIVE ENDING AT (2004,3)  INCLUSIVE
                                                                            IN inv_ts2004r1,
   PARTITION prt2004_2 STARTING FROM (2004,4) INCLUSIVE ENDING AT (2004,6)  INCLUSIVE
                                                                            IN inv_ts2004r4,
   PARTITION prt2004_3 STARTING FROM (2004,7) INCLUSIVE ENDING AT (2004,8)  INCLUSIVE
                                                                            IN inv_ts2004r7,
   PARTITION prt2004_4 STARTING FROM (2004,9) INCLUSIVE ENDING AT (2004,10) INCLUSIVE
                                                                            IN inv_ts2004r9,
   PARTITION prt2005_1 STARTING FROM (2005,1) INCLUSIVE ENDING AT (2005,3)  INCLUSIVE
                                                                            IN inv_ts20051,
   PARTITION prt2005_2 STARTING FROM (2005,4) INCLUSIVE ENDING AT (2005,6)  INCLUSIVE
                                                                            IN inv_ts20054,
   PARTITION prt2005_3 STARTING FROM (2005,7) INCLUSIVE ENDING AT (2005,8)  INCLUSIVE
                                                                            IN inv_ts20057,
   PARTITION prt2005_4 STARTING FROM (2005,9) INCLUSIVE ENDING AT (2005,10) INCLUSIVE
                                                                            IN inv_ts20059);
```

Note: When multiple columns are used as the table partitioning key, they are treated as a composite key, which is similar to a composite key in an index in that the trailing columns are dependent on the leading columns.

This is demonstrated in Example 4-13 on page 154 where the statement depicted produces the error message shown below:

```
DB21034E  The command was processed as an SQL statement because it was
not a valid Command Line Processor command.  During SQL processing it
returned:
SQL0636N  Range specified for data partition "PRT2004_1" is not valid.
Reason code = "10".  SQLSTATE=56016
```

Reason code 10 states:

```
The range overlaps with another partition. Each data partition must
have a well defined starting and ending boundary and each data value
must go into one and only one data partition. Also, if the same value
```

```
(except MINVALUE or MAXVALUE) is used in the ending bound of one
partition and the starting bound of the next partition, then at least
one of these bounds must be defined as EXCLUSIVE.
```

This occurs because the trailing key parameter, in this case, `year`, is not the varying parameter. The correct syntax is shown in Example 4-12 on page 153.

Example 4-13 Range specification: Trailing column dependency

```
PARTITION BY RANGE (region,inv_year)
  (PARTITION prt2004_1 STARTING FROM (1,2004) INCLUSIVE ENDING AT (3,2004)  INCLUSIVE
                                                                    IN inv_ts2004r1,
   PARTITION prt2004_2 STARTING FROM (4,2004) INCLUSIVE ENDING AT (6,2004)  INCLUSIVE
                                                                    IN inv_ts2004r4,
   PARTITION prt2004_3 STARTING FROM (7,2004) INCLUSIVE ENDING AT (8,2004)  INCLUSIVE
                                                                    IN inv_ts2004r7,
   PARTITION prt2004_4 STARTING FROM (9,2004) INCLUSIVE ENDING AT (10,2004) INCLUSIVE
                                                                    IN inv_ts2004r9,
   PARTITION prt2005_1 STARTING FROM (1,2005) INCLUSIVE ENDING AT (3,2005)  INCLUSIVE
                                                                     IN inv_ts20051,
   PARTITION prt2005_2 STARTING FROM (4,2005) INCLUSIVE ENDING AT (6,2005)  INCLUSIVE
                                                                     IN inv_ts20054,
   PARTITION prt2005_3 STARTING FROM (7,2005) INCLUSIVE ENDING AT (8,2005)  INCLUSIVE
                                                                     IN inv_ts20057,
   PARTITION prt2005_4 STARTING FROM (9,2005) INCLUSIVE ENDING AT (10,2005) INCLUSIVE
                                                                    IN inv_ts20059);
```

4.2.6 Handling large objects

Large objects, by default, are stored in the same table space as the corresponding data objects. This default applies whether the partitioned table uses only one table space or multiple table spaces. That is, when a partitioned table's data is stored in multiple table spaces, the large object data is also stored in multiple table spaces.

You can override the default behavior by using the LONG IN clause of the CREATE TABLE statement. You can also specify a list of table spaces for the table where long data is to be stored. The space specified in the LONG IN clause must be a large table space.

You can specify a particular table space for all long data or specify a table space for the long data belonging to individual partitions. Example 4-14 on page 155 is an example of using a separate table space for the long data in each partition. Using this approach is called *storing long data remotely* (remote from the data). Note that you can specify the same table space for the long data of multiple partitions in the same way that you can specify the data table spaces.

Example 4-14 Specifying table spaces for long data

```
CREATE TABLE invoice (
custno BIGINT  NOT NULL ,
transaction_date DATE  NOT NULL ,
amount DECIMAL (15, 2)  NOT NULL ,
custname CHARACTER (10)  NOT NULL ,
time TIME  NOT NULL,
region INT NOT NULL)
PARTITION BY RANGE (TRANSACTION_DATE NULLS LAST)
  (PARTITION INV_0 STARTING FROM (MINVALUE) INCLUSIVE ENDING AT
                   ('12/31/2001') EXCLUSIVE IN inv_tbsp00 LONG IN inv_tbspl0,
   PARTITION INV_1 STARTING FROM ('01/01/2002') INCLUSIVE ENDING AT
                   ('12/31/2003') EXCLUSIVE IN inv_tbsp01 LONG IN inv_tbspl1,
   PARTITION INV_2 STARTING FROM ('01/01/2004') INCLUSIVE ENDING AT
                   ('12/31/2005') EXCLUSIVE IN inv_tbsp02 LONG IN inv_tbspl2,
   PARTITION INV_3 STARTING FROM ('01/01/2006') INCLUSIVE ENDING AT
                   ('12/31/2007') EXCLUSIVE IN inv_tbsp03 LONG IN inv_tbspl2);
```

In the next example, Example 4-15, there is a long table space, `INV_TBSPL99`, allocated for the partitions that have no long table space specified.

Example 4-15 Specifying a collective long table space

```
CREATE TABLE testtab(
  custno BIGINT  NOT NULL ,
  transaction_date DATE  NOT NULL ,
  amount DECIMAL (15, 2)  NOT NULL ,
  custname CHARACTER (10)  NOT NULL ,
  time TIME  NOT NULL,
  region INT NOT NULL)
  LONG IN inv_tbspl99
PARTITION BY RANGE (TRANSACTION_DATE NULLS LAST)(
   PARTITION INV_0 STARTING FROM (MINVALUE) INCLUSIVE ENDING AT
                   ('12/31/2001') EXCLUSIVE IN inv_tbsp00 LONG IN inv_tbspl0,
   PARTITION INV_1 STARTING FROM ('01/01/2002') INCLUSIVE ENDING AT
                   ('12/31/2003') EXCLUSIVE IN inv_tbsp01 LONG IN inv_tbspl1,
   PARTITION INV_2 STARTING FROM ('01/01/2004') INCLUSIVE ENDING AT
                   ('12/31/2005') EXCLUSIVE IN inv_tbsp02,
   PARTITION INV_3 STARTING FROM ('01/01/2006') INCLUSIVE ENDING AT
                   ('12/31/2007') EXCLUSIVE IN inv_tbsp03);
```

Note: You cannot store long data remotely for certain data partitions and store long data locally for other partitions.

4.2.7 Optimal storage configurations for table partitioning

This section discusses possible storage configuration advantages.

Table spaces, file systems, and logical drives

Because of the relationship between the table spaces and file systems or logical drives, there is an opportunity to manage the I/O performance of the individual data partition. For instance, if each container is defined to an individual drive (or group of drives), the I/O performance of one partition can be separated from another. That is, one or more data partitions in the table can be independent of others in terms of I/O performance.

For example, we can define each table space used by an individual partition to an individual physical container by defining:

- For UNIX and Linux:
  - A DMS table space specifying a file on a file system on a logical volume on a physical drive so that each table space container is independent of the other
  - A DMS table space specifying a raw device
- For Windows:
  - A DMS table space specifying a file on a logical drive on a physical drive
  - A DMS table space specifying a raw device

The other method is to allocate the less frequently used partition containers to slower disk. But, remember that there is an underlying improvement in the way that the optimizer might ignore certain partitions on scans depending on the nature of the transaction.

The table, partition, table space, and container layout in Figure 4-16 on page 157 suggests the possibilities of data distribution across disk that can be achieved with table partitioning. In this case, the older accumulated `pre-2000` data is accumulated on four file systems (or raw devices) utilizing older, slower drives. The data for the year `2000` is kept on three file systems (or raw devices) utilizing intermediate performance drives, and the current year data (`2007`) is spread over multiple file systems (or raw devices) utilizing the highest performance drives.

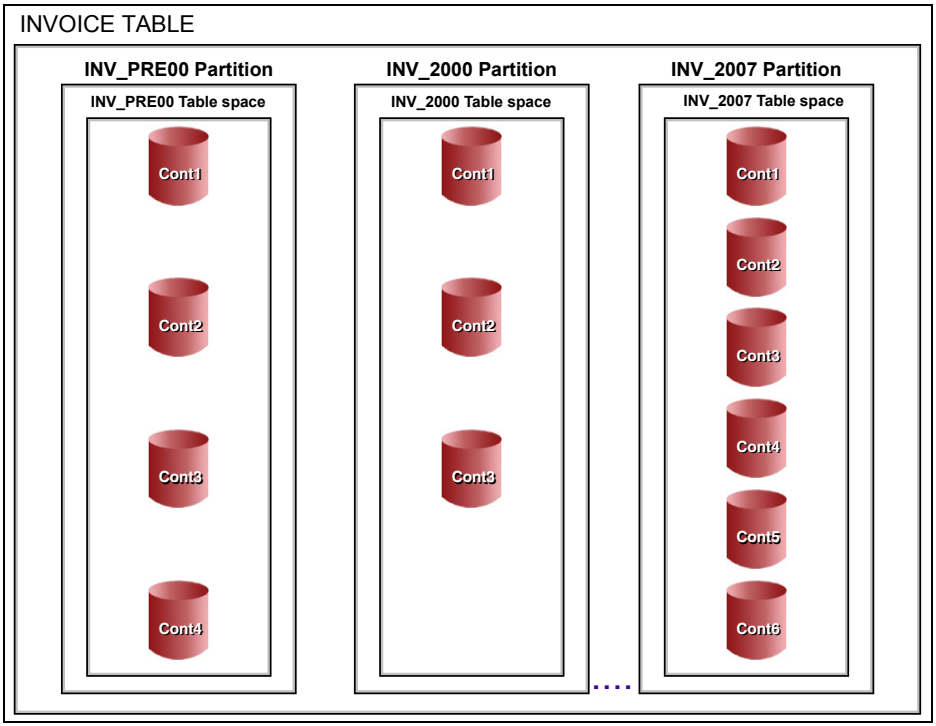


Figure 4-16 Relationships of partitions, table spaces, and containers

4.2.8 Partition elimination

Partition elimination is the ability of the optimizer to determine that certain ranges do not need to be accessed at all for a query. Partition elimination improves the performance for many queries. Figure 4-17 on page 158 shows how this works for a simple table scan. The optimizer determines that the data range specified in the WHERE clause resides on partition 2 and 3; therefore, only these partitions are scanned.

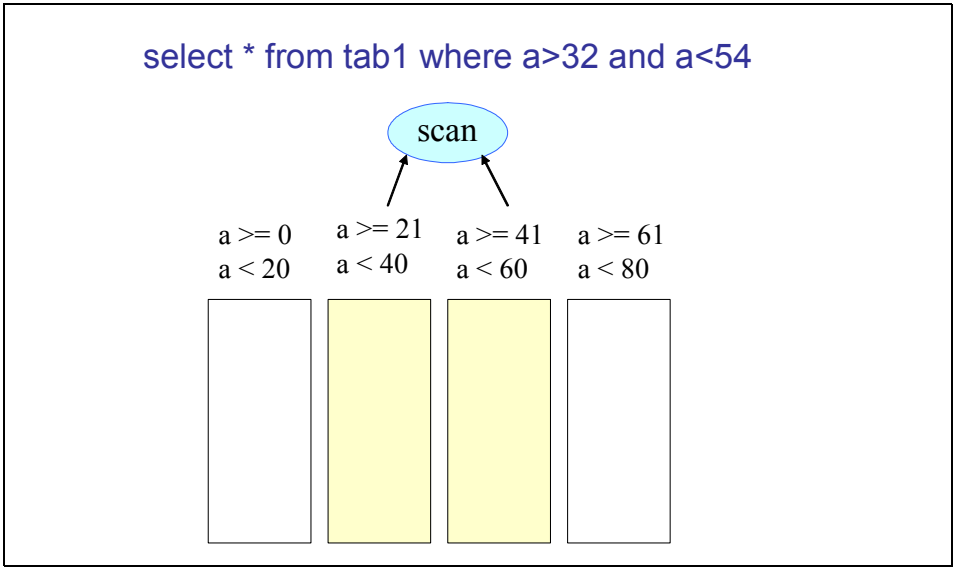


Figure 4-17 Partition elimination: Table scan

Partition elimination also applies for index scans. For example, many plans are possible for the query shown in Figure 4-18 on page 159. Without partitioning, one likely plan is to use "*index ANDing*." Index ANDing performs these tasks:

- Read all relevant index entries from each index.
- Save both sets of record IDs (RIDs).
- Match both sets of RIDs to see which occurred in both indexes.
- Use those to fetch the rows.

With partitioning, each RID in the index contains the `datapartID`. Instead of reading from the `l_shipdate` index, the optimizer looks at the `datapartID` to discover if the row might be in the desired date range. It saves half the I/O in indexes. Index ANDing passes RIDs back up to the runtime routine, "ands" them, and then goes back to the kernel to fetch them. In contrast, partition elimination skips irrelevant RIDs without ever returning them to run time, thus, improving performance.

```
SELECT l_shipdate, l_partkey, l_returnflag
  FROM lineitem
  WHERE l_shipdate BETWEEN
        '01/01/1993' AND '03/31/1993'
        AND l_partkey=49981
```

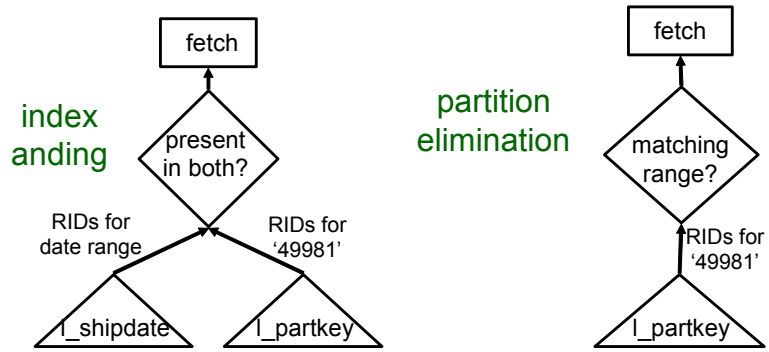


Figure 4-18 Partition elimination: Index scan

4.3 Administration and management

Consider these topics for your administration and management of partitioned tables.

4.3.1 Utilities

These are DB2 utilities affected by table partitioning.

BACKUP and RESTORE

Table partitioning provides the opportunity to reduce backup times for large tables, because the individual partitions can be backed up if they are allocated to different table spaces.

LOAD or IMPORT

LOAD or IMPORT can be used to load a partitioned table. The table is regarded by the utility as a standard table for the purposed of loading data with the following restrictions:

- Consistency points are not supported.

- 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.
- Exact ordering of input data records is not preserved when loading partitioned tables. Ordering is only maintained within the data partition.
- 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).
- A load replace operation does not truncate detached or attached data partitions.
- Because the LOAD utility acquires locks on the catalog system tables, the LOAD utility waits for any uncommitted ALTER TABLE transactions.

REORG

There are restrictions when using REORG TABLE for partitioned tables:

- You cannot use REORG on a partitioned table in a DMS table space during an online backup of *any* table space in which the table resides (including LOBs and indexes). You get an SQL2048 error. This does not occur when the table spaces are SMS.
- REORG is supported at the table level. You can reorganize an individual data partition by detaching the data partition, reorganizing the resulting non-partitioned table, and then reattaching the data partition. The table must have an ACCESS_MODE in SYSCAT.TABLES of Full Access.
- Reorganization skips data partitions that are in a restricted state due to an ATTACH or DETACH operation.
- If an error occurs, the non-partitioned indexes of the table are marked as bad indexes and are rebuilt on the next access to the table.
- If a reorganization operation fails, certain data partitions might be in a reorganized state and others might not. When the REORG TABLE command is reissued, all the data partitions are reorganized regardless of the data partitions' reorganization state.
- When reorganizing indexes on partitioned tables, we recommend that you perform a RUNSTATS operation after asynchronous index cleanup is complete in order to generate accurate index statistics in the presence of detached data partitions. To determine whether there are detached data partitions in the table, you can check the status field in

SYSDATAPARTITIONS and look for the value “I” (index cleanup) or “D” (detached with dependent MQT).

- The REORG INDEXES or REORG TABLE command is not supported on a partitioned table in ALLOW WRITE or ALLOW READ modes (except when CLEANUP ONLY is specified for REORG INDEXES).

4.3.2 DB2 Explain

Visual explain provides detailed information about which data partitions are used when a query is run. The **db2exfmt** command also provides details indicating which partitions are used when a query takes place. Example 4-16 contains the output when DB2 Explain has been used to analyze the following query:

```
SELECT * FROM INVOICE
WHERE TRANSACTION_DATE >='01/01/2000'
AND TRANSACTION_DATE<='12/31/2000'
```

- Table partition label (circled in Example 4-16)

  In the access plan, table partitioned tables are labeled with the tag DP-TABLE.

- List of data partitions accessed (circled in Example 4-16)

  Look for “List of data partitions accessed” to see which partitions are to be scanned after irrelevant ones have been excluded by partition elimination. In this example in Example 4-16, the information provided indicates that one partition (partition 3) was used in the scan.

  ```
  DPLSTPRT: (List of data partitions accessed)
     3
  DPNUMPRT: (Number of data partitions accessed)
     1
  ```

- DP Elim Predicates (circled in Example 4-16)

  When partition elimination is used, you can see the boundaries of the ranges scanned in the “Predicates” section labeled with DP Elim Predicates in Example 4-16.

  Notice that the terminology used is the same terminology that is used for index scans. Here, you can see start and stop predicates from your query that delimit the portions of the table that is scanned.

Example 4-16 db2exfmt output

```
DB2 Universal Database Version 9.1, 5622-044 (c) Copyright IBM Corp. 1991, 2006
Licensed Material - Program Property of IBM
IBM DATABASE 2 Explain Table Format Tool
```

```
******************** EXPLAIN INSTANCE ********************
DB2_VERSION: 09.01.2
SOURCE_NAME: SYSSH200
SOURCE_SCHEMA: NULLID
SOURCE_VERSION:
EXPLAIN_TIME: 2007-03-29-15.08.56.265004
EXPLAIN_REQUESTER: DB2ADMIN

Database Context:
----------------
   Parallelism: None
   CPU Speed: 2.282997e-007
   Comm Speed: 100
   Buffer Pool size: 1014
   Sort Heap size: 53
   Database Heap size: 1263
   Lock List size: 3251
   Maximum Lock List: 92
   Average Applications: 1
   Locks Available: 254228

Package Context:
---------------
   SQL Type: Dynamic
   Optimization Level: 5
   Blocking: Block All Cursors
   Isolation Level: Cursor Stability


---------------- STATEMENT 1  SECTION 65 ----------------
   QUERYNO: 1
   QUERYTAG:
   Statement Type: Select
   Updatable: No
   Deletable: No
   Query Degree: 1

Original Statement:
------------------
SELECT *
FROM DB2ADMIN.INVOICE
where transaction_date > '01/01/2002' and transaction_date < '03/01/2002'


Optimized Statement:
-------------------
SELECT Q1.CUSTNO AS "CUSTNO", Q1.TRANSACTION_DATE AS "TRANSACTION_DATE",
        Q1.AMOUNT AS "AMOUNT", Q1.CUSTNAME AS "CUSTNAME"
FROM DB2ADMIN.INVOICE AS Q1
```

```
WHERE (Q1.TRANSACTION_DATE < '2002-03-01') AND ('2002-01-01' <
        Q1.TRANSACTION_DATE)

Access Plan:
-----------
   Total Cost: 4960.79
   Query Degree:1

        Rows
       RETURN
       (   1)
        Cost
         I/O
         |
        31773
       TBSCAN
       (   2)
       4960.79
        3336
         |
       3e+006
 DP-TABLE: DB2ADMIN
       INVOICE


Extended Diagnostic Information:
--------------------------------

No extended Diagnostic Information for this statment.


