IBM Db2 V11.5

Partitioning and Clustering Guide
2020-08-21

Notice regarding this document

This document in PDF form is provided as a courtesy to customers who have requested documentation in this format. It is provided As-Is without warranty or maintenance commitment..

About this book

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

Who should use this book

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

How this book is structured

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

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

Parts in the Partitioning and Clustering Guide	**Mapping to Books in the DB2 library**
Part 1. Planning and design considerations	*Database Administration Concepts and Configuration Reference* *Database Security Guide*
Part 2. Installation considerations	*Database Administration Concepts and Configuration Reference* *Installing Db2 Servers*

Table 1. The mapping of this book's Parts to other books in the DB2 library (continued)	
Parts in the Partitioning and Clustering Guide	**Mapping to Books in the DB2 library**
Part 3. Implementation and maintenance	*Data Movement Utilities Guide and Reference* *Data Recovery and High Availability Guide and Reference* *Database Administration Concepts and Configuration Reference* *Upgrading to Db2 Version 10.5* *Database Monitoring Guide and Reference* *XQuery Reference*
Part 4. Performance issues	*Database Administration Concepts and Configuration Reference* *Troubleshooting and Tuning Database Performance*
Part 5. Administrative APIs, commands, SQL statements	*Administrative API Reference* *Administrative Routines and Views* *Command Reference* *Developing ADO.NET and OLE DB Applications* *Developing Embedded SQL Applications* *Developing Java Applications* *Developing Perl, PHP, Python, and Ruby on Rails Applications* *Developing User-defined Routines (SQL and External)* *Getting Started with Database Application Development* *SQL Reference Volume 1* *SQL Reference Volume 2*
Part 6. Appendixes	*Data Recovery and High Availability Guide and Reference* *Installing Db2 Servers* *SQL Reference Volume 1*

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

Part 1. Planning and design considerations

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

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

Part 2. Installation considerations

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

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

Part 3. Implementation and maintenance

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

- Chapter 9, "Before creating a database," describes what should be considered before creating a database, such as enabling parallelism, creating partitioned database environments, creating and configuring database partitions, and establishing communications between database partitions.
- Chapter 10, "Creating and managing partitioned database environments," describes how to create and manage database partitions and partition groups.
- Chapter 11, "Creating tables and other related table objects," presents information on how to create and set up partitioned tables, range-clustered tables, and MDC tables.
- Chapter 12, "Altering a database," describes how to alter an instance and/or a database.
- Chapter 13, "Altering tables and other related table objects," provides information on how to modify partitioned tables.
- Chapter 14, "Load," discusses load considerations in cases of parallelism, multi-dimensional clustering, and partitioned tables.
- Chapter 15, "Loading data in a partitioned database environment," describes how to start, resume, restart or terminate data load operations in partitioned database environments.
- Chapter 16, "Migration of partitioned database environments," briefly gives an overview about migrating partitioned databases and a reference for more detailed information.
- Chapter 17, "Using snapshot and event monitors," gives relevant information about using snapshot monitor results to monitor table reorganization or to assess the global status of a partitioned database system, in addition to describing how to use the CREATE EVENT MONITOR statement.
- Chapter 18, "Developing a good backup and recovery strategy," describes the concepts behind crash recovery in a partitioned database environment which will help to develop backup and recovery strategies before a failure occurs.
- Chapter 19, "Troubleshooting," gives a brief overview of troubleshooting and useful information about how to issue commands useful in troubleshooting, such as **db2trc**, across all computers in the instance, or on all database partition servers.

Part 4. Performance issues

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

- Chapter 20, "Performance issues in database design," describes performance enhancing features of table partitioning and multi-dimensional clustering, including optimization strategies for both.
- Chapter 21, "Indexes," presents conceptual information that is helpful in understanding indexes on partitioned tables.
- Chapter 22, "Design advisor," describes how to use the design advisor to obtain information about migrating from a single-partition to a multi-partition database, as well as recommendations about distributing your data and creating new indexes, materialized query tables, and multi-dimensional clustering tables.