Plan Details:
-------------


   1) RETURN: (Return Result)
      Cumulative Total Cost: 4960.79
      Cumulative CPU Cost: 5.04807e+008
      Cumulative I/O Cost: 3336
      Cumulative Re-Total Cost: 4960.79
      Cumulative Re-CPU Cost: 5.04805e+008
      Cumulative Re-I/O Cost: 3336
      Cumulative First Row Cost: 10.7846
      Estimated Bufferpool Buffers: 3336

      Arguments:
      ---------
      BLDLEVEL: (Build level)
         DB2 v9.1.200.98 : s070210
```

```
    STMTHEAP: (Statement heap size)
       2048

    Input Streams:
    -------------
       2) From Operator #2

          Estimated number of rows: 31773
          Number of columns: 4
          Subquery predicate ID: Not Applicable

          Column Names:
          ------------
          +Q2.CUSTNAME+Q2.AMOUNT+Q2.TRANSACTION_DATE
          +Q2.CUSTNO


  2) TBSCAN: (Table Scan)
    Cumulative Total Cost: 4960.79
    Cumulative CPU Cost: 5.04807e+008
    Cumulative I/O Cost: 3336
    Cumulative Re-Total Cost: 4960.79
    Cumulative Re-CPU Cost: 5.04805e+008
    Cumulative Re-I/O Cost: 3336
    Cumulative First Row Cost: 10.7846
    Estimated Bufferpool Buffers: 3336

    Arguments:
    ---------
    DPESTFLG: (Number of data partitions accessed are Estimated)
       FALSE
    DPLSTPRT: (List of data partitions accessed)
       3
    DPNUMPRT: (Number of data partitions accessed)
       1
    GLOBLOCK: (Global Lock intent)
       INTENT SHARE
    MAXPAGES: (Maximum pages for prefetch)
       ALL
    PREFETCH: (Type of Prefetch)
       SEQUENTIAL
    ROWLOCK : (Row Lock intent)
       NEXT KEY SHARE
    SCANDIR : (Scan Direction)
       FORWARD
    TABLOCK : (Table Lock intent)
       INTENT SHARE
    TBISOLVL: (Table access Isolation Level)
       CURSOR STABILITY
```

```
Predicates:
----------
2) Sargable Predicate
   Comparison Operator: Less Than (<)
   Subquery Input Required: No
   Filter Factor: 0.661248

   Predicate Text:
   --------------
   (Q1.TRANSACTION_DATE < '2002-03-01')

3) Sargable Predicate
   Comparison Operator: Less Than (<)
   Subquery Input Required: No
   Filter Factor: 0.349343

   Predicate Text:
   --------------
   ('2002-01-01' < Q1.TRANSACTION_DATE)

DP Elim Predicates:
------------------
Range 1)
   Stop  Predicate: (Q1.TRANSACTION_DATE < '2002-03-01')
   Start Predicate: ('2002-01-01' < Q1.TRANSACTION_DATE)

Input Streams:
-------------
   1) From Object DB2ADMIN.INVOICE

      Estimated number of rows: 3e+006
      Number of columns: 5
      Subquery predicate ID: Not Applicable

      Column Names:
      ------------
      +Q1.$RID$+Q1.CUSTNAME+Q1.AMOUNT+Q1.CUSTNO
      +Q1.TRANSACTION_DATE


Output Streams:
--------------
   2) To Operator #1

      Estimated number of rows: 31773
      Number of columns: 4
      Subquery predicate ID: Not Applicable
```

```
            Column Names:
            ------------
            +Q2.CUSTNAME+Q2.AMOUNT+Q2.TRANSACTION_DATE
            +Q2.CUSTNO


Objects Used in Access Plan:
---------------------------

   Schema: DB2ADMIN
   Name: INVOICE
   Type: Data Partitioned Table
         Time of creation: 2007-03-29-08.34.37.296002
         Last statistics update: 2007-03-29-09.05.17.937000
         Number of columns: 4
         Number of rows: 3000000
         Width of rows: 36
         Number of buffer pool pages: 30022
         Number of data partitions: 9
         Distinct row values: No
         Tablespace name: <VARIOUS>
         Tablespace overhead: 10.500000
         Tablespace transfer rate: 0.140000
         Source for statistics: Single Node
         Prefetch page count: 16
         Container extent page count: 16
         RCT pages: -1
         RCT full key cardinality: -1
         Index first key cardinality: -1
         Index first 2 keys cardinality: -1
         Index first 3 keys cardinality: -1
         Index first 4 keys cardinality: -1
```

4.3.3 Locking considerations

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 (lock object identifier name is TABLE_PART_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 the data partitions that are not in the result set. After 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 reacquiring 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 can be accessed, a non-intent lock might be chosen at the table level to prevent deadlocks between data partition locks from concurrent transactions.

You can obtain more information from the DB2 Information Center:

http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.udb.admin.doc/doc/c0021606.htm

4.3.4 Troubleshooting

You might encounter problems when creating tables, adding partitions, attaching partitions, or detaching partitions. Reasons for these problems can be:

- An ATTACH fails if the source table data does not conform to the range specified for the new partition in the target table.
- The source table must be an existing non-partitioned table or a partitioned table with only a single data partition.
- 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.
- The source table must not be a hierarchical table (typed table).
- The source table must not be a range-clustered table (RCT).
- The page size of table spaces must match. If they do not match, the result is message SQL1860.
- Detaching fails because you are detaching to an existing object.

The DB2 Information Center has more in-depth information about problems that you might encounter, restrictions, and usage guidelines and can be accessed by using this link:

http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.udb.admin.doc/doc/c0022680.htm

4.3.5 Using partitioned tables in your existing database

These are the approaches that you need to use to migrate an existing table or view to a partitioned table:

- Migrating regular tables: Use either one of the following approaches:
  - 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.
  - Unload the source table by 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.
- Migrating UNION ALL views:
  - Create a partitioned table with a single dummy data partition and then attach all of the tables.

  There is a requirement that the table spaces of the tables involved in this migration must have the same characteristics. They must all be LARGE or REGULAR, the same page size, and the same extent size.

Converting regular tables

To migrate data from a DB2 9.1 table into a partitioned table, you can use the LOAD command, use LOAD FROM ... CURSOR, or INSERT INTO SELECT ... FROM .... to migrate data to an empty partitioned table. Example 4-17 provides an example of converting a regular table to a partitioned table.

Example 4-17 Converting regular tables

```
-- Starting with a regular table TABLE1 defined with columns
-- COL1(INT) & COL2(INT) and already populated.
-- Create a new partitioned table, TABLE2 with the required matching
-- parameters.
--
   CREATE TABLE table11 (col1 INT, col2 INT) PARTITION BY RANGE
   (col1)(STARTING FROM FROM 0 ENDING AT 100 EVERY 5)

--Load the data from TABLE1 into TABLE2 using LOAD FROM ... CURSOR
--
   DECLARE c1 CURSOR FOR SELECT * FROM table1;
   LOAD FROM c1 OF CURSOR INSERT INTO table2 ;
```

```
--Or load the data from TABLE1 into TABLE2 using two step LOAD
--
   EXPORT TO TABLE1.DEL OF DEL SELECT * FROM table1;
   LOAD FROM TABLE1.DEL OF DEL INSERT INTO table2
```

If the any of the data is outside of the partition boundary, the load completes but with this message:

```
SQL0327N  The row cannot be inserted into table because it is outside
the bounds of the defined data partition ranges.  SQLSTATE=22525
```

Converting UNION ALL views

You can convert DB2 9.1 data in a UNION ALL view into a partitioned table. 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 (MQTs) must be re-created after the conversion.

One approach is to create a partitioned table with a single dummy data partition and ATTACH all of the tables of the UNION ALL view. Drop the dummy data partition after the first ATTACH to avoid any problems that might arise from overlapping ranges. Example 4-18 provides an example of converting a UNION ALL view to a partitioned table.

> **Note:** The data table spaces for a partitioned table must be either all SMS, all regular DMS, or all large DMS.

Example 4-18 Converting UNION ALL views

```
-- DDL of the tables included in the view

   CREATE TABLE invoice_20xx(CUSTNO BIGINT NOT NULL,
   transaction_date DATE NOT NULL,
   amount DECIMAL(15,2) NOT NULL,
   custname CHAR(10) NOT NULL);

-- DDL of the view that includes tables INVOICE_2000 through
-- INVOICE_2007

   CREATE VIEW invoice_v2000_2007 AS
   (SELECT * FROM invoice_2000 UNION ALL
   SELECT * FROM invoice_2001 UNION ALL
   SELECT * FROM invoice_2002 UNION ALL
   SELECT * FROM invoice_2003 UNION ALL
```

```
   SELECT * FROM invoice_2004 UNION ALL
   SELECT * FROM invoice_2005 UNION ALL
   SELECT * FROM invoice_2006 UNION ALL
   SELECT * FROM invoice_2007);

-- Create the partitioned table with the single dummy partition

   CREATE TABLE invoice(
   custno BIGINT NOT NULL,
   transaction_date DATE NOT NULL,
   amount DECIMAL(15,2),
   custname CHAR(10))
   PARTITION BY RANGE (transaction_date)
   (PART dummy STARTING FROM '01-01-1970' ENDING AT '01-01-1970');

-- Add the first partition

   ALTER TABLE invoice ATTACH PARTITION
   STARTING FROM '01-01-2000' ENDING AT '12-31-2000'
   FROM invoice_2000;

-- Detach the dummy partition and drop the resulting table

   ALTER TABLE invoice DETACH PARTITION dummy INTO dummy;
   DROP TABLE dummy;

-- Attach remaining partitions

   ALTER TABLE invoice ATTACH PARTITION
   STARTING FROM '01-01-2001' ENDING AT '12-31-2001'
   FROM INVOICE_2001;

   ALTER TABLE invoice ATTACH PARTITION
   STARTING FROM '01-01-2002' ENDING AT '12-31-2002'
   FROM invoice_2002;

   ALTER TABLE invoice ATTACH PARTITION
   STARTING FROM '01-01-2003' ENDING AT '12-31-2003'
   FROM invoice_2003;

   ALTER TABLE invoice ATTACH PARTITION
   STARTING FROM '01-01-2004' ENDING AT '12-31-2004'
   FROM invoice_2004;

   ALTER TABLE invoice ATTACH PARTITION
```

```
   STARTING FROM '01-01-2005' ENDING AT '12-31-2005'
   FROM invoice_2005;

   ALTER TABLE invoice ATTACH PARTITION
   STARTING FROM '01-01-2006' ENDING AT '12-31-2006'
   FROM invoice_2006;

   ALTER TABLE invoice ATTACH PARTITION
   STARTING FROM '01-01-2007' ENDING AT '12-31-2007'
   FROM invoice_2007;

-- Issue SET INTEGRITY on table

   SET INTEGRITY FOR invoice IMMEDIATE CHECKED;
```

4.3.6 Authorization levels

To alter a table to *attach* a data partition, the privileges held by the authorization ID of the statement must also include at least one of the following privileges or authorization levels on the source table:

- SELECT privilege on the table and DROPIN privilege on the schema of the table
- CONTROL privilege on the table
- SYSADM or DBADM authority

In addition, altering a table to *attach* a data partition requires at least one of the following privileges or authorization levels on the target table:

- ALTER and INSERT privileges on the table
- CONTROL privilege on the table
- SYSADM or DBADM authority

To alter a table to *detach* a data partition, the privileges held by the authorization ID of the statement must also include at least one of the following authorization levels or privileges on the target table of the detached partition:

- CREATETAB authority on the database and USE privilege on the table spaces used by the table, as well as one of these:
  - IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the new table does not exist
  - CREATEIN privilege on the schema, if the schema name of the new table refers to an existing schema

- SYSADM or DBADM authority

In addition, altering a table to *detach* a data partition requires at least one of the following privileges or authorization levels on the source table:

- SELECT, ALTER, and DELETE privileges on the table
- CONTROL privilege on the table
- SYSADM or DBADM authority

4.4 Best practices

In general, the following best practices apply to the table partitioning:

- Consider table partitioning for large fact tables in a data warehouse.
- Partition on date column.
- Make ranges match the size of the roll-out.
- Use DETACH for fast roll-out.
- Use table partitioning in combination with Database Partitioning Feature (DPF) and multi-dimensional clustering (MDC).
- Spread data across the table spaces.
- Use ADD plus LOAD for roll-in.

Define ranges

When defining ranges for your partitioned table, consider these recommendations:

- Partition on date.

  Fast roll-out by using DETACH is one of the greatest benefits of table partitioning. You need to partition on a date column to get this benefit. Side benefits include:

  – You receive performance benefits for common business intelligence (BI)-type queries.

    Many BI queries are date-oriented; therefore, you often get a performance boost from partition elimination.

  – Partitioning on a date column allows the separation of active and static data for reduced backup volume.

    You can tailor your backups to back up active data more frequently and static data less often.

- Fast roll-out by using DETACH reduces the need for REORG TABLE.

  One of the key reasons for reorganizing a table is to reclaim space after bulk deletes. If DETACH is used instead of bulk delete, this reason goes away.

- Choose the size of ranges to match the roll-out.

  The range size needs to match your roll-out strategy. Ranges typically are by month or quarter, which yields a manageable number of ranges. You can have ranges smaller than the required roll-out amount and roll-out several ranges at a time. For example, define the range by month and roll-out three months at a time each quarter.

Spreading data

Separation of indexes and table data is not at all required, but we recommend that you separate indexes and data to simplify space planning and your backup strategy. Advanced performance tuning often results in putting indexes in separate table spaces.

Index placement

For partitioned tables, indexes are separate objects. Different indexes of the same table can be placed in different table spaces. Individual indexes can then be reorganized independently. When placing the index, consider these actions:

- Separate indexes and table data to simplify size planning:
  - The size for each month's worth of table data is fairly predictable.
  - For space planning, putting the indexes and table data together is relatively more complicated than separating the index and the table data into different table spaces.
  - Separating indexes is necessary for certain performance tuning.
- Specify the index placement at CREATE INDEX time.

  If you do not specify the table space when you CREATE INDEX, it looks at what you specified for INDEX IN when you created the table. If you did not specify anything for INDEX IN, the index is placed in the table space that held the first range at the time the table was created. Because indexes tend to be much larger than one range of the table, this often causes you to run out of space unexpectedly.
- Default placement is undesirable, because all indexes end up together with one range.

Data placement for backup

Spreading ranges across more than one table space allows finer granularity of backups. Consider grouping current data for frequent backup and grouping historical data for infrequent backup.

The ability to spread large tables across multiple table spaces via table partitioning means that you can make each table space backup smaller. You still backup the same total amount of data, but you have flexibility to do it in more manageable chunks. Better yet, you can arrange the schedule to back up active data more frequently than historical data.

Restores TO END OF LOG can also be more granular. That is, you can recover from a disk failure by restoring only the affected table spaces. However, point-in-time roll-forward must include all table spaces related to the table.

By separating active and historical data, you can back up the historic data less frequently. Indexes that contain references to active data can be rebuilt rather than restored.

Smoother roll-in

To facilitate a smoother roll-in process, consider these actions:

- Issue COMMIT WORK after ATTACH and SET INTEGRITY:
  - ATTACH locks the whole table until committed.
  - New data is invisible after SET INTEGRITY is issued until committed.
- SET LOCK TIMEOUT WAIT:
  - Prevents SET INTEGRITY from failing on a lock conflict at the end.
- Plan for query draining by ATTACH:
  - ATTACH does not complete until it drains existing queries for the table.
  - Meanwhile, no new queries can start.
- Use a single SET INTEGRITY statement:
  - Include all refresh immediate MQTs and the base table in the same SET INTEGRITY statement.
  - MQTs that are not refreshed in the first pass go offline.
  - Multiple SET INTEGRITY statements can mean more passes through the data.
- Specify ALLOW WRITE ACCESS with SET INTEGRITY:
  - ALLOW NO ACCESS is faster because it gets an exclusive lock.
  - The default is the old, offline behavior.
  - ALLOW READ ACCESS is also available.
  - The trade-off is that higher availability options might run more slowly.

- Make use of exception tables:
  - Consider performing roll-in and roll-out together (also known as *rotate*).
  - ATTACH and DETACH in the same transaction minimizes the time that the table is unavailable.

5

Multi-dimensional clustering

This chapter provides information for planning, implementing, and administering multi-dimensional clustered (MDC) tables. In addition, we illustrate application programming techniques that leverage the components of the MDC technology.

© Copyright IBM Corp. 2007. All rights reserved.

5.1 Planning for the use of MDC on a table

Deciding to use MDC is a table-by-table decision. However, the choices that you make during the planning phase might affect other database decisions, particularly, the page size and extent size for the table spaces that will contain the MDC tables. Because you cannot alter page size and extent size after the table space is created (without dropping and recreating the table space), carefully consider the values that you select for the page size and the extent size.

5.1.1 Verify database configuration

If the MDC table resides in a system-managed (SMS) table space, be sure that multi-page file allocation is enabled. To check, use **db2 get db cfg for <*database*>** command. Databases created prior to DB2 9 might not have this feature enabled. Run command **db2empfa** to enable the feature. On a partitioned database, you must run **db2empfa** on each partition.

For additional information about the **db2empfa** command, see:

http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.udb.admin.doc/doc/r0002054.htm

5.1.2 Determine query workload

Knowledge of the current or planned workload of SQL queries on the table is a prerequisite to designing the MDC structure of a table. Even if your database is still in the planning stage, a general idea of the types of queries that you will use on the table is necessary. The goal is to recognize columns that are potential candidates for dimensions (for the definition of dimension, see “Dimension” on page 11). When looking for dimension candidates, focus on identifying columns that are:

- Of low cardinality

  Low cardinality is relative. In a table of 5,000,000 rows, a cardinality of 1000 is fairly low, but that same cardinality in a 5,000 row table is high.

- Relatively static in value

  If a dimension column changes value, extra processing is needed to move the row to an appropriate cell, update any row-level indexes, and possibly update the dimension block indexes.

- Frequently used in queries

  Because the primary benefit of using MDC tables is query performance, the more frequently that you use dimension columns in queries, the more performance gain is available.

5.1.3 Identify dimensions and columns

After you have a list of candidate columns, your next goal is to determine a set of dimension columns. You need to decide:

- The number of dimensions

  Every dimension that is defined on a table has its own dimension block index. Therefore, each dimension can add additional overhead to insert, update, and delete processing, but not to the same extent as a row-level index. A row-level index is updated every time that a row in the table is inserted or deleted, or when the key values are updated. A block index is updated only when a new extent is added to the table or the last row is removed from an extent. In most cases, from one to four dimensions is sufficient. MDC tables are not required to have more than one dimension, although the term “multi-dimensional” implies otherwise.

- The columns for each dimension

  Good candidates for dimension columns include:

  – Columns used for range, equality, and IN predicates
  – Roll-in or roll-out of data
  – Columns referenced in a GROUP BY clause
  – Columns referenced in an ORDER BY clause
  – Combinations of all of the previous candidates
  – Columns in a fact table that are foreign keys to dimension tables in a star schema

- The organization of any multi-column dimension

  A dimension can be composed of more than one column. The order of the columns in the dimension is critical to ensure that queries use the dimension block index effectively. For example, if the dimension consists of columns COL1 and COL2, in that order, query predicates (and ORDER BY or GROUP BY clauses) that reference COL2 need to also reference COL1. If not, the database server cannot start the index search at a known COL1 value but has to scan the dimension block index from start to finish, or bypass the block index completely.

5.1.4 Estimate space requirements

After you decide on a tentative list of dimensions, estimate the space requirements for the table and determine if the dimensions are suitable. This is a critical step in the process. If the dimensions that you have are inappropriate, the space requirements might be much higher than you expect. The basic steps are:

1. Gather the necessary statistics.
2. Estimate the average rows for a cell.
3. Determine candidate page size and extent size combinations.
4. Select a page size and extent size combination.
5. Estimate the average blocks for a cell.
6. Estimate the total data space required.

We discuss each of the steps next.

Gather the necessary statistics

In order to compute the space requirements, collect (or estimate) the following information:

- Cardinality of each dimension (how many distinct values exist)

  For new tables, you either estimate cardinality or base it on known information. For example, if you have a `STATE` column that contains the state abbreviations for U.S. customers, you can estimate that cardinality as `50`).

  For existing tables, use a query similar to Example 5-1.

  Example 5-1 Counting cardinality for dimensions

  ```
  SELECT     COUNT(DISTINCT col1), COUNT(DISTINCT col2), ...
             COUNT(DISTINCT coln)
  FROM       table
  ```

- Average row length

  For new tables, estimate the average row length based on known column lengths for fixed length columns, and either estimate average length or maximum length for variable length columns. For details about estimating an average row length, refer to the DB2 9 Information Center at:

  http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp

  For existing tables, make sure that the statistics are up-to-date (use RUNSTATS to update, if necessary). Ideally, reorganize the table and then use RUNSTATS to update statistics. Then, run REORGCHK CURRENT STATISTICS ON TABLE and compute the average row size as `TSIZE/CARD` (Table size divided by number of rows) from the output.

Alternatively, you can compute average row size by summing the AVGCOLLEN (average column length) column from the SYSCAT.COLUMNS view for all columns in the table, plus one for each nullable column, plus four for each variable length column. The SQL in Example 5-2 is a query that performs this computation.

Example 5-2 Computing average row size