- Chapter 23, "Managing concurrency," provides information about lock modes.
- Chapter 24, "Agent management," describes how to optimize database agents that are used to service application requests.
- Chapter 25, "Optimizing access plans," describes how to improve an access plan, how the optimizer uses information from various scans to optimize data access strategies, and includes information about join strategies, all to improve performance in partitioned database environments, clustered tables, and/or systems using parallelism.
- Chapter 26, "Data redistribution," helps you to determine if data redistribution should be done, and, if so, it describes how to redistribute data across database partitions.
- Chapter 27, "Configuring self-tuning memory," discusses the use of the self-tuning memory feature in a partitioned database environment and provides configuration recommendations.
- Chapter 28, "DB2 configuration parameters and variables," presents information about how to set database configuration parameters and environmental variables across multiple partitions, and lists the parameters and variables related to partitioned database environments and the parallelism feature.

Part 5. Administrative APIs, commands, SQL statements

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

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

Part 6. Appendixes

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

Highlighting conventions

The following highlighting conventions are used in this book.

Bold	Indicates commands, keywords, and other items whose names are predefined by the system.
Italics	Indicates one of the following: • Names or values (variables) that must be supplied by the user • General emphasis • The introduction of a new term • A reference to another source of information

Monospace	Indicates one of the following: • Files and directories • Information that you are instructed to type at a command prompt or in a window • Examples of specific data values • Examples of text similar to what might be displayed by the system • Examples of system messages • Samples of programming code

Contents

Chapter 1. Planning and design considerations

Partitioned databases and tables

Setting up partitioned database environments

The decision to create a multi-partition database must be made before you create your database. As part of the database design decisions you make, you will have to determine if you should take advantage of the performance improvements database partitioning can offer.

About this task

In a partitioned database environment, you still use the **CREATE DATABASE** command or the sqlecrea() function to create a database. Whichever method is used, the request can be made through any of the partitions listed in the `db2nodes.cfg` file. The `db2nodes.cfg` file is the database partition server configuration file.

Except on the Windows operating system environment, any editor can be used to view and update the contents of the database partition server configuration file (`db2nodes.cfg`). On the Windows operating system environment, use **db2ncrt** and **db2nchg** commands to create and change the database partition server configuration file

Before creating a multi-partition database, you must select which database partition will be the catalog partition for the database. You can then create the database directly from that database partition, or from a remote client that is attached to that database partition. The database partition to which you attach and execute the **CREATE DATABASE** command becomes the *catalog partition* for that particular database.

The catalog partition is the database partition on which all system catalog tables are stored. All access to system tables must go through this database partition. All federated database objects (for example, wrappers, servers, and nicknames) are stored in the system catalog tables at this database partition.

If possible, you should create each database in a separate instance. If this is not possible (that is, you must create more than one database per instance), you should spread the catalog partitions among the available database partitions. Doing this reduces contention for catalog information at a single database partition.

Note: You should regularly do a backup of the catalog partition and avoid putting user data on it (whenever possible), because other data increases the time required for the backup.

When you create a database, it is automatically created across all the database partitions defined in the `db2nodes.cfg` file.

When the first database in the system is created, a system database directory is formed. It is appended with information about any other databases that you create. When working on UNIX, the system database directory is `sqldbdir` and is located in the `sqllib` directory under your home directory, or under the directory where Db2® database was installed. When working on UNIX, this directory must reside on a shared file system, (for example, NFS on UNIX platforms) because there is only one system database directory for all the database partitions that make up the partitioned database environment. When working on Windows, the system database directory is located in the instance directory.

Also resident in the `sqldbdir` directory is the system intention file. It is called `sqldbins`, and ensures that the database partitions remain synchronized. The file must also reside on a shared file system since there is only one directory across all database partitions. The file is shared by all the database partitions making up the database.

Configuration parameters have to be modified to take advantage of database partitioning. Use the **GET DATABASE CONFIGURATION** and the **GET DATABASE MANAGER CONFIGURATION** commands to find out the values of individual entries in a specific database, or in the database manager configuration file.

To modify individual entries in a specific database, or in the database manager configuration file, use the **UPDATE DATABASE CONFIGURATION** and the **UPDATE DATABASE MANAGER CONFIGURATION** commands respectively.

The database manager configuration parameters affecting a partitioned database environment include **conn_elapse**, **fcm_num_buffers**, **fcm_num_channels**, **max_connretries**, **max_coordagents**, **max_time_diff**, **num_poolagents**, and **start_stop_time**.

Database partitioning across multiple database partitions
The database manager allows great flexibility in spreading data across multiple database partitions of a partitioned database.

Users can choose how to distribute their data by declaring distribution keys, and can determine which and how many database partitions their table data can be spread across by selecting the database partition group and table space in which the data is to be stored.

In addition, a distribution map (which is updatable) specifies the mapping of distribution key values to database partitions. This makes it possible for flexible workload parallelization across a partitioned database for large tables, while allowing smaller tables to be stored on one or a small number of database partitions if the application designer so chooses. Each local database partition can have local indexes on the data it stores to provide high performance local data access.

In a partitioned database, the distribution key is used to distribute table data across a set of database partitions. Index data is also partitioned with its corresponding tables, and stored locally at each database partition.

Before database partitions can be used to store data, they must be defined to the database manager. Database partitions are defined in a file called `db2nodes.cfg`.

The distribution key for a table in a table space on a partitioned database partition group is specified in the CREATE TABLE statement or the ALTER TABLE statement. When specified through the CREATE TABLE statement the distribution key selection is dependent on the `DISTRIBUTE BY` clause in use:

- If `DISTRIBUTE BY HASH` is specified, the distribution keys are the keys explicitly included in the column list following the HASH keyword.
- If `DISTRIBUTE BY RANDOM` is specified, the distribution key is selected by the database manager in an effort to spread data evenly across all database partitions the table is defined on. There are two methods that the database manager uses to achieve this:
  - **Random by unique**: If the table includes a unique or primary key, it uses the unique characteristics of the key columns to create a random spread of the data. The columns of the unique or primary key are used as the distribution keys.
  - **Random by generation**: If the table does not have a unique or primary key, the database manager will include a column in the table to generate and store a generated value to use in the hashing function. The column will be created with the `IMPLICITLY HIDDEN` clause so that it does not appear in queries unless explicitly included. The value of the column will be automatically generated as new rows are added to the table. By default, the column name is **RANDOM_DISTRIBUTION_KEY**. If it collides with the existing column, a non-conflicting name will be generated by the database manager.
- If `DISTRIBUTE BY REPLICATION` is specified, this means that a copy of all of the data in the table exists on each database partition, so no distribution keys are selected. This option can only be specified for a materialized query table
- If not specified, a distribution key for a table is created by default. A table in a table space that is in a single partition database partition group will have a distribution key only if it is explicitly specified.

Rows are placed in a database partition as follows:

1. A hashing algorithm (database partitioning function) is applied to all of the columns of the distribution key, which results in the generation of a distribution map index value.
2. The database partition number at that index value in the distribution map identifies the database partition in which the row is to be stored.

The database manager supports *partial declustering*, which means that a table can be distributed across a subset of database partitions in the system (that is, a database partition group). Tables do not have to be distributed across all of the database partitions in the system.

The database manager has the capability of recognizing when data being accessed for a join or a subquery is located at the same database partition in the same database partition group. This is known as *table collocation*. Rows in collocated tables with the same distribution key values are located on the same database partition. The database manager can choose to perform join or subquery processing at the database partition in which the data is stored. This can have significant performance advantages.

Random distribution tables that are using random by generation method generally cannot take advantage of table collocation because the distribution key is based on the generated value of the **RANDOM_DISTRIBUTION_KEY** column.

Collocated tables must:

- Be in the same database partition group, one that is not being redistributed. (During redistribution, tables in the database partition group might be using different distribution maps - they are not collocated.)
- Have distribution keys with the same number of columns.
- Have the corresponding columns of the distribution key be database partition-compatible.
- Single-partition tables are collocated only if they are defined in the same database partition group.

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

Consistency across all partitions is recommended.

Database partition groups
A database partition group is a named set of one or more database partitions that belong to a database.

A database partition group that contains more than one database partition is known as a *multiple partition database partition group*. Multiple partition database partition groups can only be defined with database partitions that belong to the same instance.

Figure 1 on page 4 shows an example of a database with five database partitions.