```
SELECT SUM(AVGCOLLEN + (CASE NULLS WHEN 'Y' THEN 1 ELSE 0 END) +
(CASE TYPENAME WHEN 'VARCHAR' THEN 4 ELSE 0 END)) FROM
SYSCAT.COLUMNS WHERE TABSCHEMA='TPCD' AND TABNAME='CUSTOMER'
```

- Page size and extent (block) size

  For existing tables, you can get this information by using the following command and examining the output for the table space that the table uses:

  ```
  db2 list tablespaces show detail
  ```

  If you do not know the table space that is used, execute the following statement to determine the table space name:

  ```
  SELECT TBSPACE FROM SYSCAT.TABLES WHERE TABNAME='<table name>' AND
  TABSCHEMA='<table schema>'
  ```

- Number of rows

  For new tables, estimate the number of rows in the table using whatever information is available.

  For existing tables, use either CARD from the SYSCAT.TABLES entry for the table or execute a count on the table.

- Number of cells

  This is the number of unique combinations of the dimension columns. For a new table, estimate this number. You can compute a convenient upper limit for the number of cells by multiplying all of the expected cardinalities of the dimensions together. This assumes that the dimensions are independent, which might not be the case. If several of the dimensions are dependent, the estimate can be lowered. For example, if one dimension is month (cardinality 12), and another is day of the month (cardinality 31), the initial upper limit is 12 x 31 or 372. However, we know that not all months have 31 days, and we can confidently lower that upper limit to 366.

  For an existing table, you can get an accurate count of the number of cells with the SQL shown in Example 5-3 on page 182.

Example 5-3 Computing number of cells for an existing table

```
SELECT COUNT(*) FROM
(SELECT DISTINCT col1, col2, ... coln FROM tab1) cell_list
```

Determine candidate page size and extent size combinations

For a new table, estimate the average number of rows in a cell. The simplest approach is to divide the estimated row count by the estimated number of cells.

For existing tables, use the SQL in Example 5-4 to compute an average row count, as well as minimum, maximum, and standard deviation.

Example 5-4 Computing average rows per cell

```
SELECT     AVG(RowsPerCell) AS RowsPerCell,
           MIN(RowsPerCell) AS MinRowsPerCell,
           MAX(RowsPerCell) AS maxRowsPerCell,
           STDDEV(RowsPerCell) AS sdRowsPerCell
FROM       (SELECT     col1, col2, ... , colN, COUNT( * ) RowsPerCell
            FROM       table
            GROUP BY   col1, col2, ... , colN
           ) cell_table
```

Compute the maximum rows per page and extent size for various combinations. A table similar to Table 5-1 on page 183 might be useful. The formulas for the columns in the table are:

```
Rows in a page = (page size - 68) / (average row size + 10) rounded
down to an integer

Rows in extent = rows in a page x extent size

Number of extents needed for a cell = rows in cell / rows in extent
```

The probability of a cell smaller than one extent is the probability assuming a normal distribution of rows in a cell, using the average and standard deviation from the SQL in Example 5-4 and calculated using a cumulative distribution function. We use the Lotus® 1-2-3® spreadsheet function to calculate the probability:

```
@NORMAL(rows in extent;average rows in cell;stddev of rows in cell;0)
```

Stop the calculation for any entry that has a “rows in extent” greater than the average number of rows in a cell.

We developed a spreadsheet to perform these calculations. Lotus 1-2-3 and Microsoft Excel® versions of the spreadsheet are available for downloading from the IBM Redbooks publications Web site. See Appendix B, “Additional material” on page 243 for details about the download of additional material.

In our example, we use an average row size of 139 and an estimated rows in a cell of 686. Consequently, we stop considering extent sizes larger than 24 pages for the 4096 page size. The last column is computed by assuming a normal distribution of cell sizes with the average of 139 and standard deviation of 131.

Table 5-1 Computing rows in a page and extent

Page size (in bytes)	Rows in a page	Extent size (in pages)	Rows in extent	Extents needed for a cell	Probability of a cell smaller than one extent
4096	27	4	108	6.35	<0.005%
		8	216	3.18	0.02%
		12	324	2.12	0.29%
		16	432	1.59	2.63%
		20	540	1.27	13.25%
		24	648	1.06	38.59%
		28	756	0.91	70.35%
8192	54	4	216	3.18	0.02%
		8	432	1.59	2.63%
		12	648	1.06	38.59%
16384	109	4	436	1.57	2.82%
		8	872	0.79	92.22%
32768	219	4	876	0.78	92.65%

Select a page size and extent size combination

The ideal page size and extent size combination has one extent in a cell with a near-zero probability of cells smaller than one extent. However, the ideal situation rarely exists. The trade-off is between excess I/O and excess space.

After you construct a table similar to Table 5-1, examine it to see which page and extent combinations seem most reasonable. Our selections are a 4K page size with an extent size of either eight, or twelve pages, or an 8K page size with an

extent size of four pages. These selections offer a fairly small amount of I/O to read a cell combined with a very small probability of encountering cells occupying less than one extent.

If you are converting an existing table, compare your selections to the table's actual values. If they do not match, consider creating a table space with appropriate values and moving the table when you convert it. In this example, an existing table space with 4K page size and four pages in an extent is probably left unchanged.

Estimate average blocks for a cell

After you decide on a page size and extent size from the table, estimate the average blocks for a cell. The average number of blocks in a cell is equal to the total number of rows divided by the number of cells. Because we cannot have a partial row in a block, round the average number of blocks in a cell up to an integer value (3.49 becomes 4).

Estimate total data space required

The estimated total data space required, in blocks, is equal to number of blocks in a cell multiplied by the estimated number of cells. Convert this to pages by multiplying by the extent size.

For an existing table, compare this to the existing NPAGES from a REORGCHK to determine how much larger the converted table will be.

5.1.5 Adjust design as needed

If you discover that the dimensions you select leave too much wasted space in a block, you have several alternatives that might improve the average rows in a cell. The goal is to increase the average rows per cell and reduce wasted space:

- Eliminate the highest cardinality dimension column.

  This eliminates the column with the greatest impact on the number of cells.

- Replace a dimension column with a less granular column.

  For example, replace a column containing city names with a column containing state names (in the United States). Or, add a computed column to the table that replaces a date column with an integer column containing year and month. Example 5-6 on page 186 shows such a column.

- Reduce the block size.

  This does not reduce the number of active pages in the cell, but does waste fewer pages. It also increases the I/O activity required to read the entire cell, because the number of blocks increases.

If you are working with an existing table, this is a drastic step, because it requires dropping and recreating the table space.

- Reduce the page size.

  The only effect of reducing page size is to reduce the size of the wasted space in each cell, but it means more pages (and possibly more blocks) are required to store the data, increasing the I/O activity required to read the entire cell.

  If you are working with an existing table, this is a drastic step, because it requires dropping and recreating the table space.

- Reconsider the entire list of columns.

  If at first you do not succeed, try again.

5.1.6 DB2 Design Advisor

DB2 Design Advisor can recommend multi-dimensional clustering for an existing table. When using the command, it considers MDC tables and clustering indexes if the -m C option is specified. However, the DB2 Design Advisor does not consider MDC for empty tables. The table needs to have a sample load of data, comprising at least twelve extents.

DB2 Design Advisor does *not* determine average rows per cell or determine what the space utilization is. You need to continue to perform those calculations if you use the DB2 Design Advisor to recommend dimension keys.

5.2 Implementing MDC on a table

After you decide the dimensions, page size, extent size, and the space required, it is very easy to implement MDC on the table. First, create a table space with the required page size, extent size, and space.

To create an MDC table, use the ORGANIZE BY DIMENSIONS clause (the word DIMENSIONS is optional) of the CREATE TABLE statement to specify the columns of each dimension. Example 5-5 on page 186 shows the DDL for a regular (non-MDC) table.

Example 5-5 Regular table

```
CREATE TABLE mdc_samp (
       sales_amount DECIMAL(10,2) NOT NULL,
       date_of_sale DATE NOT NULL,
       salesperson CHAR(10) NOT NULL,
       store_nbr INTEGER,
       year_and_month GENERATED AS (INTEGER(date_of_sale)/100) )
```

Example 5-6 shows the DDL for the same table, as an MDC table with two dimensions. This command creates the table `MDC_SAMP`, two dimension block indexes (one index for `STORE_NBR` and one index for the computed column `YEAR_AND_MONTH`), and a consolidated block index containing all the dimension columns (`STORE_NBR` and `YEAR_AND_MONTH`).

Example 5-6 Creating a simple MDC table

```
CREATE TABLE mdc_samp (
       sales_amount DECIMAL(10,2) NOT NULL,
       date_of_sale DATE NOT NULL,
       salesperson CHAR(10) NOT NULL,
       store_nbr INTEGER,
       year_and_month GENERATED AS (INTEGER(date_of_sale)/100) )
       ORGANIZE BY DIMENSIONS (store_nbr, year_and_month)
```

To combine two or more columns into one dimension, enclose the columns for the dimension in parentheses, as shown in Example 5-7. Here, the two columns `STORE_NBR` and `SALESPERSON` are combined in a single dimension, with a second dimension on `YEAR_AND_MONTH`.

Example 5-7 Creating an MDC table with multi-column dimensions

```
CREATE TABLE mdc_samp (
       sales_amount DECIMAL(10,2) NOT NULL,
       date_of_sale DATE NOT NULL,
       salesperson CHAR(10) NOT NULL,
       store_nbr INTEGER,
       year_and_month GENERATED AS (INTEGER(date_of_sale)/100) )

ORGANIZE BY ((store_nbr,salesperson), year_and_month)
```

To change an existing table to use MDC, you must recreate the table with the ORGANIZE BY DIMENSIONS clause added to the CREATE TABLE statement. This might involve unloading data, dropping views, dropping the table, creating the table, reloading the data, recreating the views, and granting authorizations. A

more automated approach is to use the SYSPROC.ALTOBJ stored procedure, as shown in Example 5-8.

Example 5-8 Using SYSPROC.ALTOBJ to change the table to use MDC

```
CALL SYSPROC.ALTOBJ('APPLY_CONTINUE_ON_ERROR','
CREATE TABLE mdc_samp (
       sales_amount DECIMAL(10,2) NOT NULL,
       date_of_sale DATE NOT NULL,
       salesperson CHAR(10) NOT NULL,
       store_nbr INTEGER,
       year_and_month GENERATED AS (INTEGER(date_of_sale)/100) )
       ORGANIZE BY DIMENSIONS (store_nbr, year_and_month)
',-1,?)
```

SYSPROC.ALTOBJ captures the definition of all dependent objects, drops them, renames the table, creates the new table, loads the new table from the renamed old table, creates the dependent objects, and issues any necessary grants. Figure 5-1 shows the sample output from the procedure.

```
 Value of output parameters
 --------------------------
 Parameter Name  : ALTER_ID
 Parameter Value : 5

 Parameter Name  : MSG
 Parameter Value : SELECT OBJ_TYPE, OBJ_SCHEMA, OBJ_NAME,    SQL_OPERATION, SQL_STMT
RE ALTER_ID=5 AND EXEC_MODE LIKE '_1_____'  ORDER BY EXEC_SEQ
```

Figure 5-1 Output from SYSPROC.ALTOBJ

After the table has been altered, execute SYSPROC.ALTOBJ one more time to clean up the renamed old table. Using the ALTER_ID parameter value from the output of the previous step (In our example, 5), execute the SYSPROC.ALTOBJ call as shown in Example 5-9.

Example 5-9 Sample cleanup SYSPROC.ALTOBJ SQL

```
CALL SYSPROC.ALTOBJ('FINISH',",5,?)
```

After completion of these steps, be sure to execute RUNSTATS on the restructured table.

Important: ALTOBJ is very sensitive about the table name. It must be uppercase, preceded by a space, and followed by a space, as shown in Example 5-8.

For further information on ALTOBJ, refer to the ALTOBJ article in the DB2 9 Information Center at:

http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.udb.admin.doc/doc/r0011934.htm

5.3 Administering and monitoring MDC tables

This section describes the special considerations that MDC tables add to the administration and monitoring of the database.

5.3.1 Utilities

Although most utilities are unaffected by MDC tables, there are certain considerations and specific output changes.

LOAD

The SAVECOUNT option and TOTALFREESPACE file-type modifier are not supported with MDC tables. The ANYORDER option is required but, if not specified, is set on automatically.

To improve performance, consider increasing the database configuration parameter UTIL_HEAP_SIZE (utility heap size) in the database configuration. In addition, set database configuration parameters SORTHEAP and SHEAPTHRES_SHR to high values, because the load always includes a build phase to create the required dimension indexes.

You need to sort the data by the dimension columns before loading. Because of the need to build blocks and block indexes, the load is very sensitive to whether the data is grouped by the dimension columns. In one test, we discovered that the rows per second achieved with sorted data was about seven times greater than with unsorted data.

IMPORT

You cannot use the CREATE or REPLACE_CREATE options to load an MDC table.

REORG

You cannot use the ALLOW WRITE ACCESS option on the reorganization of an MDC table, unless you also specify the CLEANUP ONLY option.

Reorganization of MDC tables needs to be a very rare event, unless there is significant delete activity resulting in sparsely occupied blocks. Unlike a table with a clustered index, there is no degradation in the data clustering, because every row is always stored in a block with rows that have the same dimension column values.

RUNSTATS and REORGCHK

Although there are no changes in the RUNSTATS and REORGCHK commands, there are new statistics columns gathered for MDC tables and several changes in the computation of the ratios reported on REORGCHK. Figure 5-2 on page 190 shows the output from REORGCHK on an MDC table. Items of interest include:

- ACTBLK is the number of active blocks (blocks with data).
- MDC tables are designated with an asterisk (*) after the name.
- Block indexes are designated with an asterisk (*) after the name.
- The formula for F3 uses empty blocks and blocks with data instead of empty pages and pages with data.

```
Table statistics:

F1: 100 * OVERFLOW / CARD < 5
F2: 100 * (Effective Space Utilization of Data Pages) > 70
F3: 100 * (Required Pages / Total Pages) > 80

SCHEMA.NAME                      CARD     OV     NP     FP ACTBLK   TSIZE  F1  F2  F3 REORG
-------------------------------------------------------------------------------------------
Table: MDC.CUSTOMER*
                               300000      0  13667  16032    500 53400000   0 100 100 ---
-------------------------------------------------------------------------------------------

Index statistics:

F4: CLUSTERRATIO or normalized CLUSTERFACTOR > 80
F5: 100 * (Space used on leaf pages / Space available on non-empty leaf pages) > MIN(50, (100
0
F6: (100 - PCTFREE) * (Amount of space available in an index with one less level / Amount of
F7: 100 * (Number of pseudo-deleted RIDs / Total number of RIDs) < 20
F8: 100 * (Number of pseudo-empty leaf pages / Total number of leaf pages) < 20

SCHEMA.NAME                 INDCARD  LEAF ELEAF LVLS  NDEL    KEYS LEAF_RECSIZE NLEAF_RECSIZE
ERHEAD  F4  F5  F6  F7  F8 REORG
-------------------------------------------------------------------------------------------------
---------------------------------
Table: MDC.CUSTOMER
Index: MDC.PKCUST
                              300000  1251     0    3     0  300000            4             4
    710   5  92  14   0   0 *----
Index: SYSIBM.SQL070328112408740*
                                 500     2     0    2     0     125           14            14
    442 100 150   -   0   0 -----
Index: SYSIBM.SQL070328112409490*
                                 500     2     0    2     0      25            4             4
    710 100 107   -   0   0 -----
Index: SYSIBM.SQL070328112409560*
                                 500     2     0    2     0       5           10            10
    516 100  99   -   0   0 -----
-------------------------------------------------------------------------------------------------
---------------------------------
```

Figure 5-2 REORGCHK output for an MDC table

5.3.2 Monitoring MDC tables

The primary item to monitor on MDC tables is the size of the table. Because each unique combination of dimension column values is stored in separate blocks, the table can grow extremely quickly if several unique combinations are inserted. Unfortunately, the only way to reduce the space used in this case is to redesign the dimensioning. If new combinations are inserted, expect that more rows with the same values will be added at a later date, so that what is initially seen as “wasted space” can be considered “free space.”

5.3.3 Explain

MDC tables and block indexes add new types of data access to queries. Example 5-10 on page 191 shows the output from **db2expln** of a select against a non-MDC table. Compare it with Example 5-11 on page 192, which shows the output from **db2expln** of the same query against an MDC table.

Example 5-10 Query against a non-MDC table

```
Statement:

  select *
  from tpcd.customer
  where c_mktsegment='FURNITURE'


Section Code Page = 819

Estimated Cost = 13928.163086
Estimated Cardinality = 60000.000000

Coordinator Subsection - Main Processing:
   Distribute Subsection #1
   |  Directed to Single Node
   |  |  Node Number = 1
   Access Table Queue  ID = q1  #Columns = 8
   Return Data to Application
   |  #Columns = 8

Subsection #1:
   Access Table Name = TPCD.CUSTOMER  ID = 3,4
   |  #Columns = 8
   |  Relation Scan
   |  |  Prefetch: Eligible
   |  Lock Intents
   |  |  Table: Intent Share
   |  |  Row  : Next Key Share
   |  Sargable Predicate(s)
   |  |  #Predicates = 1
   |  |  Insert Into Asynchronous Table Queue  ID = q1
   |  |  |  Broadcast to Coordinator Node
   |  |  |  Rows Can Overflow to Temporary Table
   Insert Into Asynchronous Table Queue Completion  ID = q1

End of section
```

Example 5-11 Query against an MDC table

```
Statement:

  select *
  from mdc.customer
  where c_mktsegment='FURNITURE'


Section Code Page = 819

Estimated Cost = 3294.023438  (1)
Estimated Cardinality = 60000.000000

Coordinator Subsection - Main Processing:
   Distribute Subsection #1
   |  Directed to Single Node
   |  |  Node Number = 1
   Access Table Queue  ID = q1  #Columns = 8
   Return Data to Application
   |  #Columns = 8

Subsection #1:
   Access Table Name = MDC.CUSTOMER  ID = 3,5
   |  Index Scan:  Name = SYSIBM.SQL070405161312650  ID = 3
   |  |  Dimension Block Index  (2)
   |  |  Index Columns:
   |  |  |  1: C_MKTSEGMENT (Ascending)
   |  #Columns = 8
   |  Clustered by Dimension for Block Index Access  (3)
   |  #Key Columns = 1
   |  |  Start Key: Inclusive Value
   |  |  |  |  1: 'FURNITURE '
   |  |  Stop Key: Inclusive Value
   |  |  |  |  1: 'FURNITURE '
   |  Data Prefetch: None  (4)
   |  Index Prefetch: None
   |  Lock Intents
   |  |  Table: Intent Share
   |  |  Block: Intent Share  (5)
   |  |  Row  : Next Key Share
   |  Sargable Predicate(s)
   |  |  Insert Into Asynchronous Table Queue  ID = q1
   |  |  |  Broadcast to Coordinator Node
   |  |  |  Rows Can Overflow to Temporary Table
   Insert Into Asynchronous Table Queue Completion  ID = q1

End of section
```

Descriptions of the items worth noting are:

- (1) Estimated timerons

  Remember, *timerons* are an artificial measure of execution cost used by the optimizer to select between access paths. A timeron cannot be used to compare results between systems. However, within the same system in the same operational environment, a timeron *can* be a valid form of comparison between queries.

- (2) Block index selection

  If an MDC index is selected for use, it is identified as shown.

- (3) Index keys used

  Index key fields of an MDC appear in the same manner as other indexes.

- (4) Prefetch status

  Prefetch status here is None, which means that the optimizer does not see an advantage to prefetching several blocks of data at a time. In the non-MDC table, prefetch was Eligible, because the server was planning to scan the entire table. Because of the clustering caused by the blocks, there is no guarantee that prefetching is useful. Every block contains rows to be selected compared to a percentage of the extents in the non-MDC table.

- (5) Locks

  A new level of locking, block level, is introduced with MDC tables.

5.4 Application considerations for MDC tables

The use of MDC tables is generally transparent to the application program. There is almost nothing that an application must do to specifically exploit them.

However, the application can make use of the way that MDC handles delete processing under certain specific conditions. There is an Information Center article that outlines these conditions at:

http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.udb.admin.doc/doc/c0007338.htm

For the application programmer, the prime consideration is that the WHERE clause on the DELETE either does not exist or references only MDC dimension columns. In that case, the delete operation does not physically delete and log each row in the cells involved, but only modifies a few bytes in each page. In addition, only these small modifications are logged, along with any index maintenance and block map modification. This results in a significant reduction in

both processing time and log space requirements when the number of affected rows is large.

Applications must not make massive updates to an MDC dimension column. Updating an MDC dimension column causes the following actions to occur (at a minimum):

- The row is deleted from its current page.

  Because the row's dimension columns no longer match the values for the cell in which it resides, it must be moved to a different cell.

- The row is placed on another page.

  In the worst case, this includes creating a new cell. In that case, the dimension block indexes and consolidated block index are updated. At best, this process is the same as a new row insertion into an existing cell with room in a block.

- Any row-level indexes are updated to reflect the new position of the row.

  The row's row ID changes, because it now resides on a different page. All row-level indexes for the table contain the old row ID of the row.

5.5 Examples of using MDC

This section offers examples of MDC tables. Any experiences in performance improvement are particular to the example cited and are not predictive of what improvements (if any) that you might experience in your particular environment.

5.5.1 Applying MDC to the TPC customer table

To show how the MDC calculations are applied and the results achieved in a test environment, we converted the customer table defined in TPC-D to use MDC. Any performance results noted in this section are from a specialized test environment and are for illustrative purposes *only*. Your results will vary.

The customer table was originally defined with the DDL shown in Example 5-12 on page 195. The data (300,000 rows) was loaded after being sorted into `c_mktsegment, c_nationkey order`.

Example 5-12 DDL to create customer table