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

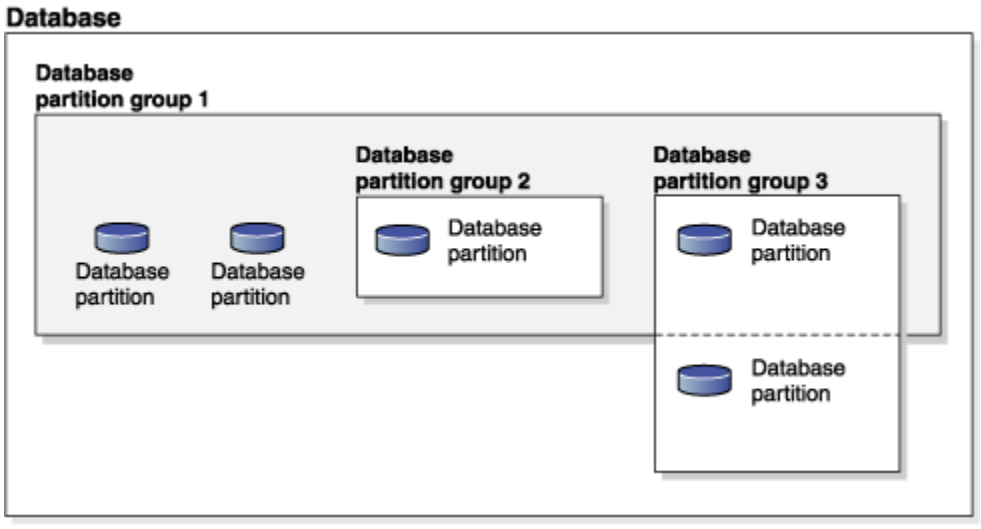


Figure 1. Database partition groups in a database

When a database is created, all database partitions that are specified in the *database partition configuration file* named `db2nodes.cfg` are created as well. Other database partitions can be added or removed with the **ADD DBPARTITIONNUM** or **DROP DBPARTITIONNUM VERIFY** command, respectively. Data is divided across all of the database partitions in a database partition group.

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

Default database partition groups

Three database partition groups are defined automatically at database creation time:

- IBMCATGROUP for the SYSCATSPACE table space, holding system catalog tables
- IBMTEMPGROUP for the TEMPSPACE1 table space, holding temporary tables created during database processing
- IBMDEFAULTGROUP for the USERSPACE1 table space, holding user tables and indexes. A user temporary table space for a declared temporary table or a created temporary table can be created in IBMDEFAULTGROUP or any user-created database partition group, but not in IBMTEMPGROUP.

Table spaces in database partition groups

When a table space is associated with a multiple partition database partition group (during execution of the CREATE TABLESPACE statement), all of the tables within that table space are partitioned across each database partition in the database partition group. A table space that is associated with a particular database partition group cannot later be associated with another database partition group.

Creating a database partition group

Create a database partition group by using the CREATE DATABASE PARTITION GROUP statement. This statement specifies the set of database partitions on which the table space containers and table data are to reside. This statement also performs the following actions:

- It creates a distribution map for the database partition group.
- It generates a distribution map ID.
- It inserts records into the following catalog views:
  - SYSCAT.DBPARTITIONGROUPDEF
  - SYSCAT.DBPARTITIONGROUPS

– SYSCAT.PARTITIONMAPS

Altering a database partition group

Use the ALTER DATABASE PARTITION GROUP statement to add database partitions to (or drop them from) a database partition group. After adding or dropping database partitions, use the **REDISTRIBUTE DATABASE PARTITION GROUP** command to redistribute the data across the set of database partitions in the database partition group.

Database partition group design considerations

Place small tables in single-partition database partition groups, except when you want to take advantage of collocation with a larger table. *Collocation* is the placement of rows from different tables that contain related data in the same database partition. Collocated tables help the database manager to use more efficient join strategies. Such tables can exist in a single-partition database partition group. Tables are considered to be collocated if they are in a multiple partition database partition group, have the same number of columns in the distribution key, and if the data types of corresponding columns are compatible. Rows in collocated tables with the same distribution key value are placed on the same database partition. Tables can be in separate table spaces in the same database partition group, and still be considered collocated.

Avoid extending medium-sized tables across too many database partitions. For example, a 100-MB table might perform better on a 16-partition database partition group than on a 32-partition database partition group.

You can use database partition groups to separate online transaction processing (OLTP) tables from decision support (DSS) tables. This will help to ensure that the performance of OLTP transactions is not adversely affected.

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

- In a multiple partition database partition group, you can only create a unique index if the index is a superset of the distribution key.
- Each database partition must be assigned a unique number, because the same database partition might be found in one or more database partition groups.
- To ensure fast recovery of a database partition containing system catalog tables, avoid placing user tables on the same database partition. Place user tables in database partition groups that do not include the database partition in the IBMCATGROUP database partition group.

Distribution maps

In a partitioned database environment, the database manager must know where to find the data that it needs. The database manager uses a map, called a *distribution map*, to find the data.

A distribution map is an internally generated array containing either 32 768 entries for multiple-partition database partition groups, or a single entry for single-partition database partition groups. For a single-partition database partition group, the distribution map has only one entry containing the number of the database partition where all the rows of a database table are stored. For multiple-partition database partition groups, the numbers of the database partition group are specified in a way such that each database partition is used one after the other to ensure an even distribution across the entire map. Just as a city map is organized into sections using a grid, the database manager uses a *distribution key* to determine the location (the database partition) where the data is stored.

For example, assume that you have a database on four database partitions (numbered 0-3). The distribution map for the IBMDEFAULTGROUP database partition group of this database is:

```
0 1 2 3 0 1 2 ...
```

If a database partition group had been created in the database using database partitions 1 and 2, the distribution map for that database partition group is:

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

If the distribution key for a table to be loaded into the database is an integer with possible values between 1 and 500 000, the distribution key is hashed to a number between 0 and 32 767. That number is used as an index into the distribution map to select the database partition for that row.

Figure 2 on page 6 shows how the row with the distribution key value (c1, c2, c3) is mapped to number 2, which, in turn, references database partition n5.

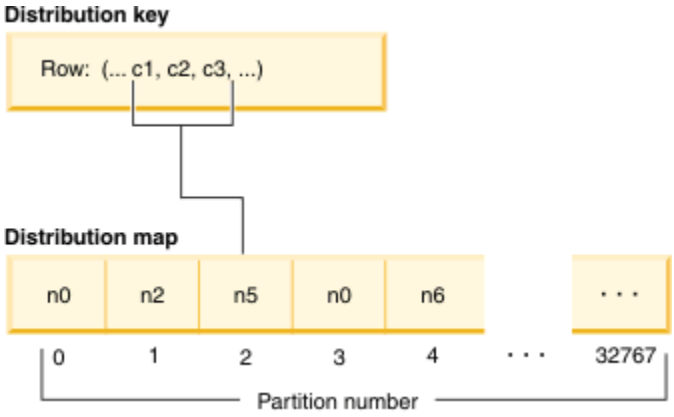


Figure 2. Data distribution using a distribution map

A distribution map is a flexible way of controlling where data is stored in a multi-partition database. If you must change the data distribution across the database partitions in your database, you can use the data redistribution utility. This utility allows you to rebalance or introduce skew into the data distribution.

You can use the `db2GetDistMap` API to obtain a copy of a distribution map that you can view. If you continue to use the `sqlugtpi` API to obtain the distribution information, this API might return error message SQL2768N, because it can only retrieve distribution maps containing 4096 entries.

Distribution keys

A *distribution key* is a column (or group of columns) that is used to determine the database partition in which a particular row of data is stored.

A distribution key is defined on a table using the **CREATE TABLE** statement. The selection of the distribution key is dependent on the **DISTRIBUTE BY** clause in use:

- If **DISTRIBUTE BY HASH** is specified, the distribution keys are the keys explicitly included in the column list following the HASH keyword.
- If **DISTRIBUTE BY RANDOM** is specified, the distribution key is selected by the database manager in an effort to spread data evenly across all database partitions the table is defined on. There are two methods that the database manager uses to achieve this:
  - **Random by unique**: If the table includes a unique or primary key, it uses the unique characteristics of the key columns to create a random spread of the data. The columns of the unique or primary key are used as the distribution keys.
  - **Random by generation**: If the table does not have a unique or primary key, the database manager will include a column in the table to generate and store a generated value to use in the hashing function. The column will be created with the**IMPLICITLY HIDDEN** clause so that it does not appear in queries unless explicitly included. The value of the column will be automatically generated as new rows are added to the table. By default, the column name is **RANDOM_DISTRIBUTION_KEY**. If it collides with the existing column, a non-conflicting name will be generated by the database manager.
- If **DISTRIBUTE BY REPLICATION** is selected, this means that a copy of all of the data in the table exists on each database partition, so no distribution keys are selected. This option can only be specified for a materialized query table.
- If not specified, and the table is defined in a table space that is divided across more than one database partition, a distribution key for a table is created by default from the first column of the primary key. If no primary key is defined, the default distribution key is the first column defined in that table that has a

data type other than a long or a **LOB** data type. Tables in partitioned databases must have at least one column that is neither a long nor a **LOB** data type.

- If not specified and the table is in a table space that is in a single partition database partition group, no distribution key is defined. Tables without a distribution key are only allowed in single-partition database partition groups. You can add or drop distribution keys later, using the **ALTER TABLE** statement. Altering the distribution key can only be done to a table whose table space is associated with a single-partition database partition group.

Choosing a good distribution key is important. Take into consideration:

- How tables are to be accessed
- The nature of the query workload
- The join strategies employed by the database system

If collocation is not a major consideration, a good distribution key for a table is one that spreads the data evenly across all database partitions in the database partition group. The distribution key for each table in a table space that is associated with a database partition group determines if the tables are collocated. Tables are considered collocated when:

- The tables are placed in table spaces that are in the same database partition group
- The distribution keys in each table have the same number of columns
- The data types of the corresponding columns are partition-compatible.

These characteristics ensure that rows of collocated tables with the same distribution key values are located on the same database partition.

An inappropriate distribution key can cause uneven data distribution. Do not choose columns with unevenly distributed data or columns with a small number of distinct values for the distribution key. The number of distinct values must be great enough to ensure an even distribution of rows across all database partitions in the database partition group. The cost of applying the distribution algorithm is proportional to the size of the distribution key. The distribution key cannot be more than 16 columns, but fewer columns result in better performance. Do not include unnecessary columns in the distribution key.

Random distribution can remove the guess work of the distribution key selection. This method will instruct the database manager to pick the distribution keys. It will pick them to ensure that data is spread evenly across all database partitions in the database partition group. However, if the random distribution method is random by generation, you will lose the ability to control collocation and joining of tables cannot be done in an efficient manner. If those will be issues for the expected usage of the table, then explicit selection of the distribution keys is recommended.

Consider the following points when defining a distribution key:

- Creation of a multiple-partition table that contains only BLOB, CLOB, DBCLOB, LONG VARCHAR, LONG VARGRAPHIC, XML, or structured data types is not supported.
- The distribution key definition cannot be altered.
- Include the most frequently joined columns in the distribution key.
- Include columns that often participate in a GROUP BY clause in the distribution key.
- Any unique key or primary key must contain all of the distribution key columns.
- In an online transaction processing (OLTP) environment, ensure that all columns in the distribution key participate in a transaction through equality predicates. For example, assume that you have an employee number column, EMP_NO, that is often used in transactions such as:

```
UPDATE emp_table SET ... WHERE
emp_no = host-variable
```

  In this case, the EMP_NO column makes a good single column distribution key for EMP_TABLE.

Database partitioning is the method by which the placement of each row in the table is determined. The method works as follows:

1. A hashing algorithm is applied to the value of the distribution key, and generates a number between zero (0) and 32 767.
2. The distribution map is created when a database partition group is created. Each of the numbers is sequentially repeated in a round-robin fashion to fill the distribution map.
3. The number is used as an index into the distribution map. The number at that location in the distribution map is the number of the database partition where the row is stored.

Table collocation
If two or more tables frequently contribute data in response to certain queries, you will want related data from these tables to be physically located as close together as possible. In a partitioned database environment, this process is known as *table collocation*.