```
CREATE TABLE tpcd.customer  (
                  c_custkey INTEGER NOT NULL ,
                  c_name VARCHAR(25) NOT NULL ,
                  c_address VARCHAR(40) NOT NULL ,
                  c_nationkey INTEGER NOT NULL ,
                  c_phone CHAR(15) NOT NULL ,
                  c_acctbal DECIMAL(15,2) NOT NULL ,
                  c_mktsegment CHAR(10) NOT NULL ,
                  c_comment VARCHAR(117) NOT NULL )
                 IN tbsp1 ;

CREATE INDEX tpcd.ndxcust ON tpcd.customer(c_mktsegment,c_nationkey)
CLUSTER;
CREATE UNIQUE INDEX tpcd.pkcust ON tpcd.customer (c_custkey);
ALTER TABLE tpcd.customer ADD PRIMARY KEY(c_custkey);
```

`TBSP1` is a single-partition table space with a page size of 4096 and extent size of 32 pages.

Using the techniques from section 5.1, “Planning for the use of MDC on a table” on page 178, we select the dimension columns and estimate the space requirements:

- Determine the dimension column.

  The C_NATIONKEY and C_MKTSEGMENT columns were chosen as the dimension columns. The choice was arbitrary, because this is an artificially generated table with no application use. However, these were two columns with very low cardinalities and an argument can be made that they are frequently used in queries, particularly queries performing analysis by country or market segment.

  Example 5-13 shows the query to check the cardinality of a column: C_MKTSEGMENT had a cardinality of `5`, and C_NATIONKEY had a cardinality of `25`.

  Example 5-13 Checking cardinality

```
SELECT COUNT(DISTINCT c_mktsegment) c_mktsegment,COUNT(DISTINCT
c_nationkey) c_nationkey FROM tpcd.customer

C_MKTSEGMENT C_NATIONKEY
------------ -----------
           5          25
1 record(s) selected.
```

- Estimate the space required:
  - Calculate the average row size.

    Example 5-14 illustrates how to calculate the average row size. For our example table, it is 180 bytes.

Example 5-14 Calculating row size

```
SELECT SUM(AVGCOLLEN + (CASE NULLS WHEN 'Y' THEN 1 ELSE 0 END) +
(CASE TYPENAME WHEN 'VARCHAR' THEN 4 ELSE 0 END)) FROM
SYSCAT.COLUMNS WHERE TABSCHEMA='TPCD' AND TABNAME='CUSTOMER'
1
-----------
        180
   1 record(s) selected
```

  - The actual number of cells is 125. Rows per cell ranged between 2298 and 2518, with an average of 2400 and a standard deviation of 50.9. Example 5-15 shows the SQL query to obtain this information.

Example 5-15 Calculating cells and rows per cell

```
SELECT COUNT(*) cells,AVG(RowsPerCell) AS RowsPerCell,
MIN(RowsPerCell) AS MinRowsPerCell, MAX(RowsPerCell) AS
maxRowsPerCell, STDDEV(RowsPerCell) AS sdRowsPerCell FROM ( SELECT
c_nationkey,c_mktsegment,COUNT(*) RowsPerCell FROM tpcd.customer
GROUP BY c_nationkey,c_mktsegment ) cell_table

CELLS       ROWSPERCELL MINROWSPERCELL MAXROWSPERCELL SDROWSPERCELL
----------- ----------- -------------- -------------- --------------
        125        2400           2298           2518 +5.092645E+001
```

  - The existing table had a page size of 4096 and an extent size of 32 pages. Based on the average rows in a cell and the standard deviation, there is a less than 0.005% probability of having a cell smaller than one extent using these values for almost any page size and extent size selected.
  - The page size used is 4096. With that average, the rows per page are (4096 - 68)/(10 + 180) = 21.
  - The extent size used is 32K. The maximum rows per block and extent are 21 x 32 = 672.
  - The average number of blocks/cell are 2400/672 = 3.6 (therefore, using 4 blocks).
  - Total estimated blocks required was 4 x 125 = 500.

A new table, MDC.CUSTOMER, was created by using the DDL in Example 5-16 and the data copied from TPCD.CUSTOMER.

Example 5-16 Creating the customer table with MDC

```
CREATE TABLE mdc.customer (
   c_custkey INTEGER NOT NULL ,
   c_name VARCHAR(25) NOT NULL,
   c_address VARCHAR(40) NOT NULL,
   c_nationkey INTEGER NOT NULL,
   c_phone CHAR(15) NOT NULL,
   c_acctbal DECIMAL(15, 2) NOT NULL,
   c_mktsegment CHAR(10) NOT NULL ,
   c_comment VARCHAR(117) NOT NULL)
ORGANIZE BY (c_mktsegment, c_nationkey)
IN tbsp1 ;

CREATE UNIQUE INDEX  mdc.pkcust ON mdc.customer (c_custkey);

ALTER TABLE mdc.customer ADD PRIMARY KEY(c_custkey);
```

After executing REORGCHK, we compared the space requirements of the two tables, which are summarized in Table 5-2 on page 198, and the space requirements of the indexes, which are summarized in Table 5-3 on page 198, and noted:

- The number of pages with data (NPAGES) was slightly higher for the MDC table.

  The original table had 13609 full pages and one partial. The MDC table has a partial page in the last block for each cell.

- The full number of pages (FPAGES) matched our estimated requirement.

  The extra 32 pages is because the first block of an MDC table is reserved.

- The block indexes on the MDC table took significantly less space than the cluster index on the original table.

  The cluster index is a row-level index, so each entry contains the key and the list of row IDs with that key. It is this volume of entries that drives the number of leaf pages to 600-plus and the index levels to three. In contrast, each of the block index entries contains the key and list of the blocks that contain rows for that key value. This index is much smaller and only has two levels.

Table 5-2 Comparison of non-MDC and MDC customer tables

REORGCHK data	TPCD.CUSTOMER	MDC.CUSTOMER
CARD	300,000	300,000
NPAGES	13,610	13,667
FPAGES	13,611	16,032
ALTBLK	N/A	500
TSIZE	53,400,000	53,400,000

Table 5-3 Comparison of indexes for non-MDC and MDC customer tables

REORGCHK data	TPCD.CUSTOMER	MDC.CUSTOMER
	NDXCUST clustering index	Composite block index
INDCARD	300,000	500
LEAF	679	2
LVLS	3	2
KEYS	125	125
		C_NATIONKEY dimension index
INDC ARD		500
LEAF		2
LVLS		2
KEYS		25
		C_MKTSEGMENT dimension index
INDCARD		500
LEAF		2
LVLS		2
KEYS		5

After creating the tables, the SQL shown in Example 5-17 on page 199 was executed 50 times using **db2batch** to compare performance.

Example 5-17 SQL for comparing non-MDC table to MDC table

```
--#BGBLK 50
SELECT     * FROM tpcd.customer WHERE c_nationkey = 0;
SELECT     * FROM tpcd.customer WHERE c_mktsegment = 'FURNITURE';
SELECT     * FROM tpcd.customer
WHERE      c_mktsegment = 'FURNITURE'
           AND c_nationkey = 0;
SELECT     * FROM mdc.customer WHERE c_nationkey = 0;
SELECT     * FROM mdc.customer WHERE c_mktsegment = 'FURNITURE';
SELECT     * FROM mdc.customer
WHERE      c_mktsegment = 'FURNITURE'
           AND c_nationkey = 0;
--#EOBLK
```

Again, we remind you that this database was in a lab environment. Do not use the results that we received to predict the performance in your environment. We also ran explains on the SQL. Timings and information gleaned from the explains are shown in Table 5-4 on page 200.

Note that the non-MDC table is in pristine condition where the clustering index is concerned. The data was loaded in cluster sequence and no updates had taken place. As the table is updated, the effectiveness of the clustering decreases until the table is reorganized. In the MDC table, the data is kept clustered, because each cell only contains one combination of clustering column values. So, reorganizing the table to re-cluster the data is unnecessary.

Table 5-4 Comparison of queries

Where clause of query (select * from customer where ...)	Statistic	Non-MDC	MDC
where c_nationkey = 0	Average time (proportional to non-MDC average)	1.00	0.44
	Minimum time (proportional to non-MDC average)	0.42	0.46
	Maximum time (proportional to non-MDC average)	1.87	0.53
	Basic access plan	Relational scan of table	Uses dimension block index for c_nationkey
where c_mktsegment = 'FURNITURE'	Average time (proportional to non-MDC average)	1.00	0.51
	Minimum time (proportional to non-MDC average)	0.79	0.48
	Maximum time (proportional to non-MDC average)	2.27	0.58
	Basic access plan	Uses clustering index	Uses dimension block index for c_mktsegment

Where clause of query (select * from customer where ...)	Statistic	Non-MDC	MDC
where c_mktsegment = 'FURNITURE' and c_nationkey = 0	Average time (proportional to non-MDC average)	1.00	0.23
	Minimum time (proportional to non-MDC average)	0.97	0.21
	Maximum time (proportional to non-MDC average)	3.83	0.26
	Basic access plan	Uses clustering index	Uses consolidated block index

5.5.2 Utilizing both dimension and row-level indexes

Another table in the TPC database, `LINEITEM`, provides an opportunity to examine how to use dimension and row-level indexes together to achieve better query performance. Example 5-18 shows the original `LINEITEM` table definition.

Example 5-18 Original LINEITEM table

```
CREATE TABLE tpcd.lineitem ( l_orderkey    INTEGER NOT NULL,
                             l_partkey     INTEGER NOT NULL,
                             l_suppkey     INTEGER NOT NULL,
                             l_linenumber  INTEGER NOT NULL,
                             l_quantity    DECIMAL(15,2) NOT NULL,
                             l_extendedprice  DECIMAL(15,2) NOT NULL,
                             l_discount    DECIMAL(15,2) NOT NULL,
                             l_tax         DECIMAL(15,2) NOT NULL,
                             l_returnflag  CHAR(1) NOT NULL,
                             l_linestatus  CHAR(1) NOT NULL,
                             l_shipdate    DATE NOT NULL,
                             l_commitdate  DATE NOT NULL,
                             l_receiptdate DATE NOT NULL,
                             l_shipinstruct CHAR(25) NOT NULL,
                             l_shipmode     CHAR(10) NOT NULL,
                             l_comment      VARCHAR(44) NOT NULL)
                           IN TBSP1;
```

DB2 Design Advisor was executed against the table using the workload, which is shown in Example 5-19, and the following command:

```
db2advis -d doug -i tpcd.workload.advis.sql -m C -o tpcd.advis.results
```

The resulting output is shown in Example 5-20.

Example 5-19 Workload for DB2 Design Advisor

```
SELECT * FROM tpcd.lineitem
WHERE l_shipmode IN ('AIR','RAIL','FOB')
AND l_shipdate = '1997-01-01'
AND l_partkey BETWEEN 2000 AND 5000;

SELECT * FROM tpcd.lineitem
WHERE l_shipdate BETWEEN '1997-01-01' AND '1997-06-01';

SELECT * FROM tpcd.lineitem
WHERE l_shipmode IN ('AIR','REG AIR');

SELECT * FROM tpcd.lineitem
WHERE l_shipdate = '1997-01-01'
AND l_partkey BETWEEN 2000 AND 5000;

SELECT * FROM tpcd.lineitem
WHERE l_shipdate BETWEEN '1997-01-01' AND '1997-12-31'
AND l_shipmode='RAIL'
AND l_partkey BETWEEN 2000 AND 5000;
```

Example 5-20 DB2 Design Advisor output

```
found [5] SQL statements from the input file
Recommending Multi-Dimensional Clusterings...
total disk space needed for initial set [   0.122] MB
total disk space constrained to         [3052.923] MB

Note: MDC selection in the DB2 Design Advisor requires the target
database to be populated with a data sample. This sample is used for
estimating the number and density of MDC cells in any MDC solution that
the DB2 Design Advisor will recommend. If your database is empty, the
DB2 Design Advisor will not recommend MDC.


Prioritizing Multi-dimensional Clustering candidate tables...
Multi-dimensional Clustering candidate tables, in priority sequence:
```

```
Table 0: LINEITEM,
   number of pages 415425,
   block size 32
There are 1 candidate tables considered for Multi-dimensional
Clustering conversion

Searching the multi-dimensional space for solutions for LINEITEM...

Percentage of search points visited...
100

2 clustering dimensions in current solution
[620904.0000] timerons (without any recommendations)
[194247.0744] timerons (with current solution)
[68.72%] improvement

--
--
-- LIST OF MODIFIED CREATE-TABLE STATEMENTS WITH RECOMMENDED
PARTITIONING KEYS AND TABLESPACES AND/OR RECOMMENDED MULTI-DIMENSIONAL
CLUSTERINGS
-- ==========================
-- CREATE TABLE "TPCD "."LINEITEM" ( "L_ORDERKEY" INTEGER NOT NULL ,
--       "L_PARTKEY" INTEGER NOT NULL ,
--       "L_SUPPKEY" INTEGER NOT NULL ,
--       "L_LINENUMBER" INTEGER NOT NULL ,
--       "L_QUANTITY" DECIMAL(15,2) NOT NULL ,
--       "L_EXTENDEDPRICE" DECIMAL(15,2) NOT NULL ,
--       "L_DISCOUNT" DECIMAL(15,2) NOT NULL ,
--       "L_TAX" DECIMAL(15,2) NOT NULL ,
--       "L_RETURNFLAG" CHAR(1) NOT NULL ,
--       "L_LINESTATUS" CHAR(1) NOT NULL ,
--       "L_SHIPDATE" DATE NOT NULL ,
--       "L_COMMITDATE" DATE NOT NULL ,
--       "L_RECEIPTDATE" DATE NOT NULL ,
--       "L_SHIPINSTRUCT" CHAR(25) NOT NULL ,
--       "L_SHIPMODE" CHAR(10) NOT NULL ,
--       "L_COMMENT" VARCHAR(44) NOT NULL   ,
--      MDC704121438030000 GENERATED ALWAYS AS (
((INT(L_SHIPDATE))/16)) )
-- ORGANIZE BY (
--      MDC704121438030000,
--      L_SHIPMODE )
-- ;
-- COMMIT WORK ;
```

```
158 solutions were evaluated by the advisor
DB2 Workload Performance Advisor tool is finished.
```

Three modifications were made to the advisor output:

- `L_DAYS_IN_TRANSIT` was added to the table as a generated column and defined as a dimension column.

  This column was added to provide an example of a non-monotonic function and to create a third dimension column so that the consolidated block index was not used when the WHERE clause mentioned only some of the dimensions. This was for testing purposes only to show how DB2 utilizes index ANDing for block and row-level indexes.
- A meaningful name (`L_SHIP_YYMM`) was substituted for the generated column name.
- `L_SHIP_YYMM` used a different divisor from that suggested by the DB2 Design Advisor to provide an integer that contains the year and the month of the ship date.

  The net effect of this divisor change is to increase the number of rows in a cell.

Example 5-21 is the modified table.

Example 5-21 DDL for LINEITEM table with MDC and regular index

```
CREATE TABLE "MDC    "."LINEITEM"  (
      "L_ORDERKEY" INTEGER NOT NULL ,
     "L_PARTKEY" INTEGER NOT NULL ,
     "L_SUPPKEY" INTEGER NOT NULL ,
     "L_LINENUMBER" INTEGER NOT NULL ,
     "L_QUANTITY" DECIMAL(15,2) NOT NULL ,
     "L_EXTENDEDPRICE" DECIMAL(15,2) NOT NULL ,
     "L_DISCOUNT" DECIMAL(15,2) NOT NULL ,
     "L_TAX" DECIMAL(15,2) NOT NULL ,
     "L_RETURNFLAG" CHAR(1) NOT NULL ,
     "L_LINESTATUS" CHAR(1) NOT NULL ,
     "L_SHIPDATE" DATE NOT NULL ,
     "L_COMMITDATE" DATE NOT NULL ,
     "L_RECEIPTDATE" DATE NOT NULL ,
     "L_SHIPINSTRUCT" CHAR(25) NOT NULL ,
     "L_SHIPMODE" CHAR(10) NOT NULL ,
     "L_COMMENT" VARCHAR(44) NOT NULL,
      l_days_in_transit GENERATED AS (DAYS(l_receiptdate) -
      DAYS(l_shipdate)),
    l_ship_YYMM GENERATED AS (INTEGER(l_shipdate)/100) )
```

```
    ORGANIZE BY (l_shipmode, l_ship_yymm, l_days_in_transit)
     IN "TBSP1" ;