Tables are collocated when they are stored in the same database partition group, and when their distribution keys are compatible. Placing both tables in the same database partition group ensures a common distribution map. The tables might be in different table spaces, but the table spaces must be associated with the same database partition group. The data types of the corresponding columns in each distribution key must be *partition-compatible*. Collocation is not possible for random distribution tables using the random by generation method.

When more than one table is accessed for a join or a subquery, the database manager determines whether the data to be joined is located at the same database partition. When this happens, the join or subquery is performed at the database partition where the data is stored, instead of having to move data between database partitions. This ability has significant performance advantages.

Partition compatibility
The base data types of corresponding columns of distribution keys are compared and can be declared *partition-compatible*. Partition-compatible data types have the property that two variables, one of each type, with the same value, are mapped to the same number by the same partitioning algorithm.

Partition-compatibility has the following characteristics:

- A base data type is compatible with another of the same base data type.
- Internal formats are used for DATE, TIME, and TIMESTAMP data types. They are not compatible with each other, and none are compatible with character or graphic data types.
- Partition compatibility is not affected by the nullability of a column.
- Partition-compatibility is affected by collation. Locale-sensitive UCA-based collations require an exact match in collation, except that the strength (S) attribute of the collation is ignored. All other collations are considered equivalent for the purposes of determining partition compatibility.
- Character columns defined with FOR BIT DATA are only compatible with character columns without FOR BIT DATA when a collation other than a locale-sensitive UCA-based collation is used.
- NULL values of compatible data types are treated identically; those of non-compatible data types might not be.
- Base data types of a user-defined type are used to analyze partition-compatibility.
- Decimals of the same value in the distribution key are treated identically, even if their scale and precision differ.
- Trailing blanks in character strings (CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC) are ignored by the hashing algorithm.
- BIGINT, SMALLINT, and INTEGER are compatible data types.
- When a locale-sensitive UCA-based collation is used, CHAR, VARCHAR, GRAPHIC, and VARGRAPHIC are compatible data types. When another collation is used, CHAR and VARCHAR of different lengths are compatible types and GRAPHIC and VARGRAPHIC are compatible types, but CHAR and VARCHAR are not compatible types with GRAPHIC and VARGRAPHIC.
- Partition-compatibility does not apply to LONG VARCHAR, LONG VARGRAPHIC, CLOB, DBCLOB, and BLOB data types, because they are not supported as distribution keys.

Partitioned tables

Partitioned tables use a data organization scheme in which table data is divided across multiple storage objects, called *data partitions* or *ranges*, according to values in one or more table partitioning key columns of the table.

A data partition or range is part of a table, containing a subset of rows of a table, and stored separately from other sets of rows. Data from a given table is partitioned into multiple data partitions or ranges based on the specifications provided in the PARTITION BY clause of the CREATE TABLE statement. These data partitions or ranges can be in different table spaces, in the same table space, or a combination of both. If a table is created using the PARTITION BY clause, the table is partitioned.

All of the table spaces specified must have the same page size, extent size, storage mechanism (DMS or SMS), and type (REGULAR or LARGE), and all of the table spaces must be in the same database partition group.

A partitioned table simplifies the rolling in and rolling out of table data and a partitioned table can contain vastly more data than an ordinary table. You can create a partitioned table with a maximum of 32,767 data partitions. Data partitions can be added to, attached to, and detached from a partitioned table, and you can store multiple data partition ranges from a table in one table space.

Indexes on a partitioned table can be partitioned or nonpartitioned. Both nonpartitioned and partitioned indexes can exist together on a single partitioned table.

Restrictions

Partitioned hierarchical or temporary tables, range-clustered tables, and partitioned views are not supported for use in partitioned tables.

Table partitioning

Table partitioning is a data organization scheme in which table data is divided across multiple storage objects called *data partitions* according to values in one or more table columns. Each data partition is stored separately. These storage objects can be in different table spaces, in the same table space, or a combination of both.

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

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

This organization scheme can be used in isolation or in combination with other organization schemes. By combining the DISTRIBUTE BY and PARTITION BY clauses of the CREATE TABLE statement, data can be spread across database partitions spanning multiple table spaces. The organization schemes include:

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

Table partitioning is available with the Db2 Version 9.1 Enterprise Server Edition for Linux, UNIX, and Windows, and later.