CREATE INDEX mdc.ndx_l_partnbr ON mdc.lineitem (l_partkey)
   ALLOW REVERSE SCANS;
```

As we noted previously, this table includes both a monotonic and a non-monotonic function. Monotonic functions increase as the variables in the function increase. The function for `L_DAYS_IN_TRANSIT` is non-monotonic, because increasing one or both dates does not necessarily mean that the function value increases. For example, if the dates were `2001-01-31` for `L_SHIPDATE` and `2001-02-05` for `L_RECEIPTDATE`, the function computes a value of `5`. Increase `L_SHIPDATE` to `2001-02-01`, and the function computes a value of `4`. The function for `L_SHIP_YYMM` is monotonic, which is readily apparent by noting that division or multiplication by a positive constant always yields a value that increases as the original number increases.

When a column defined as a monotonic function on a base column is used as a dimension, DB2 can use predicates on the base column to compute predicates on the dimension column. For example, a predicate of `L_SHIPDATE BETWEEN '2001-02-05' AND '2001-03-02'` can be translated by the database server into a range predicate of `L_SHIP_YYMM BETWEEN 200102 AND 200103`.

Some sample queries and explanations of their access paths:

- `SELECT * FROM mdc.lineitem WHERE l_shipmode IN ('AIR', 'RAIL', 'FOB') AND l_days_in_transit IN(5,6,7,8,9,10) AND l_partkey BETWEEN 2000 AND 5000;`

  Use the block index for `L_SHIPMODE` and the block index for `L_DAYS_IN_TRANSIT` to get two lists of block IDs. AND those two lists to select block IDs that are in both lists. Match that list to the list of row IDs found from the `PARTKEY` index. Use the resultant list of row IDs to fetch data rows.

- `SELECT * FROM mdc.lineitem WHERE l_shipdate BETWEEN '1997-01-01' AND '1997-06-01' AND l_partkey BETWEEN 2000 AND 5000;`

  Use the block index for `L_SHIP_YYMM` limited by keys between `199701` and `199706` to get the block IDs. Match the block IDs with the row IDs from the `PARTKEY` index limited by keys between `2000` and `5000`.

- `SELECT * FROM mdc.lineitem WHERE YEAR(l_shipdate) = 1997 AND l_partkey BETWEEN 2000 AND 5000;`

  Use the `PARTKEY` index limited by a key between `2000` and `5000`. `L_SHIPDATE` values are resolved as SARGable predicates (tested after the row is read). This query did *not* use the dimension index.

- `SELECT * FROM mdc.lineitem WHERE l_shipdate BETWEEN '1997-01-01' AND '1997-12-31' AND l_partkey BETWEEN 2000 AND 5000;`

  Use the dimension index on `L_SHIP_YYMM` limited by a key between `199701` and `199712`, matching the block IDs to the row IDs from the `PARTKEY` index limited by a key between `2000` and `5000`.

- `SELECT * FROM mdc.lineitem WHERE l_shipdate = '1997-01-01' AND l_partkey BETWEEN 2000 AND 5000;`

  Use the dimension index on `L_SHIP_YYMM` limited by key = `199701`, matching block IDs to row IDs from the `PARTKEY` index limited by a key between `2000` and `5000`.

5.5.3 Using the Control Center to run DB2 Design Advisor

You can invoke DB2 Design Advisor through the Control Center. This section shows the major steps in that process. It assumes that you have cataloged the database on your client machine. The steps are:

1. Open the list of databases in the Control Center.
2. Select your database and then select the Design Advisor (**Selected** → **Design Advisor**). See Figure 5-3 on page 207.

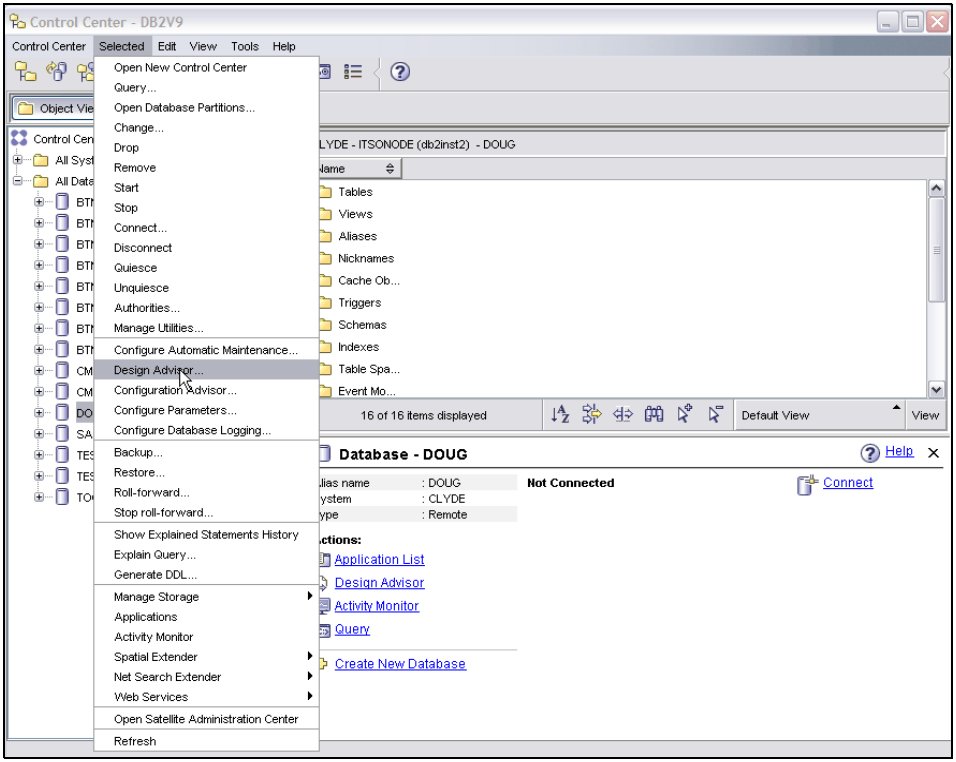


Figure 5-3 Selecting Design Advisor

3. Click **Next** on the introduction page.
4. Select the desired performance features to consider. In our case, only MDC is selected. Click **Next**.
5. The workload panel that is displayed depends on whether a workload has previously been defined for the table. In our case, there is no previous workload, so we are presented with a panel to enter a workload name. See Figure 5-4 on page 208. The workload name is not significant to the process; it is used to recall the workload later. You can either import a workload from another source or add the individual SQL statements for the workload to be analyzed. We have our workload in a text file, so we click **Import**.

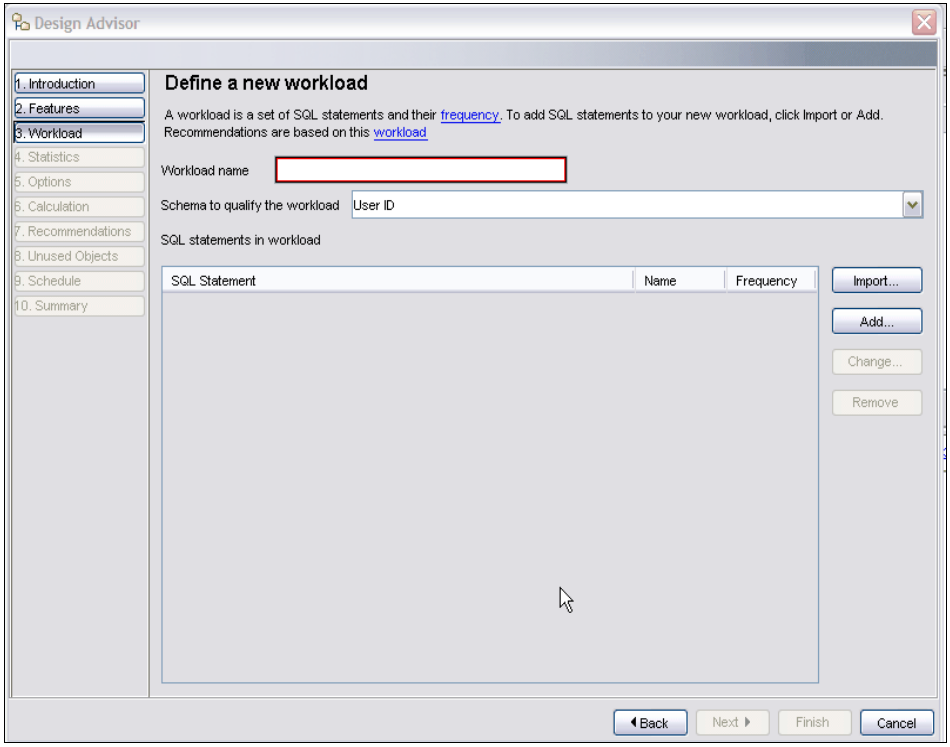


Figure 5-4 Workload definition page

6. You can import SQL from various sources, including a workload file, recently explained statements (in the EXPLAIN tables of the database), or recently executed SQL (from a dynamic SQL snapshot). In our example, we chose **Workload file**, entered our file name, and clicked **Load file**. See Figure 5-5 on page 209.

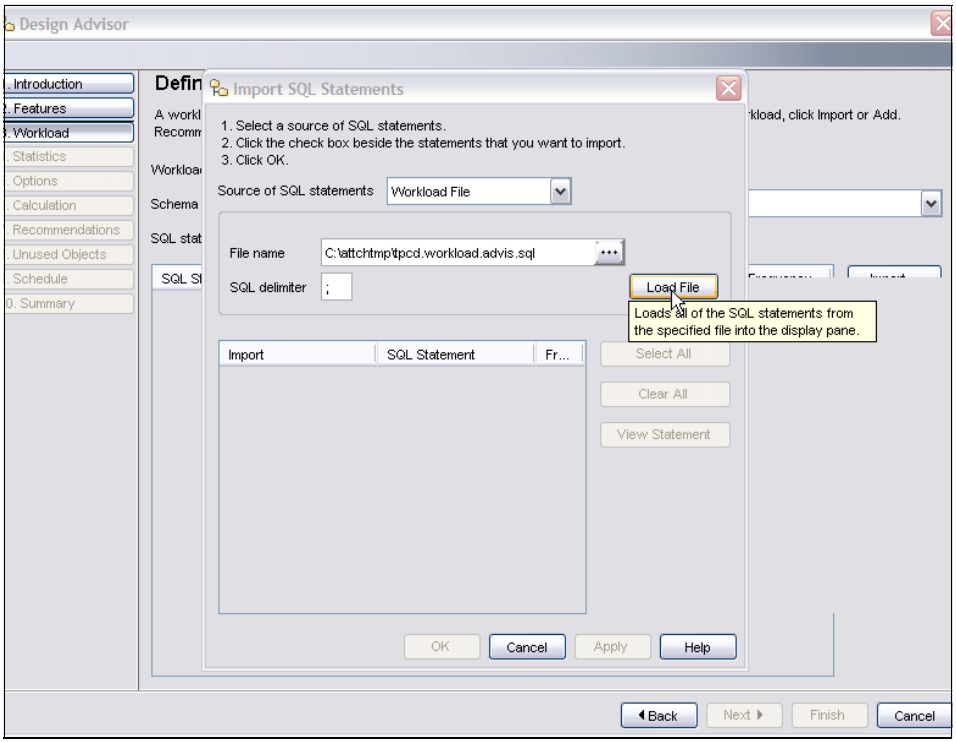


Figure 5-5 Selecting the workload file

7. The statements in the workload file are displayed. We selected all the statements and clicked **OK**.
8. Back on the workload panel, we clicked **Next**, because we did not want to change any of the statements or their relative execution frequency.
9. We do not need to calculate statistics on the tables in the workflow, because all of our statistics are current. Click **Next** to proceed.
10. Choose any options needed on the Options panel of the Design Advisor. We do not want to limit the disk space utilized for MDC, so do not select any options and just click **Next**.
11. On the Calculation panel, select **Now** for when to execute analysis and click **Next**.
12. On the recommendations panel (Figure 5-6 on page 210), a list of the tables to alter is displayed. Click on the selection box to select which table changes are accepted and click **Show DDL** to see the DDL generated by the Design Advisor. You can also click **Show workload detail** to see the estimated savings (in timerons) for each statement in the workload. Click **Next** to proceed to the next step.

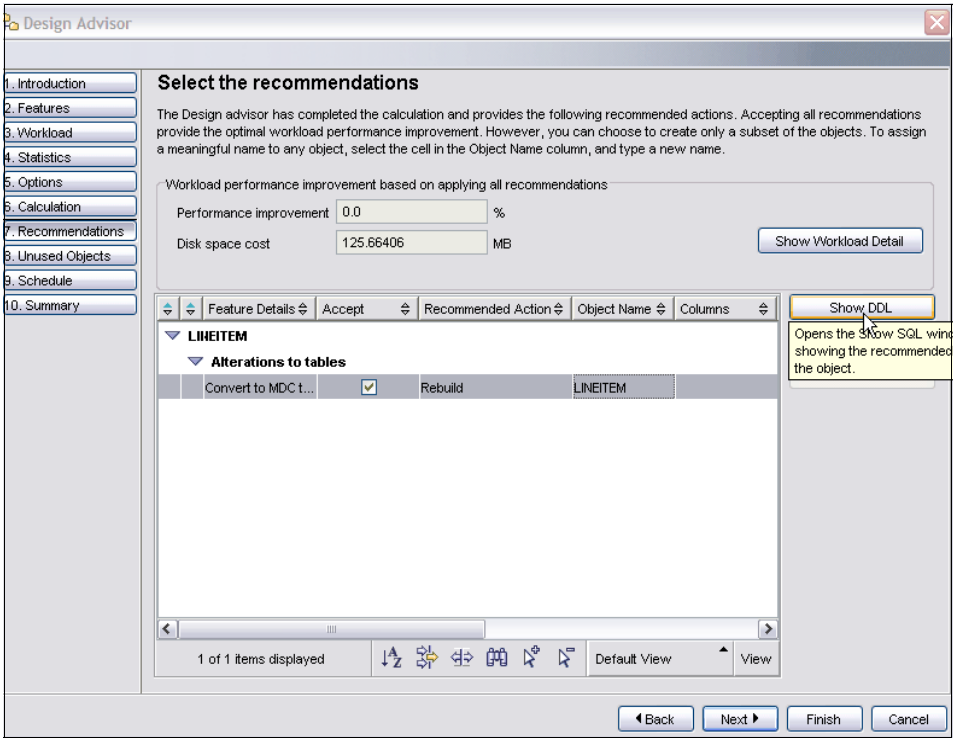


Figure 5-6 Design Advisor recommendations

13. The Unused Objects panel shows the list of database objects (usually indexes) that were not referenced in the workload. If there are items listed that can be removed, you might mark them here by clicking the box in the **Accept** column for the object. Be certain that other SQL statements are not impacted by the removal. Click **Next** to continue.

14. The Schedule panel (Figure 5-7 on page 211) is where you can choose to save the DDL necessary to convert the table, and either execute the DDL immediately or schedule it for later execution. Click **Cancel** if you do not want to execute the DDL. Otherwise, make a scheduling selection and click **Next** for a final summary panel or **Finish** to exit the Design Advisor and proceed to executing the DDL as scheduled.

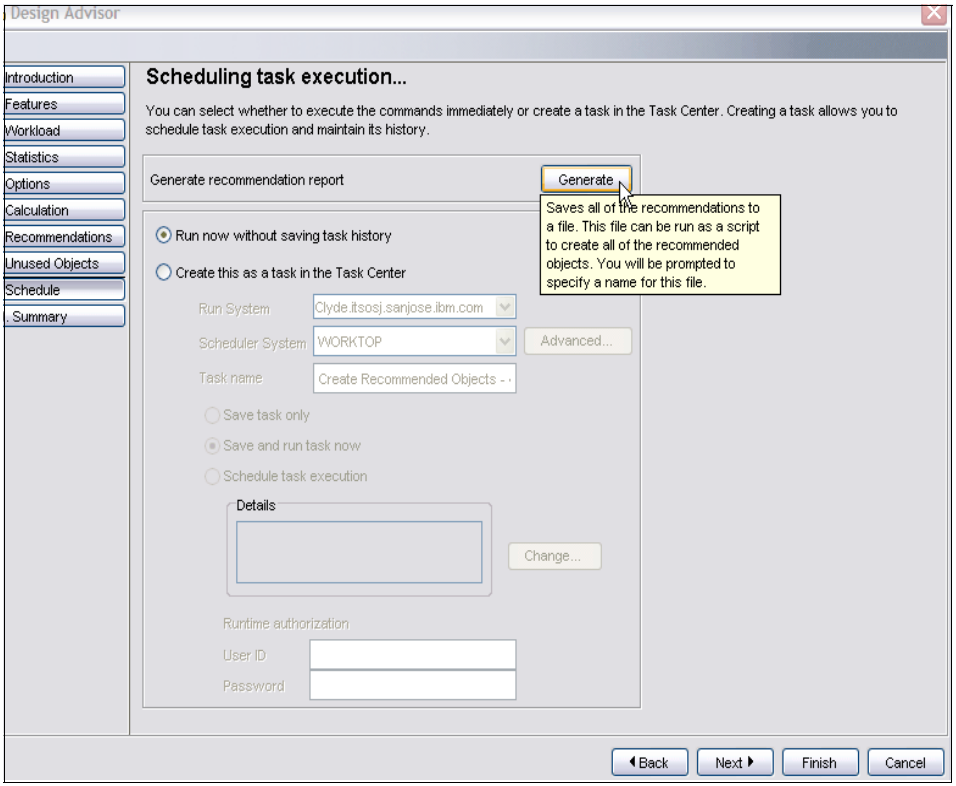


Figure 5-7 Executing the MDC conversion

5.5.4 Using MDC to provide roll-out functionality

A data warehouse contains an achievement table as originally defined in Example 5-22 on page 212. This table in production contains several years of data with approximately 1,500,000,000 rows per year. There is a requirement to roll-out the oldest year when the maximum number of years planned for the table has been reached. In other words, if we are storing three years in the table, we delete the oldest year after adding the fourth year.

As the table was defined, the simple `DELETE FROM AG.CMWT_O_ACHIEVEMENT WHERE ACCT_YR = '2005'` either failed due to “transaction log full” errors or ran for a significant amount of time that was considered excessive for the available update window in the warehouse processing cycle. Various other programmatic solutions were tried, but most had the same problems of taking too much time to perform the total delete or exhausting the available transaction logs.

Example 5-22 DDL for achievement table in warehouse

```
CREATE TABLE ag.cmwt_o_achievement  (
   ee_ie_apsk             DECIMAL (15,0) NOT NULL ,
   meas_detail_apsk      CHAR (8) FOR BIT DATA NOT NULL ,
   meas_detail_ver       SMALLINT NOT NULL ,
   meas_apsk              CHAR (8) FOR BIT DATA NOT NULL ,
   meas_ver              SMALLINT NOT NULL ,
   ach_posting_mth       CHAR (2) NOT NULL ,
   ach_posting_seq_no     SMALLINT NOT NULL ,
   acct_yr               CHAR (4) NOT NULL ,
   ach_eff_mth           CHAR (2) NOT NULL ,
   sys_id                CHAR (4) NOT NULL ,
   orig_achvt_amt        DECIMAL (15,2) NOT NULL ,
   recg_achvt_amt        DECIMAL (15,2) NOT NULL ,
   int_tset_id           INTEGER NOT NULL WITH DEFAULT 0,
   int_tsset_id          INTEGER NOT NULL WITH DEFAULT 0,
   batch_nbr             CHAR (3) NOT NULL ,
   key_set_nbr            SMALLINT NOT NULL ,
   frozen_indc            CHAR (1) NOT NULL WITH DEFAULT 'N',
   PRIMARY KEY  (
      meas_apsk,
      meas_ver,
      ach_posting_mth,
      ach_posting_seq_no,
      acct_yr,
      ee_ie_apsk,
      meas_detail_apsk,
      meas_detail_ver )
   )
   DISTRIBUTE BY HASH(meas_apsk)
   IN tsrp_ag_achieve LONG IN tslp_ag_lobsp
```

The best solution for this table is to use table partitioning to attach and detach a year of data. However, if the warehouse processing requirements did not allow for granting the necessary authorities for that process to the user executing the update stream, MDC provides another alternative.

By using `ACCT_YR` as a dimension column, we can take advantage of the delete processing for MDC tables. The potential wasted space, even if the last block in a cell used only one page, is insignificant compared to the approximately 24,000,000 pages required to store the data for one year. The normal delete processing deletes one row at a time and logs the deleted data. This requires more than 90 gigabytes of log space. The MDC table delete, using the simple delete statement, requires approximately 100 megabytes of log space for the few

bytes on each page that were updated. In addition, because the delete performs at the page level instead of the row level and updates only a few bytes per page, the processing time is significantly reduced.

5.5.5 Using MDC on a fact table in a star schema warehouse

In the classic star-schema design, a central fact table is connected by keys to several dimension tables. The *dimension tables* contain the attributes used in the query's WHERE clause and are joined to the fact table by the key fields to find facts (or rows) with the desired attributes. For example, a fact table of `PRODUCT_SALES` (with foreign key columns, such as `CUSTOMER_KEY`, `PERIOD_KEY`, and `PRODUCT_KEY`, and fact columns, such as `QUANTITY` and `VALUE`) might be associated with dimension tables of `CUSTOMER` (which contains customer attributes, such as `LOCATION`, `BUSINESS_TYPE`, and `SALES_VOLUME`), `TIME_PERIOD` (with columns, such as `DATE`, `QUARTER`, and `MONTH`) and `PRODUCT` (which contains information, such as `COST`, `PRODUCT_TYPE`, and `PACKAGE_SIZE`). Each dimension table has a primary key, and the fact table contains foreign keys for each of the dimension tables. In our brief example, each fact table row contains the key value of the customer to whom the sale was made and the key value of the product sold. A sample query that shows total quantity and dollar value of sales by quarter for `2005` for customers with sales volume between `40000` and `100000` might be similar to this:

```
SELECT quarter, SUM(quantity), SUM(value) FROM product_sales INNER
JOIN customer ON product_sales.customer_key=customer.customer_key
INNER JOIN time_period ON
product_sales.period_key=time_period.period_key WHERE
time_period.year=2005 AND customer.sales_volume BETWEEN 40000 AND
100000 GROUP BY quarter
```

Without multi-dimensional clustering, the usual implementation of the fact table includes row-level indexes for `CUSTOMER_KEY`, `PRODUCT_KEY`, and `PERIOD_KEY`. DB2 typically uses the RID-level indexes and index ANDing to determine which rows of the fact table have to be retrieved, after the list of key values that satisfied the where conditions was determined by scanning the dimension tables.

With MDC, any (or all) of these key fields can be considered for dimensions. For each dimension, the resulting block index is significantly smaller than the corresponding RID-level index, by a factor approximately equal to the number of rows in a block. In addition, the clustering of the data in the fact table means that less I/O is usually required to read all the qualifying rows.

Of course, as with any MDC table, you must analyze the data to determine the number of rows per cell to be sure that wasted space is at a minimum. In our simple example, it is quite a stretch to assume that all three keys qualify as

dimensions, because that implies that for any one customer, period, and product, there is enough data to fill a block. However, by adding a few computed columns to the fact table that are computed by dividing the customer and product keys by a certain value, we might be able to arrive at a dimension structure that does have enough data to fill each block. These computed columns add length to each row of the fact table, but the improved performance of the MDC dimensions and the reduced size of the MDC block indexes compared to the row-level indexes might offset that cost.

Based on this exercise, we strongly recommend that you use the DB2 Design Advisor to help you determine a reasonable solution.

6

Using database partitioning, table partitioning, and MDC together

In this chapter, we discuss how you can use database partitioning (DPF), table partitioning, and multi-dimensional clustering (MDC) together.

Except as noted in each section, there are no special requirements for administration and management of an environment that uses two or three of these partitioning technologies concurrently. You must comply with any restrictions defined in previous chapters, because they apply to the individual technologies.

© Copyright IBM Corp. 2007. All rights reserved.

6.1 Database partitioning and MDC

A table that spans multiple database partitions can also utilize multi-dimensional clustering (MDC). Each database partition contains the dimension block indexes and consolidated block index for the rows in that partition. Example 6-1 shows the DDL to define an MDC table distributed over multiple database partitions. Compare this to Example 5-21 on page 204.

Example 6-1 DDL for LINEITEM table with MDC and partitioning

```
CREATE TABLE  mdc.lineitem
    (l_orderkey INTEGER NOT NULL ,
     l_partkey INTEGER NOT NULL ,
     l_suppkey INTEGER NOT NULL ,
     l_linenumber INTEGER NOT NULL ,
     l_quantity DECIMAL(15, 2) NOT NULL ,
     l_extendedprice DECIMAL (15, 2) NOT NULL ,
     l_discount DECIMAL(15, 2) NOT NULL ,
     l_tax DECIMAL(15, 2) NOT NULL ,
     l_returnflag CHAR(1) NOT NULL ,
     l_linestatus CHAR (1) NOT NULL ,
     l_shipdate DATE NOT NULL ,
     l_commitdate DATE NOT NULL ,
     l_receiptdate DATE NOT NULL ,
     l_shipinstruct CHAR(25) NOT NULL ,
     l_shipmode CHAR(10) NOT NULL ,
     l_comment VARCHAR( 44 ) NOT NULL,
     l_days_in_transit GENERATED AS
        (DAYS(l_receiptdate) - DAYS(l_shipdate)),
     l_ship_yymm GENERATED AS (INTEGER( l_shipdate )/100))
ORGANIZE BY (l_shipmode,l_ship_yymm,l_days_in_transit)
DISTRIBUTE BY hash( l_partkey )
IN tbsp123 ;


CREATE index mdc.ndx_l_partnbr ON mdc.lineitem (l_partkey)
   ALLOW REVERSE SCANS;
```

In general, there is no conflict between DPF and MDC. The distribution key might or might not be used as a dimension key. If it is used as a dimension key, there is no impact on cell size in the partitions. If the distribution key is not a dimension key, you must estimate the average blocks per cell by database partition. Database partitioning might reduce what was a reasonable average cell size to a value less than one block, which means wasted space. *This does not mean that*

you must make the distribution key an MDC dimension. The keys serve different purposes and have different standards for column selection. The columns must be selected based on how well they fill those purposes. Choose the distribution key to provide the maximum collocation with other tables in the database with which the table is often joined. Choose the dimension keys for query performance.

Estimate average blocks per cell by partition

For a new table or an existing table that is not distributed across multiple database partitions, there is no practical way to estimate the distribution of the rows and cells across database partitions. Assume the best case, which is that the cells are equally dense (that is, have the same number of rows) across all database partitions. Compute the average blocks per cell by partition by dividing the average blocks per cell across the whole table by the number of database partitions.

For an existing table that is distributed across multiple database partitions, use SQL similar to Example 6-2 to compute the average rows for cells in each database partition.

Example 6-2 Computing average rows for a cell across database partitions

```
SELECT     partition,
           COUNT( * ) cells,
           AVG(RowsPerCell) AS RowsPerCell,
           MIN(RowsPerCell) AS MinRowsPerCell,
           MAX(RowsPerCell) AS maxRowsPerCell,
           STDDEV(RowsPerCell) AS sdRowsPerCell
FROM       (
           SELECT     dbpartitionnum(col1) partition,
                      col1,
                      col2,
                      ... ,
                      colN,
                      COUNT( * ) RowsPerCell
           FROM       table1
           GROUP BY   dbpartitionnum(col1),
                      col1,
                      col2,
                      ... ,
                      colN
           ) cell_table
GROUP BY   ROLLUP(partition)
ORDER BY   1
```

If the data is not evenly distributed, look for partitions that have an average rows in a cell that is significantly lower than the overall average. This might indicate more wasted space in that partition, which translates into more space required to store the cells in the partition. Because the partitioned database allocates the table space equally across partitions, this means that the amount of additional space is multiplied by the number of partitions. For example, if a table requires five extents in most partitions, but one of the partitions requires seven extents, each partition has seven extents allocated for the table. This is a function of DPF. MDC might simply exacerbate the condition.

Determining page size and extent size

Determining the page size and the extent size for an MDC table in a DPF environment is essentially the same as for a non-partitioned environment. The major difference is that the average rows in a cell must be computed at the partition level. If the distribution key is a dimension key, the averages are the same as for the non-partitioned table calculations (as calculated with the SQL in Example 5-4 on page 182). Otherwise (and this is normally the case), the averages are approximately equal to the non-partitioned average divided by the number of partitions. This usually means that your best page size and best extent size are smaller than the optimum values for the equivalent non-partitioned table.

In our non-partitioned example (5.5.2, “Utilizing both dimension and row-level indexes” on page 201), the `LINEITEM` table had an average rows in a cell of `686` with a standard deviation of `131` and an average row size of `139`. When we executed a query based on the SQL shown in Example 6-2 on page 217, we got the results shown in Example 6-3.

Example 6-3 Average rows in cell for partitioned MDC LINEITEM table

```
PARTITION   CELLS ROWSPERCELL MINROWS MAXROWS SD
--------- ------- ----------- ------- ------- ------------------------
1           17445         228       1     304   +4.46416060975245E+001
2           17440         229       1     311   +4.45760485138464E+001
3           17445         229       1     308   +4.45651991378273E+001
-           52330         229       1     311   +4.45962812765570E+001
```

We use the methodology that is described in “Determine candidate page size and extent size combinations” on page 182 to choose the page size and extent size. If we use 228 and 46 as the average and standard deviation (taking the smallest average and the highest deviation give a worst case estimation of the probability of a small cell), we get Table 6-1 on page 219. In this table, it is obvious that the only good choice for page size and extent size is `4096` bytes for the page and `4` pages in an extent.

Table 6-1 Computing rows in a page and extent for database partition and MDC

Page size (in bytes)	Rows in a page	Extent size (in pages)	Rows in extent	Extents in a cell	Probability of a cell smaller than one extent
4096	27	4	108	2.11	0.45%
		8	216	1.06	39.71%
		12	324	0.70	98.16%
8192	54	4	216	1.06	39.71%
16384	109	4	436	0.52	100.00%
32768	219	4	876	0.26	100.00%

6.2 Database partitioning and table partitioning

In this section, we discuss the combined use of DPF and table partitioning. The benefits provided by table partitioning and DPF are distinct but complementary. You can use both together to get all the benefits:

- Table partitioning provides:
  - Scan parallelism (the same as ordinary tables)
  - Partition elimination
- DB2 Enterprise Server Edition (ESE) with DPF provides:
  - Query parallelism
  - Divide and conquer for REORG and BACKUP

6.2.1 Logical representation

A diagram of the logical layout of a partitioned table in a partitioned database is provided in Figure 6-1 on page 220. This describes a multi-partition database that contains a database partition group that spans all or some of the database partitions. The database partition group contains four table spaces (table spaces 1 through 4). The partitioned table spans those four table spaces.

The dotted line in Figure 6-1 on page 220 represents the logical boundary of the partitioned table. In this case, it spans a minimum of five database partitions and four table spaces. Implementing a data partitioned table with database partitioning means that the table is distributed across the partitions defined in the database partition group by using the hashing algorithm. Your use of the

combination of database partitioning and table partitioning now means that you can distribute data across multiple database partitions, multiple table spaces (due to the table partitioning), and multiple containers. This provides significant scalability for a single table. It can also significantly boost the query performance by taking advantage of the massively parallel processing (MPP) plus symmetric multiprocessing (SMP) with table partition elimination.

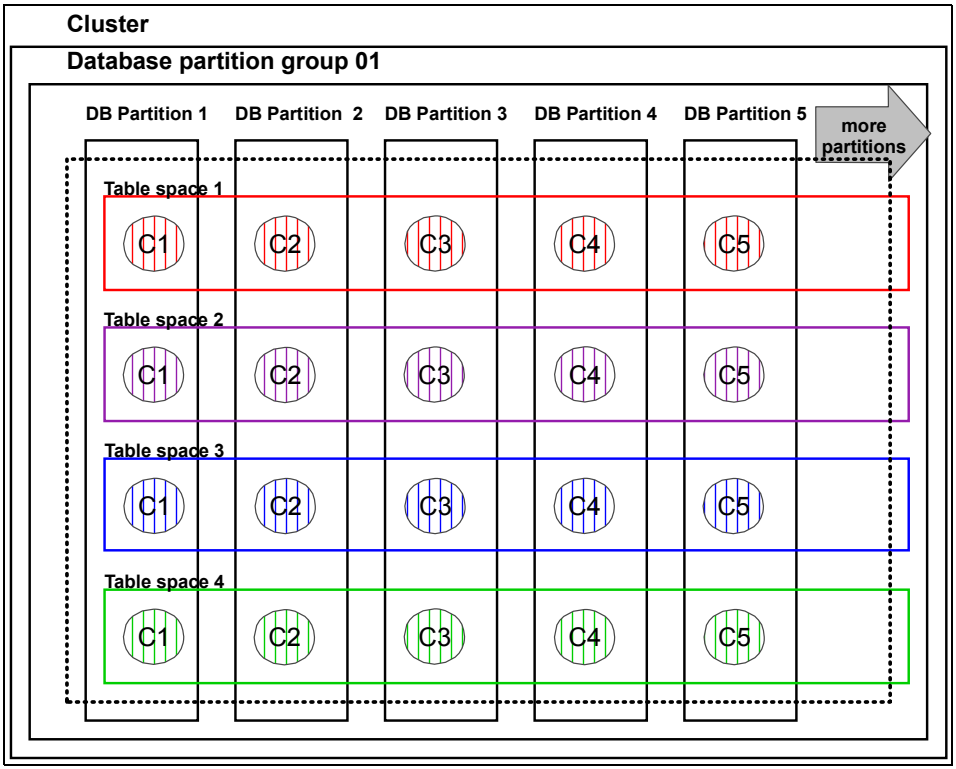


Figure 6-1 Partitioned table in a partitioned database

6.2.2 Implementing a table using table partitioning and database partitioning

The statements that we used to create a partitioned table in a DPF environment are shown in Example 6-4 on page 221. This was based on a previously created DPF environment containing eight partitions.

Example 6-4 Implementation of database partitioning and table partitioning

```
CREATE DATABASE PARTITION GROUP dbpg2007 ON DBPARTITIONNUMS (0 to 7);

CREATE BUFFERPOOL dbpg2007bp1 IMMEDIATE  DATABASE PARTITION GROUP dbpg2007 SIZE
1000 AUTOMATIC
PAGESIZE 4 K ;

CREATE TABLESPACE inv_tbsp2007_00 IN DATABASE PARTITION GROUP dbpg2007 MANAGED
BY DATABASE
USING ( FILE '/db2/db2inst4/db00/inv_tbsp2007_00.dbf' 12800 ) ON
DBPARTITIONNUMS (0)
USING ( FILE '/db2/db2inst4/db01/inv_tbsp2007_00.dbf' 12800 ) ON
DBPARTITIONNUMS (1)
USING ( FILE '/db2/db2inst4/db02/inv_tbsp2007_00.dbf' 12800 ) ON
DBPARTITIONNUMS (2)
USING ( FILE '/db2/db2inst4/db03/inv_tbsp2007_00.dbf' 12800 ) ON
DBPARTITIONNUMS (3)
USING ( FILE '/db2/db2inst4/db04/inv_tbsp2007_00.dbf' 12800 ) ON
DBPARTITIONNUMS (4)
USING ( FILE '/db2/db2inst4/db05/inv_tbsp2007_00.dbf' 12800 ) ON
DBPARTITIONNUMS (5)
USING ( FILE '/db2/db2inst4/db06/inv_tbsp2007_00.dbf' 12800 ) ON
DBPARTITIONNUMS (6)
USING ( FILE '/db2/db2inst4db2inst4/db07/inv_tbsp2007_00.dbf' 12800 ) ON
DBPARTITIONNUMS (7)
AUTORESIZE YES
BUFFERPOOL  dbpg2007bp1 ;

CREATE TABLESPACE inv_tbsp2007_01 IN DATABASE PARTITION GROUP dbpg2007 MANAGED
BY DATABASE
USING ( FILE '/db2/db2inst4/db00/inv_tbsp2007_01.dbf' 12800 ) ON
DBPARTITIONNUMS (0)
USING ( FILE '/db2/db2inst4/db01/inv_tbsp2007_01.dbf' 12800 ) ON
DBPARTITIONNUMS (1)
USING ( FILE '/db2/db2inst4/db02/inv_tbsp2007_01.dbf' 12800 ) ON
DBPARTITIONNUMS (2)
USING ( FILE '/db2/db2inst4/db03/inv_tbsp2007_01.dbf' 12800 ) ON
DBPARTITIONNUMS (3)
USING ( FILE '/db2/db2inst4/db04/inv_tbsp2007_01.dbf' 12800 ) ON
DBPARTITIONNUMS (4)
USING ( FILE '/db2/db2inst4/db05/inv_tbsp2007_01.dbf' 12800 ) ON
DBPARTITIONNUMS (5)
USING ( FILE '/db2/db2inst4/db06/inv_tbsp2007_01.dbf' 12800 ) ON
DBPARTITIONNUMS (6)
USING ( FILE '/db2/db2inst4/db07/inv_tbsp2007_01.dbf' 12800 ) ON
DBPARTITIONNUMS (7)
AUTORESIZE YES
BUFFERPOOL  dbpg2007bp1 ;
```

```
CREATE TABLESPACE inv_tbsp2007_02 IN DATABASE PARTITION GROUP dbpg2007 MANAGED
BY DATABASE
USING ( FILE '/db2/db2inst4/db00/inv_tbsp2007_02.dbf' 12800 ) ON
DBPARTITIONNUMS (0)
USING ( FILE '/db2/db2inst4/db01/inv_tbsp2007_02.dbf' 12800 ) ON
DBPARTITIONNUMS (1)
USING ( FILE '/db2/db2inst4/db02/inv_tbsp2007_02.dbf' 12800 ) ON
DBPARTITIONNUMS (2)
USING ( FILE '/db2/db2inst4/db03/inv_tbsp2007_02.dbf' 12800 ) ON
DBPARTITIONNUMS (3)
USING ( FILE '/db2/db2inst4/db04/inv_tbsp2007_02.dbf' 12800 ) ON
DBPARTITIONNUMS (4)
USING ( FILE '/db2/db2inst4/db05/inv_tbsp2007_02.dbf' 12800 ) ON
DBPARTITIONNUMS (5)
USING ( FILE '/db2/db2inst4/db06/inv_tbsp2007_02.dbf' 12800 ) ON
DBPARTITIONNUMS (6)
USING ( FILE '/db2/db2inst4/db07/inv_tbsp2007_02.dbf' 12800 ) ON
DBPARTITIONNUMS (7)
AUTORESIZE YES
BUFFERPOOL  dbpg2007bp1 ;

CREATE TABLESPACE inv_tbsp2007_03 IN DATABASE PARTITION GROUP dbpg2007 MANAGED
BY DATABASE
USING ( FILE '/db2/db2inst4/db00/inv_tbsp2007_03.dbf' 12800 ) ON
DBPARTITIONNUMS (0)
USING ( FILE '/db2/db2inst4/db01/inv_tbsp2007_03.dbf' 12800 ) ON
DBPARTITIONNUMS (1)
USING ( FILE '/db2/db2inst4/db02/inv_tbsp2007_03.dbf' 12800 ) ON
DBPARTITIONNUMS (2)
USING ( FILE '/db2/db2inst4/db03/inv_tbsp2007_03.dbf' 12800 ) ON
DBPARTITIONNUMS (3)
USING ( FILE '/db2/db2inst4/db04/inv_tbsp2007_03.dbf' 12800 ) ON
DBPARTITIONNUMS (4)
USING ( FILE '/db2/db2inst4/db05/inv_tbsp2007_03.dbf' 12800 ) ON
DBPARTITIONNUMS (5)
USING ( FILE '/db2/db2inst4/db06/inv_tbsp2007_03.dbf' 12800 ) ON
DBPARTITIONNUMS (6)
USING ( FILE '/db2/db2inst4/db07/inv_tbsp2007_03.dbf' 12800 ) ON
DBPARTITIONNUMS (7)
AUTORESIZE YES
BUFFERPOOL  dbpg2007bp1 ;

CREATE TABLESPACE inv_tbsp2007_ix IN DATABASE PARTITION GROUP dbpg2007 MANAGED
BY DATABASE
USING ( FILE '/db2/db2inst4/db00/inv_tbsp2007_ix.dbf' 12800 ) ON
DBPARTITIONNUMS (0)
USING ( FILE '/db2/db2inst4/db01/inv_tbsp2007_ix.dbf' 12800 ) ON
DBPARTITIONNUMS (1)
```

```
USING ( FILE '/db2/db2inst4/db02/inv_tbsp2007_ix.dbf' 12800 ) ON
DBPARTITIONNUMS (2)
USING ( FILE '/db2/db2inst4/db03/inv_tbsp2007_ix.dbf' 12800 ) ON
DBPARTITIONNUMS (3)
USING ( FILE '/db2/db2inst4/db04/inv_tbsp2007_ix.dbf' 12800 ) ON
DBPARTITIONNUMS (4)
USING ( FILE '/db2/db2inst4/db05/inv_tbsp2007_ix.dbf' 12800 ) ON
DBPARTITIONNUMS (5)
USING ( FILE '/db2/db2inst4/db06/inv_tbsp2007_ix.dbf' 12800 ) ON
DBPARTITIONNUMS (6)
USING ( FILE '/db2/db2inst4/db07/inv_tbsp2007_ix.dbf' 12800 ) ON
DBPARTITIONNUMS (7)
AUTORESIZE YES
BUFFERPOOL  dbpg2007bp1 ;

CREATE TABLE invoice_part_2007
(
CUSTNO BIGINT  NOT NULL ,
TRANSACTION_DATE DATE  NOT NULL ,
AMOUNT DECIMAL (15, 2)  NOT NULL ,
CUSTNAME CHARACTER (10)  NOT NULL ,
TIME TIME  NOT NULL,
REGION INT NOT NULL
)
INDEX IN inv_tbsp2007_ix
PARTITION BY RANGE (TRANSACTION_DATE NULLS LAST)
(
PARTITION inv_0 STARTING FROM ('01/01/2007') INCLUSIVE IN inv_tbsp2007_00 ,
PARTITION inv_1 STARTING FROM ('04/01/2007') INCLUSIVE IN inv_tbsp2007_01 ,
PARTITION inv_2 STARTING FROM ('07/01/2007') INCLUSIVE IN inv_tbsp2007_02 ,
PARTITION inv_3 STARTING FROM ('10/01/2007') INCLUSIVE ENDING AT ('12/31/2007')
INCLUSIVE IN inv_tbsp2007_03
);
```

Note the shortened version of the range partitioning syntax where the ENDING AT parameter is implied.

We inserted data into the table using the statement in Example 6-5. As you can see, there was considerable randomization of the data. However, the insert was very quick due to the partitioned database resources and the table partitioning that allow the distribution of the containers over a large number of physical drives.

Example 6-5 Insert command for random data

```
INSERT INTO invoice_part_2007 (custno, transaction_date, amount,
custname,time, region)
WITH custnos(custno) AS
```

```
( VALUES(1) UNION ALL
  SELECT custno+1 FROM custnos WHERE custno < 1000000 )
SELECT custno,
       DATE(732677 + RAND()*364) ,
       decimal(50000 + RAND()*90000),
       TRANSLATE ( CHAR(BIGINT(RAND() * 100000000 )), 'abcdefgHij',
'1234567890'),
       current time -(RAND()*86400) seconds,
       integer(1 + rand()*6)
FROM custnos
;
```

As mentioned in Chapter 4, “Table partitioning” on page 125, you need to keep the number of containers and the size of the containers for a particular table space uniform across all database partitions. But with table partitioning added, you can now have varying numbers of containers between table spaces that are used in the partitioned table. This is shown in Figure 6-2 on page 225.

This allows you to not only distribute the workload for a table across multiple database partitions but across a greater number of disk drives. The result of this is that the combined advantages of database partitioning and table space partitioning have a multiplying effect with regard to I/O performance.

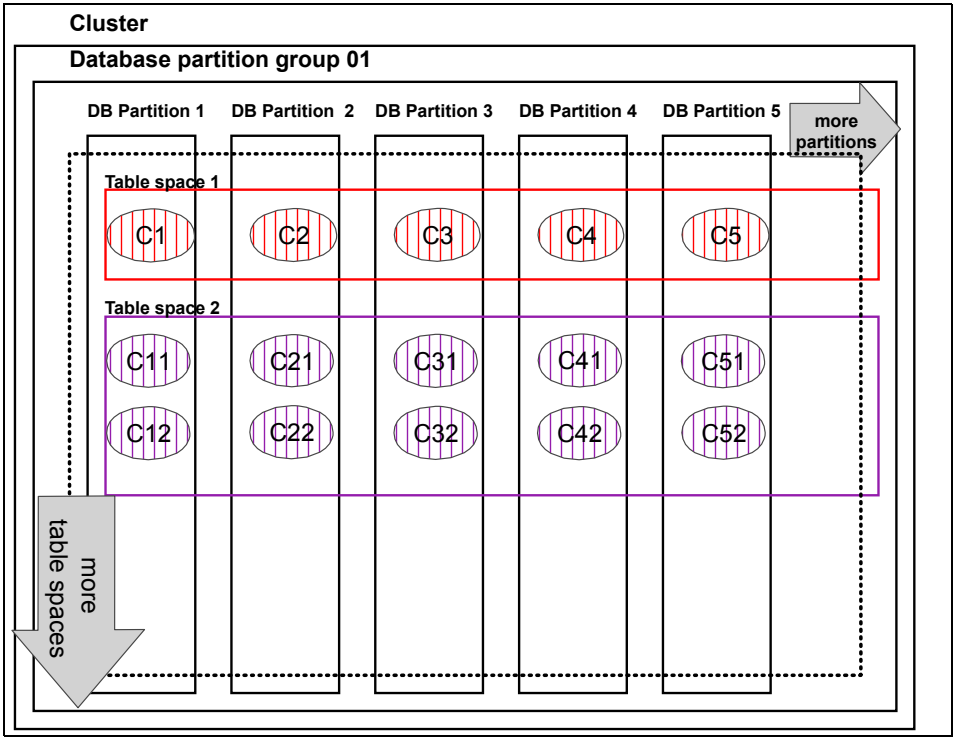


Figure 6-2 Multiple containers per table space in a partitioned table in a partitioned database

An example of the SQL to distribute the data over differing containers is in Example 6-6. The SQL statements show that you have uniformity across database partitions but have the flexibility to vary containers by table space. There are multiple containers for table space `INV_TBSP2007_03` to allow for a large data growth in the fourth quarter of `2007`.

Example 6-6 Multiple table space containers

```
CREATE DATABASE PARTITION GROUP dbpg2007 ON DBPARTITIONNUMS (0 to 7);

CREATE BUFFERPOOL dbpg2007bp1 IMMEDIATE  DATABASE PARTITION GROUP dbpg2007 SIZE
1000 AUTOMATIC
PAGESIZE 4 K ;

CREATE TABLESPACE inv_tbsp2007_00 IN DATABASE PARTITION GROUP dbpg2007 MANAGED
BY DATABASE
USING ( FILE '/db2/mclaua/db00/inv_tbsp2007_00.dbf' 12800 ) ON DBPARTITIONNUMS
(0)
```

```
USING ( FILE '/db2/mclaua/db01/inv_tbsp2007_00.dbf' 12800 ) ON DBPARTITIONNUMS
(1)
USING ( FILE '/db2/mclaua/db02/inv_tbsp2007_00.dbf' 12800 ) ON DBPARTITIONNUMS
(2)
USING ( FILE '/db2/mclaua/db03/inv_tbsp2007_00.dbf' 12800 ) ON DBPARTITIONNUMS
(3)
USING ( FILE '/db2/mclaua/db04/inv_tbsp2007_00.dbf' 12800 ) ON DBPARTITIONNUMS
(4)
USING ( FILE '/db2/mclaua/db05/inv_tbsp2007_00.dbf' 12800 ) ON DBPARTITIONNUMS
(5)
USING ( FILE '/db2/mclaua/db06/inv_tbsp2007_00.dbf' 12800 ) ON DBPARTITIONNUMS
(6)
USING ( FILE '/db2/mclaua/db07/inv_tbsp2007_00.dbf' 12800 ) ON DBPARTITIONNUMS
(7)
AUTORESIZE YES
BUFFERPOOL  dbpg2007bp1 ;

CREATE TABLESPACE inv_tbsp2007_01 IN DATABASE PARTITION GROUP DBPG2007 MANAGED
BY DATABASE
USING ( FILE '/db2/mclaua/db00/inv_tbsp2007_01.dbf' 12800 ) ON DBPARTITIONNUMS
(0)
USING ( FILE '/db2/mclaua/db01/inv_tbsp2007_01.dbf' 12800 ) ON DBPARTITIONNUMS
(1)
USING ( FILE '/db2/mclaua/db02/inv_tbsp2007_01.dbf' 12800 ) ON DBPARTITIONNUMS
(2)
USING ( FILE '/db2/mclaua/db03/inv_tbsp2007_01.dbf' 12800 ) ON DBPARTITIONNUMS
(3)
USING ( FILE '/db2/mclaua/db04/inv_tbsp2007_01.dbf' 12800 ) ON DBPARTITIONNUMS
(4)
USING ( FILE '/db2/mclaua/db05/inv_tbsp2007_01.dbf' 12800 ) ON DBPARTITIONNUMS
(5)
USING ( FILE '/db2/mclaua/db06/inv_tbsp2007_01.dbf' 12800 ) ON DBPARTITIONNUMS
(6)
USING ( FILE '/db2/mclaua/db07/inv_tbsp2007_01.dbf' 12800 ) ON DBPARTITIONNUMS
(7)
AUTORESIZE YES
BUFFERPOOL  DBPG2007bp1 ;

CREATE TABLESPACE inv_tbsp2007_02 IN DATABASE PARTITION GROUP dbpg2007 MANAGED
BY DATABASE
USING ( FILE '/db2/mclaua/db00/inv_tbsp2007_02.dbf' 12800 ) ON DBPARTITIONNUMS
(0)
USING ( FILE '/db2/mclaua/db01/inv_tbsp2007_02.dbf' 12800 ) ON DBPARTITIONNUMS
(1)
USING ( FILE '/db2/mclaua/db02/inv_tbsp2007_02.dbf' 12800 ) ON DBPARTITIONNUMS
(2)
USING ( FILE '/db2/mclaua/db03/inv_tbsp2007_02.dbf' 12800 ) ON DBPARTITIONNUMS
(3)
```

```
USING ( FILE '/db2/mclaua/db04/inv_tbsp2007_02.dbf' 12800 ) ON DBPARTITIONNUMS
(4)
USING ( FILE '/db2/mclaua/db05/inv_tbsp2007_02.dbf' 12800 ) ON DBPARTITIONNUMS
(5)
USING ( FILE '/db2/mclaua/db06/inv_tbsp2007_02.dbf' 12800 ) ON DBPARTITIONNUMS
(6)
USING ( FILE '/db2/mclaua/db07/inv_tbsp2007_02.dbf' 12800 ) ON DBPARTITIONNUMS
(7)
AUTORESIZE YES
BUFFERPOOL  dbpg2007bp1 ;

CREATE TABLESPACE inv_tbsp2007_03 IN DATABASE PARTITION GROUP dbpg2007 MANAGED
BY DATABASE
USING ( FILE '/db2/mclaua/db00_1/inv_tbsp2007_03.dbf' 12800 ,
        FILE '/db2/mclaua/db00_2/inv_tbsp2007_03.dbf' 12800 ) ON
DBPARTITIONNUMS (0)
USING ( FILE '/db2/mclaua/db01_1/inv_tbsp2007_03.dbf' 12800 ,
        FILE '/db2/mclaua/db01_2/inv_tbsp2007_03.dbf' 12800 ) ON
DBPARTITIONNUMS (1)
USING ( FILE '/db2/mclaua/db02_1/inv_tbsp2007_03.dbf' 12800 ,
        FILE '/db2/mclaua/db02_2/inv_tbsp2007_03.dbf' 12800 ) ON
DBPARTITIONNUMS (2)
USING ( FILE '/db2/mclaua/db03_1/inv_tbsp2007_03.dbf' 12800 ,
        FILE '/db2/mclaua/db03_2/inv_tbsp2007_03.dbf' 12800 ) ON
DBPARTITIONNUMS (3)
USING ( FILE '/db2/mclaua/db04_1/inv_tbsp2007_03.dbf' 12800 ,
        FILE '/db2/mclaua/db04_2/inv_tbsp2007_03.dbf' 12800 ) ON
DBPARTITIONNUMS (4)
USING ( FILE '/db2/mclaua/db05_1/inv_tbsp2007_03.dbf' 12800 ,
        FILE '/db2/mclaua/db05_2/inv_tbsp2007_03.dbf' 12800 ) ON
DBPARTITIONNUMS (5)
USING ( FILE '/db2/mclaua/db06_1/inv_tbsp2007_03.dbf' 12800 ,
        FILE '/db2/mclaua/db06_2/inv_tbsp2007_03.dbf' 12800 ) ON
DBPARTITIONNUMS (6)
USING ( FILE '/db2/mclaua/db07_1/inv_tbsp2007_03.dbf' 12800 ,
        FILE '/db2/mclaua/db07_2/inv_tbsp2007_03.dbf' 12800 ) ON
DBPARTITIONNUMS (7)
AUTORESIZE YES
BUFFERPOOL  dbpg2007bp1 ;

CREATE TABLESPACE inv_tbsp2007_ix IN DATABASE PARTITION GROUP dbpg2007 MANAGED
BY DATABASE
USING ( FILE '/db2/mclaua/db00/inv_tbsp2007_ix.dbf' 12800 ) ON DBPARTITIONNUMS
(0)
USING ( FILE '/db2/mclaua/db01/inv_tbsp2007_ix.dbf' 12800 ) ON DBPARTITIONNUMS
(1)
USING ( FILE '/db2/mclaua/db02/inv_tbsp2007_ix.dbf' 12800 ) ON DBPARTITIONNUMS
(2)
```

```
USING ( FILE '/db2/mclaua/db03/inv_tbsp2007_ix.dbf' 12800 ) ON DBPARTITIONNUMS
(3)
USING ( FILE '/db2/mclaua/db04/inv_tbsp2007_ix.dbf' 12800 ) ON DBPARTITIONNUMS
(4)
USING ( FILE '/db2/mclaua/db05/inv_tbsp2007_ix.dbf' 12800 ) ON DBPARTITIONNUMS
(5)
USING ( FILE '/db2/mclaua/db06/inv_tbsp2007_ix.dbf' 12800 ) ON DBPARTITIONNUMS
(6)
USING ( FILE '/db2/mclaua/db07/inv_tbsp2007_ix.dbf' 12800 ) ON DBPARTITIONNUMS
(7)
AUTORESIZE YES
BUFFERPOOL  dbpg2007bp1 ;

CREATE TABLE invoice_part_2007
(
custno BIGINT  NOT NULL ,
transaction_date DATE  NOT NULL ,
amount DECIMAL (15, 2)  NOT NULL ,
custname CHARACTER (10)  NOT NULL ,
time TIME  NOT NULL,
region INT NOT NULL
)
INDEX IN inv_tbsp2007_ix
PARTITION BY RANGE (TRANSACTION_DATE NULLS LAST)
(
PARTITION inv_0 STARTING FROM ('01/01/2007') INCLUSIVE IN inv_tbsp2007_00 ,
PARTITION inv_1 STARTING FROM ('04/01/2007') INCLUSIVE IN inv_tbsp2007_01 ,
PARTITION inv_2 STARTING FROM ('07/01/2007') INCLUSIVE IN inv_tbsp2007_02 ,
PARTITION inv_3 STARTING FROM ('10/01/2007') INCLUSIVE ENDING AT ('12/31/2007')
INCLUSIVE IN inv_tbsp2007_03
);
```

6.3 Table partitioning and MDC

MDC tables can take advantage of table partitioning to provide scalability and roll-in/roll-out functionality. For the query performance, MDC is best for querying multiple dimensions, and table partitioning helps through partition elimination. Use table partitioning and MDC together for maximum benefit. Use the Design Advisor in designing a table partitioned table with MDC. You first define table partitioning based on roll-out strategy. With ranges defined, Design Advisor recommends appropriate MDC dimensions.

Because table partitioning performs a similar function to clustering, there is generally no benefit in partitioning a table on a column and then using that same column as a dimension. However, if the partitioning is at a coarser granularity

than individual column values, using the column as a dimension might provide a query performance benefit.

For example, consider a table that contains a column of the form `YYYYMM`, which contains a financial reporting period, where the table is to contain several years of data with the oldest year rolled out annually. Partitioning the table on this column so that each partition contains one year of information is reasonable for roll-out purposes, while using the column as a dimension (which has monthly granularity) can improve queries that reported by month or quarter.

When determining the page size and extent size for the table, table partitioning affects the average rows in a cell the same way that database partitioning does. Instead of working with the overall average rows in a cell, compute the average rows in a cell for each range. If working with a new table, the simplest estimation is to divide the estimated overall rows in a cell by the number of ranges. For an existing table, use an SQL query similar to Example 6-7 to compute the average rows in a cell and standard deviation for each range. Analyze the results as shown in 5.1.4, “Estimate space requirements” on page 180.

Example 6-7 Computing average rows per cell across table ranges

```
-- rangeN_low_value is the lower limit for range interval N
-- rangeN_high_value is the upper limit
-- table1.rangekey is the TABLE partitioning key

WITH rangelist (range, range_low, range_high) AS (
VALUES     (1, 'range1_low_value', 'range1_high_value'),
           (2, 'range2_low_value', 'range2_high_value' ),
            ...
           (N, ' rangeN_low_value', 'rangeN_high_value' ) )
SELECT     range, COUNT( * ) cells, AVG(RowsPerCell) AS RowsPerCell,
           MIN(RowsPerCell) AS MinRowsPerCell,
           MAX(RowsPerCell) AS maxRowsPerCell,
           STDDEV(RowsPerCell) AS sdRowsPerCell
FROM       (
           SELECT     range, col1, col2,
                      ... ,
                      colN, COUNT( * ) RowsPerCell
           FROM       table1 INNER JOIN rangelist
           ON         table1.rangekey BETWEEN range_low
                      AND range_high
           GROUP BY   range, col1, col2, ... , colN ) cell_table
GROUP BY   rollup(range)
ORDER BY   1
```

When you detach a partition from a data partitioned MDC table, the new table is an MDC table with the same dimensions as the original table. The new table cannot be referenced until you commit the detach. The block indexes for the new table are built when you first access the new table. Consequently, the first access experiences reduced performance. Similarly, when you want to attach a table to the partitioned MDC table, it must be dimensioned identically to the partitioned table.

Example 6-8 shows a simple table definition that includes both table partitioning and MDC.

Example 6-8 ORDERS table with table partitioning and MDC

```
CREATE TABLE tpmdc.orders( o_orderkey       INTEGER NOT NULL,
                           o_custkey        INTEGER NOT NULL,
                           o_orderstatus    CHAR(1) NOT NULL,
                           o_totalprice     DECIMAL(15,2) NOT NULL,
                           o_orderdate      DATE NOT NULL,
                           o_orderpriority  CHAR(15) NOT NULL,
                           o_clerk          CHAR(15) NOT NULL,
                           o_shippriority   INTEGER NOT NULL,
                           o_comment        VARCHAR(79) NOT NULL,
                           PRIMARY KEY(o_orderkey))IN tbsp1
ORGANIZE BY (o_orderpriority, o_shippriority)
PARTITION BY RANGE (o_orderdate NULLS LAST)
(PARTITION order_92 STARTING FROM '01/01/1992' INCLUSIVE,
 PARTITION order_93 STARTING FROM '01/01/1993' INCLUSIVE,
 PARTITION order_94 STARTING FROM '01/01/1994' INCLUSIVE,
 PARTITION order_95 STARTING FROM '01/01/1995' INCLUSIVE,
 PARTITION order_96 STARTING FROM '01/01/1996' INCLUSIVE,
 PARTITION order_97 STARTING FROM '01/01/1997' INCLUSIVE,
 PARTITION order_98 STARTING FROM '01/01/1998' INCLUSIVE
                    ENDING at '12/31/1998' INCLUSIVE)
```

In this example, the `ORDERS` table has dimension keys of `O_ORDERPRIORITY` and `O_SHIPPRIORITY` for query purposes. The table is also range-partitioned by `O_ORDERDATE` to allow orders to be rolled-out of the table by year as they age. When we detach the `ORDER_92` partition into a table, we get a new MDC table. Before we can access the data in that table, we first have to execute a SET INTEGRITY statement on the table. Our first access then causes the block indexes to be built for the table.

Example 6-9 on page 231 shows the SQL statement that can be used to analyze the existing `ORDERS` table to determine the average rows in a cell by range. Note

the casting of the values in the RANGELIST table to match the data type of the column used for range partitioning.

Example 6-9 SQL to compute average rows in a cell by range for the ORDERS table

```
WITH rangelist (range , range_low , range_high ) AS
(VALUES  (1,DATE('1992-01-01'),DATE('1992-12-31')),
        (2,DATE('1993-01-01'),DATE('1993-12-31')),
        (3,DATE('1994-01-01'),DATE('1994-12-31')),
        (4,DATE('1995-01-01'),DATE('1995-12-31')),
        (5,DATE('1996-01-01'),DATE('1996-12-31')),
        (6,DATE('1997-01-01'),DATE('1997-12-31')),
        (7,DATE('1998-01-01'),DATE('1998-12-31'))
)
SELECT     range, COUNT( * ) cells, AVG(RowsPerCell) AS RowsPerCell,
           MIN(RowsPerCell) AS MinRowsPerCell,
           MAX(RowsPerCell) AS maxRowsPerCell,
           STDDEV(RowsPerCell) AS sdRo wsPerCell
FROM       (
           SELECT     range, o_shippriority, o_orderpriority,
                      COUNT( * ), RowsPerCell
           FROM       tpcd.orders INNER JOIN rangelist
           ON         o_orderdate
                      BETWEEN range_low AND range_high
           GROUP BY   range, o_shippriority, o_orderpriority
           ) cell_table
GROUP BY   ROLLUP(range)
ORDER BY   1
```

6.4 Database partitioning, table partitioning, and MDC

If conditions are appropriate, it is possible to utilize all three partitioning technologies on a single table. The primary conditions that you have to meet include:

- A large volume of data
- Periodic removal of data based on a certain range of values
- Frequently queried columns with relatively low cardinalities or foreign keys in a fact table

All the considerations about table spaces, partition keys, and dimensions that were mentioned in conjunction with attaching and detaching tables apply.

A table such as that defined in 5.5.4, “Using MDC to provide roll-out functionality” on page 211 might meet all these requirements:

- With 1,500,000,000 rows per year, the table contains a large volume of data.

  Use database partitioning to distribute this data across several database partitions.

- Only the most recent three or four years of data is kept.

  Use the `ACCT_YR` column to partition the table in each database partition by year. Use the DETACH command to remove years of data as they are no longer needed.

- The columns ending in “APSK” are foreign keys to dimension tables, so they are good candidates for dimensions.

  Create dimension indexes on the foreign keys. You might have to create computed columns from the APSK columns to get the average rows per block high enough to be useful.

Computation of page sizes and extent sizes follows the patterns already established. For a new table, divide the estimated rows in a cell by the number of ranges and then divide that result by the number of database partitions to get an average number of rows in a cell within a range within a database partition. For an existing non-distributed table, use the SQL query in Example 6-7 on page 229 to compute the average rows in a cell by range and divide that result by the number of database partitions. Also, divide the standard deviation by the number of database partitions. For an existing distributed table, use the SQL in Example 6-10 to compute the average rows in a cell by range and database partition and the standard deviation.

Example 6-10 Computing average rows in cells across table ranges

```
WITH rangelist (range, range_low, range_high) AS (
VALUES     (1, 'range1_low_value', 'range1_high_value'),
           (2, 'range2_low_value', 'range2_high_value'),
           ...
           (N, 'rangeN_low_value', 'rangeN_high_value') )
SELECT     partition, range, COUNT( * ) cells, AVG(RowsPerCell)
           AS RowsPerCell,
           MIN(RowsPerCell) AS MinRowsPerCell,
           MAX(RowsPerCell) AS maxRowsPerCell,
           STDDEV(RowsPerCell) AS sdRowsPerCell
FROM       (
           SELECT     DBPARTITIONNUM(col1) partition, range, col1,
                      col2, ... ,
                      colN, COUNT( * ) RowsPerCell
           FROM       table1 inner join rangelist
```

```
          ON         table1.rangekey
                     BETWEEN range_low AND range_high
          GROUP BY   DBPARTITIONNUM(col1), range, col1, col2,
                     ... ,
                     colN) cell_table
GROUP BY  ROLLUP(partition, range)
ORDER BY  1
```

A

Configuring DB2 for SSH in a partitioned environment

When you set up your partitioned environment on UNIX or Linux, the inter-partition communication normally uses the remote shell (rsh). While rsh provides an easy to set up and maintain solution, it lacks the security that your organization might require. The secured shell (ssh) provides the needed security, and this appendix shows you how to set up DB2 to use ssh.

You can choose to use either the Rivest-Shamir-Adleman algorithm (RSA) or the digital signature algorithm (DSA) encryption when setting up ssh. You need to base this decision on your own security requirements. RSA and DSA are two encryption methods that you can used with ssh, because both methods provide a secure environment.

© Copyright IBM Corp. 2007. All rights reserved.

A.1 Setting up public key authentication

Public key-based authentication allows a single user ID (in our case, the DB2 instance owning ID in our Data Partitioning Feature (DPF) environment) to log in as the same user ID on each partition server without being prompted for a password. This can be advantageous if you have a large number of partitions in your DPF environment. Public key-based authentication is considered to be less of a security exposure, because only one particular user is ever configured to ssh or log in to other hosts without being prompted for a password.

We carry out the setup for public key authentication from the instance owning partition server. In our example, `serverA` is the instance owning partition server, and we log on with the instance owning ID. The setup procedures are:

1. Generate an RSA or DSA key pair.

   The first step in setting up public key authentication is to generate an RSA or DSA key pair. When you are logged in as the instance owner, navigate to the .ssh directory in the instance home directory. If the .ssh directory does not exist, you need to create one. Ensure that the .ssh directory does not allow group or other write access.

   Example A-1 shows generating a public key/private key pair for RSA.

   Example: A-1 Generating an RSA-encrypted key pair

   ```
   $ ssh-keygen -t rsa
   ```

   Or you can generate a DSA key pair, see Example A-2.

   Example: A-2 Generating a DSA-encrypted key pair

   ```
   $ ssh-keygen -t dsa
   ```

   You are prompted for input but just accept the default. You then are prompted to enter a passphrase. In our environment, we do not want a passphrase so we press Enter twice. If you enter a passphrase, ssh challenges every authentication attempt. DB2 does not allow rsh to prompt for additional verification.

   Two new files are generated in the ~/.ssh directory, id_rsa (the private key) and id_rsa.pub (the public key), for RSA encryption. In a similar manner, name files are generated for DSA encryption.

2. Enable the key pair.

   Example A-3 on page 237 shows the commands to enable the RSA key pair.

Example: A-3 Enabling the RSA key pair

```
$ cd ~/.ssh
$ mv id_rsa identity
$ chmod 600 identity
$ cat id_rsa.pub >> authorized_keys
$ chmod 644 authorized_keys
$ rm id_rsa.pub
```

See Example A-4 to enable a DSA key pair.

Example: A-4 Enabling the DSA key pair

```
$ cd ~/.ssh
$ mv id_dsa identity
$ chmod 600 identity
$ cat id_dsa.pub >> authorized_keys
$ chmod 644 authorized_keys
$ rm id_dsa.pub
```

3. Set up the host trust relationships.

   Because we have not set up any host trust relationships, the first time that ssh is issued to a new host, a message appears stating that the authenticity of the target host cannot be established. To overcome this situation, we can form a trust relationship with the other hosts in our DPF environment. To achieve this trust relationship, use the ssh-keyscan utility to gather the public host key for each host in the DPF environment and save the keys in the known_hosts file. In our example, we only have two partition servers, `serverA` and `serverB`.

   Example A-5 shows how to use the ssh-keyscan utility to gather RSA keys to set up the host trust relationships.

   Example: A-5 Gathering the RSA public keys

```
$ ssh-keyscan -t rsa serverA,serverA.my.domain.com,192.168.0.10
>>~/.ssh/known_hosts
$ ssh-keyscan -t rsa serverB,serverB.my.domain.com,192.168.0.11
>>~/.ssh/known_hosts
```

   Example A-6 on page 238 shows how to use the ssh-keyscan to gather DSA keys to set up the host trust relationships.

Example: A-6 Gathering the DSA public keys

```
$ ssh-keyscan -t dsa serverA,serverA.my.domain.com,192.168.0.10
>>~/.ssh/known_hosts
$ ssh-keyscan -t dsa serverB,serverB.my.domain.com,192.168.0.11
>>~/.ssh/known_hosts
```

A.2 Setting up host-based authentication

Host-based ssh authentication differs in that it allows any user ID from `serverA` to log in as the same user ID on `serverB` using ssh. The ssh client on `serverA` must be configured to use host-based authentication and the ssh server on `serverB` needs to allow host-based authentication. In your DPF environment, each partition server needs to be configured to use host-based authentication and each partition server must have the ssh client and ssh server configured correctly.

A.2.1 SSH server configuration

To configure the ssh server, use the following steps:

1. Edit the ssh server configuration file.

   The ssh server configuration file, sshd_config, is found in /etc/ssh on AIX, Linux, and Solaris and in /opt/ssh/etc on Hewlett-Packard Unix (HP-UX). You have to log in as the root user to edit this file. Change the HostbasedAuthentication parameter to `Yes`.

   In Example A-7, we have added an additional line to the sshd_config file and left the original line commented out. This allows the ssh server to accept host-based authentication requests from the ssh clients.

   Example: A-7 Edit the sshd_config file

   ```
   # vi sshd_config

   Then edit or change the line;
   #   HostbasedAuthentication no
   HostbasedAuthentication yes
   ```

2. Edit the shosts.equiv file.

   The shosts.equiv file can be found in /etc/ssh on AIX and Linux, in /etc on Solaris, and in /opt/ssh/etc on HP-UX. This file might not exist; therefore, you must create one and ensure that it is owned by the root user and only allows

user read and write access and group/other read access. Each host must be able to communicate with every other host in the DPF environment, so you must set up the shosts.equiv file in such a way that it can be reused on all hosts or partition servers. Edit the file as shown in Example A-8.

Example: A-8 Edit the shosts.equiv file

```
serverA
serverA.domain.com
serverB
serverB.domain.com
```

3. Edit the ssh_known_hosts file.

   The ssh server host system needs to have access to the ssh client host's public key and, for host-based authentication, the trust mechanism looks for public keys in the ssh_known_hosts file. The ssh_known_hosts file is found in the /etc/ssh directory on AIX, Linux, and Solaris and in /opt/ssh/etc on HP-UX. As with the shosts.equiv file, you might need to create the ssh_known_hosts file if it does not exist. Ensure that it is owned by the root user and only allows user read and write access and group/other read access.

   Add the client machine's unqualified host name, fully qualified host name, and IP address to the ssh_known_hosts file. You can use the ssh-keyscan utility to populate this file. The command for this is shown in Example A-9 for RSA and Example A-10 for DSA. You need to change the directory (cd) to /etc/ssh on AIX Linux and Solaris and to /opt/ssh/etc on HP-UX before running this command.

Example: A-9 Updating the ssh_known_hosts file for RSA encryption

```
$ ssh-keyscan -t rsa serverA,serverA.domain.com,192.168.0.10
>>ssh_known_hosts
$ ssh-keyscan -t rsa serverB,serverB.domain.com,192.168.0.10
>>ssh_known_hosts
```

Example: A-10 Updating the ssh_known_hosts file for DSA encryption

```
$ ssh-keyscan -t dsa serverA,serverA.domain.com,192.168.0.10
>>ssh_known_hosts
$ ssh-keyscan -t dsa serverB,serverB.domain.com,192.168.0.10
>>ssh_known_hosts
```

4. Restart the ssh daemon.

   After you update the ssh_known_hosts file, you need to restart the ssh daemon for the changes to take effect. Sending a SIGHUP command to the ssh daemon so that it restarts and reloads its configuration file is one way to

do this. This only affect future ssh sessions, not the sessions currently running.

First, we find the pid for ssh daemon (**sshd**) as shown in Example A-11.

Example: A-11 Finding the sshd pid

```
# ps -ef | grep sshd
   root 1470502   57456   0 15:18:48      -  0:00 /usr/sbin/sshd
   root 1544638 1999180   0 16:08:55  pts/3  0:00 grep sshd
```

This tells us the pid is `1470502`. We can now tell the sshd process to reread the sshd_config file with the command in Example A-12.

Example: A-12 Restart the ssh daemon

```
kill -HUP 1470502
```

Alternatively, you can restart the sshd process using the following commands, which are platform-specific:

– Red Hat linux
  service sshd restart

– Suse Linux

  /etc/rc.d/sshd restart

– Solaris 9 and below

  /etc/init.d/sshd stop
  /etc/init.d/sshd start

– Solaris 10

  svcadm disable ssh
  svcadm enable ssh

– AIX

  stopsrc -s sshd
  startsrc -s sshd

– HP-UX

  /sbin/init.d/sshd2 atop
  /sbin/init.d/sshd2 start

You must execute the steps that we performed to enable the ssh server on every partition server in your DPF environment.

A.2.2 SSH client configuration

The first step in setting up the ssh client is to edit the ssh_config file. You can locate this file in /etc/ssh on AIX, Linux, and Solaris and in /opt/ssh/etc on HP-UX. Edit the file and change the line `HostbasedAuthentication` from `no` to `yes` as shown in Example A-13.

Example: A-13 Editing the ssh_config file

```
#   HostbasedAuthentication no
HostbasedAuthentication yes
```

You then need to add a line to the ssh_config file to tell the ssh client to use the ssh-keysign utility to read the host's private key as shown in Example A-14.

Example: A-14 Enable the EnableSSHKeysign parameter

```
EnableSSHKeysign yes
```

A.3 Configuring DB2 to use ssh

To set up DB2 to start with ssh support, you must enable the DB2 registry variable *DB2RSHCMD* and point it to the path of the ssh command shell. Example A-15 shows the **db2set** command. The default is for DB2 to use rsh; therefore, you only need to set this variable if using ssh.

Example: A-15 Enable DB2 to use ssh

```
db2set DB2RSHCMD=/usr/bin/ssh
```

B

Additional material

This book refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this book is available in softcopy on the Internet from the IBM Redbooks publications Web server. Point your Web browser at:

ftp://www.redbooks.ibm.com/redbooks/SG247467/

Alternatively, you can go to the IBM Redbooks publications Web site at:

http://www.redbooks.ibm.com/

Select **Additional materials** and open the directory that corresponds with the IBM Redbooks publication form number, SG247467.

© Copyright IBM Corp. 2007. All rights reserved.

Using the Web material

The additional Web material that accompanies this book includes the following files:

File name	*Description*
MDC Page and Block Size Estimator.zip	Lotus 123 and Excel spread sheets for estimating MDC page and block size.

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space: 30 KB minimum
Operating System: Windows or Linux
Processor: 486 or higher
Memory: 256 MB

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks publications

For information about ordering these publications, see “How to get Redbooks publications” on page 248. Note that some of the documents referenced here might be available in softcopy only.

- *Up and Running with DB2 UDB ESE: Partitioning for Performance in an e-Business Intelligence World*, SG24-6917

Other publications

These publications are also relevant as further information sources:

IBM - DB2 9

- *What's New*, SC10-4253
- *Administration Guide: Implementation*, SC10-4221
- *Administration Guide: Planning*, SC10-4223
- *Administrative API Reference*, SC10-4231
- *Administrative SQL Routines and Views*, SC10-4293
- *Administration Guide for Federated Systems,* SC19-1020
- *Call Level Interface Guide and Reference, Volume 1*, SC10-4224
- *Call Level Interface Guide and Reference, Volume 2*, SC10-4225
- *Command Reference,* SC10-4226
- *Data Movement Utilities Guide and Reference*, SC10-4227
- *Data Recovery and High Availability Guide and Reference*, SC10-4228
- *Developing ADO.NET and OLE DB Applications*, SC10-4230
- *Developing Embedded SQL Applications*, SC10-4232
- *Developing Java Applications*, SC10-4233

© Copyright IBM Corp. 2007. All rights reserved.

- *Developing Perl and PHP Applications*, SC10-4234
- *Developing SQL and External Routines*, SC10-4373
- *Getting Started with Database Application Development*, SC10-4252
- *Getting started with DB2 installation and administration on Linux and Windows*, GC10-4247
- *Message Reference Volume 1*, SC10-4238
- *Message Reference Volume 2*, SC10-4239
- *Migration Guide*, GC10-4237
- *Performance Guide*, SC10-4222
- *Query Patroller Administration and User's Guide*, GC10-4241
- *Quick Beginnings for DB2 Clients*, GC10-4242
- *Quick Beginnings for DB2 Servers*, GC10-4246
- *Spatial Extender and Geodetic Data Management Feature User's Guide and Reference*, SC18-9749
- *SQL Guide*, SC10-4248
- *SQL Reference, Volume 1,* SC10-4249
- *SQL Reference, Volume 2,* SC10-4250
- *System Monitor Guide and Reference,* SC10-4251
- *Troubleshooting Guide*, GC10-4240
- *Visual Explain Tutorial*, SC10-4319
- *XML Extender Administration and Programming*, SC18-9750
- *XML Guide*, SC10-4254
- *XQuery Reference*, SC18-9796
- *DB2 Connect User's Guide*, SC10-4229
- *DB2 9 PureXML Guide*, SG24-7315
- *Quick Beginnings for DB2 Connect Personal Edition*, GC10-4244
- *Quick Beginnings for DB2 Connect Servers*, GC10-4243

IBM - DB2 8.2

- *What’s New V8,* SC09-4848-01
- *Administration Guide: Implementation V8*, SC09-4820-01
- *Administration Guide: Performance V8*, SC09-4821-01
- *Administration Guide: Planning V8*, SC09-4822-01

- *Application Development Guide: Building and Running Applications V8*, SC09-4825-01
- *Application Development Guide: Programming Client Applications V8*, SC09-4826-01
- *Application Development Guide: Programming Server Applications V8*, SC09-4827-01
- *Call Level Interface Guide and Reference, Volume 1*, *V8*, SC09-4849-01
- *Call Level Interface Guide and Reference, Volume 2, V8*, SC09-4850-01
- *Command Reference V8*, SC09-4828-01
- *Data Movement Utilities Guide and Reference V8*, SC09-4830-01
- *Data Recovery and High Availability Guide and Reference V8*, SC09-4831-01
- *Guide to GUI Tools for Administration and Development*, SC09-4851-01
- *Installation and Configuration Supplement V8*, GC09-4837-01
- *Quick Beginnings for DB2 Clients V8*, GC09-4832-01
- *Quick Beginnings for DB2 Servers V8*, GC09-4836-01
- *Replication and Event Publishing Guide and Reference*, SC18-7568
- *SQL Reference, Volume 1, V8,* SC09-4844-01
- *SQL Reference, Volume 2, V8,* SC09-4845-01
- *System Monitor Guide and Reference V8,* SC09-4847-01
- *Data Warehouse Center Application Integration Guide Version 8 Release 1*, SC27-1124-01
- *DB2 XML Extender Administration and Programming Guide Version 8 Release 1*, SC27-1234-01
- *Federated Systems PIC Guide Version 8 Release 1*, GC27-1224-01

Online resources

These Web sites are also relevant as further information sources:

- DB2 Information Center

  http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp
- Database and Information Management home page

  http://www.ibm.com/software/data/

- DB2 Universal Database home page

  http://www.ibm.com/software/data/db2/udb/

- DB2 Technical Support

  http://www-3.ibm.com/cgi-bin/db2www/data/db2/udb/winos2unix/support/index.d2w/report

- DB2 online library

  http://www.ibm.com/db2/library

How to get Redbooks publications

You can search for, view, or download IBM Redbooks publications, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

Symbols
.rhosts 50
.ssh 236

A
access method 166
access plan 92, 120
adjustment value 108
administrative view 166
advantage 18
aggregate snapshot data 96
AIC 143
architecture 16
archival logging 20
array 5
asynchronous index cleanup 143
authentication 236

B
binary 98
block 5, 9, 27
block index 9, 178
block prefetching 27
bottlenecks 21
broadcast joins 45
buffer pool 5, 20
bulk data deletion 144
Business Intelligence 22

C
cache 20
cardinality 178, 180
catalog partition 7, 50
catalog system table 160
catalog table 43
cell 13, 178
cell sizes 183
client application 7
clustered data 27
clustering 9
clustering keys 11
collocated join 121
collocated Joins 44
collocated tables 44
collocation 124
column 5
column length 180
concurrent transaction 167
configuration 18
consistency point 159
container 4
container tag 107
contention 144
coordinator node 7
coordinator partition 7, 44
CS 25
cursor stability 25, 166

D
data partition 8, 160
data partition elimination 9
database configuration parameter
  DFT_QUERYOPT 120
  LOGPATH 21
  SHEAPTHRES_SHR 188
  SORTHEAP 188
  UTIL_HEAP_SIZE 188
database configuration switches 97
database managed space 4, 129
database partition 2, 16
db2_all 87, 99
db2_install 46
db2ckbkp 86
db2evmon 98
db2exfmt 103, 161
db2gpmap 57
db2icrt 79
db2nchg 80
db2ncrt 79
db2ndrop 81
db2nodes.cfg 47
db2pd 99, 166
db2rcmd.exe 81
deadlock 167
Decision Support Systems 20

© Copyright IBM Corp. 2007. All rights reserved.

dimension 9, 11, 178
dimension block index 27–28, 179
dimension columns 27
dimension values 13
directed join 121
directed Joins 44
disk 5
distribution key 5, 20, 40, 123
distribution map 5, 20
DMS 4, 129
domain name 49
double buffering 41
DPLSTPRT 161
DPNUMPRT 161
DP-TABLE 161
DSA 235
DSS 20, 28

E
encryption 235–236
equijoin column 41
escalation 26
event monitor 97, 166
exclusive access 167
extent 5, 9, 27
extent size 5, 180

F
Fast Communication Manager 48
file 5
file system cache 41
fixed length 180
forward rebalance 114
FPAGES 197

G
granularity 126

H
hash value 6
hierarchical table 167
high speed network 16
high-water mark 114, 117
history file 85
home directory 47, 236
host name 239
host trust relationship 237
host-based authentication 238
hosts file 49

I
I/O 5, 18
I/O parallelism 19
IBMCATGROUP 39
IBMDEFAULTGROUP 39
IBMTEMPGROUP 39
id_rsa 236
id_rsa.pub 236
index 4
index maintenance 127
index reorganization 23
index scan 161
index statistics 92
index-SARGable 27
instance level 96
instance owner 2
instance owning partition server 236
instance-owning server 2
intent lock 167
internal buffer 94
inter-partition 81
interpartition 18
Interpartition parallelism 19
intrapartition 18
Intrapartition parallelism 19
IP address 49, 239
isolation level 120, 166

K
key pair 236
keyword 97

L
large object 4, 128
leaf page 9
LOB placement option 151
lock escalation 167
log file 20
log space 127
logical database partition 18
logical layer 3
Logical partition 2

M
mass update 167
Massively Parallel Processing 16
MDC 9, 177
memory 2, 16
metadata 7
monitor switch 95
MPP 16
MQT 119
multi-column clustering index 28
multi-partition database 20
multi-partition group 3

N
named pipe 90
netname 48
non-distributed table 232
non-intent locks 26
non-partitioned table 25
normal distribution 183
NPAGES 197

O
OLTP 20
online index creation 129
Online Transaction Processing 20
optimization level 120
option 128

P
page 5
page size 5, 180
parallel 2
parallelism 18
partition attach 8
partition group 3, 5, 20, 39
partition key 8
partitioned environment 5
partitioned instance 2
partitioning key 124
partition-owning server 49
passphrase 236
password 236
performance 5, 22
physical database partition 18
physical node 3
physical objects 128
physical server 2
physical storage 4
point in time 94
pointer 128
predicates 179
prefetch size 5
prefetching 5
prerequisite 178
primary key 124
primary machine 2
Public key-based authentication 236

Q
query 7
query optimization 120
query optimizer 45
query parallelism 19
query performance 22

R
RAID 41
range 8, 106
range specifications 26
range-clustered table 167
raw device 5
RDBMS 1
rebalance 106
rebalancer 115
record 20
recovery 20, 127
Redbooks Web site 248
    Contact us xi
region 128
register variables
    CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION 120
    CURRENT REFRESH AGE 120
regular index 9
remote shell 235
resourcesetname 48
response time 18
reverse rebalance 112, 114, 116
roll-in 8, 22, 126
roll-out 8, 22, 126
round-robin algorithm 6
row count 42
row length 42
row-level index 27, 178

RSA 235
rsh 235

S

scalability 2, 18
scale out 18
scale up 18
Secure shell 235
SET INTEGRITY 127, 174
Setup wizard 49
shared disk 17
shared nothing 16
shared nothing architecture 18
shared-everything architecture 2
shosts.equiv 238
SIGHUP 239
single partition 5
single partition database 20
skew data 56
slice 12
SMP 2, 16
SMS 4, 129, 178
snapshot 94
sorting 20
Space Map Pages 115
space requirements 180
space utilization 23
ssh 235
ssh server configuration file 238
ssh_known_hosts 239
sshd_config 238
ssh-keyscan 237
standard deviation 232
state 97
statistics 180
storing long data remotely 154
strategy 127
stripe 106
stripe set 106
stripe size 41
Symmetric Multiprocessor 16
system catalog 7, 19
System Managed Space 4

T

table function 166
table lock 25
table partition elimination 9
table partitioning 8, 21
table size 23
table sizes 42
table space 4
tape 90
target table 167
terminology 161
text editor 47
throughput 21
timerons 193
transaction 20
transaction log 20
trust mechanism 239
TSM 86
typed table 167

U

uncommitted transaction 160
uniform distribution 56
unique index 124, 160
usage consideration 16
utility parallelism 19

W

workloads 20

X

X/Open Backup Services Application Programmer's Interface 86
XBSA 86

Database Partitioning, Table Partitioning, and MDC for DB2 9

(0.5” spine)
0.475”<->0.875”
250 <-> 459 pages

Database Partitioning, Table Partitioning, and MDC for DB2 9

Differentiating database partitioning, table partitioning, and MDC

Examining implementation examples

Discussing best practices

As organizations strive to do more with less, DB2 Enterprise Server Edition V9 for Linux, Unix, and Windows contains innovative features for delivering information on demand and scaling databases to new levels. The table partitioning, newly introduced in DB2 9, and the database partitioning feature provide scalability, performance, and flexibility for data store. The multi-dimension clustering table enables rows with similar values across multiple dimensions to be physically clustered together on disk. This clustering allows for efficient I/O and provides performance gain for typical analytical queries.

How are these features and functions different? How do you decide which technique is best for your database needs? Can you use more than one technique concurrently?

This IBM Redbooks publication addresses these questions and more. Learn how to set up and administer database partitioning. Explore the table partitioning function and how you can easily add and remove years of data on your warehouse. Analyze your data to discern how multi-dimensional clustering can drastically improve your query performance.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-7467-00 ISBN 0738489220