Benefits of table partitioning

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

- You have a data warehouse that would benefit from easier roll-in and roll-out of table data.
- You have a data warehouse that includes large tables.

- You are considering a migration to a Version 9.1 database from a previous release or a competitive database product.
- You want to use hierarchical storage management (HSM) solutions more effectively.

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

Efficient roll-in and roll-out

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

With Db2 Version 9.7 Fix Pack 1 and later releases, when detaching a data partition from a partitioned table by using the ALTER TABLE statement with the DETACH PARTITION clause, the source partitioned table remains accessible to dynamic queries that run under the RS, CS, or UR isolation level. Similarly, when attaching a data partition to a partitioned table by using the ALTER TABLE statement with the ATTACH PARTITION clause, the target partitioned table remains accessible to dynamic queries that run under the RS, CS, or UR isolation level.

Easier administration of large tables

Table level administration is more flexible because you can perform administrative tasks on individual data partitions. These tasks include: detaching and reattaching of a data partition, backing up and restoring individual data partitions, and reorganizing individual indexes. Time consuming maintenance operations can be shortened by breaking them down into a series of smaller operations. For example, backup operations can work data partition by data partition when the data partitions are placed in separate table spaces. Thus, it is possible to back up one data partition of a partitioned table at a time.

Flexible index placement

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

- Improved performance when dropping indexes and during 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 that provides 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 index reorganization.
- If you choose to perform index reorganization, an individual index can be reorganized.

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

Improved performance for business intelligence style queries

Query processing is enhanced to automatically eliminate data partitions based on predicates of the query. This query processing is known as *data partition elimination*, and can benefit many decision support queries.

The following example creates a table named CUSTOMER, where rows with l_shipdate >= '01/01/2006' and l_shipdate <= '03/31/2006' are stored in table space TS1, rows with l_shipdate >= '04/01/2006' and l_shipdate <= '06/30/2006' are stored in table space TS2, and so on.

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

Related information

Best practices: Managing data growth

Data partitions and ranges
Partitioned tables use a data organization scheme in which table data is divided across multiple storage objects called *data partitions* according to values in one or more table partitioning key columns of the table. The ranges specified for each data partition can be generated automatically or manually when creating a table.

Data partitions are referred to in various ways throughout the Db2 library. The following list represents the most common references:

- DATAPARTITIONNAME is the permanent name assigned to a data partition for a given table at create time. This column value is stored in the SYSCAT.DATAPARTITIONS catalog view. This name is not preserved on an attach or detach operation.
- DATAPARTITIONID is the permanent identifier assigned to a data partition for a given table at create time. It is used to uniquely identify a particular data partition in a given table. This identifier is not preserved on an attach or detach operation. This value is system-generated and might appear in output from various utilities.
- SEQNO indicates the order of a particular data partition range with regards to other data partition ranges in the table, with detached data partitions sorting after all visible and attached data partitions.

Related information
Best practices: Managing data growth

Data organization schemes
With the introduction of table partitioning, a Db2 database offers a three-level data organization scheme. There are three clauses of the CREATE TABLE statement that include an algorithm to indicate how the data is to be organized.

The following three clauses demonstrate the levels of data organization that can be used together in any combination:

- DISTRIBUTE BY to spread data evenly across database partitions (to enable intraquery parallelism and to balance the load across each database partition) (database partitioning)
- PARTITION BY to group rows with similar values of a single dimension in the same data partition (table partitioning)
- ORGANIZE BY to group rows with similar values on multiple dimensions in the same table extent (multidimensional clustering) or to group rows according to the time of the insert operation (insert time clustering table).

This syntax allows consistency between the clauses and allows for future algorithms of data organization. Each of these clauses can be used in isolation or in combination with one another. By combining the DISTRIBUTE BY and PARTITION BY clauses of the CREATE TABLE statement data can be spread across database partitions spanning multiple table spaces. This approach allows for similar behavior to the Informix® Dynamic Server and Informix Extended Parallel Server hybrid.

In a single table, you can combine the clauses used in each data organization scheme to create more sophisticated partitioning schemes. For example, partitioned database environments are not only compatible, but also complementary to table partitioning.

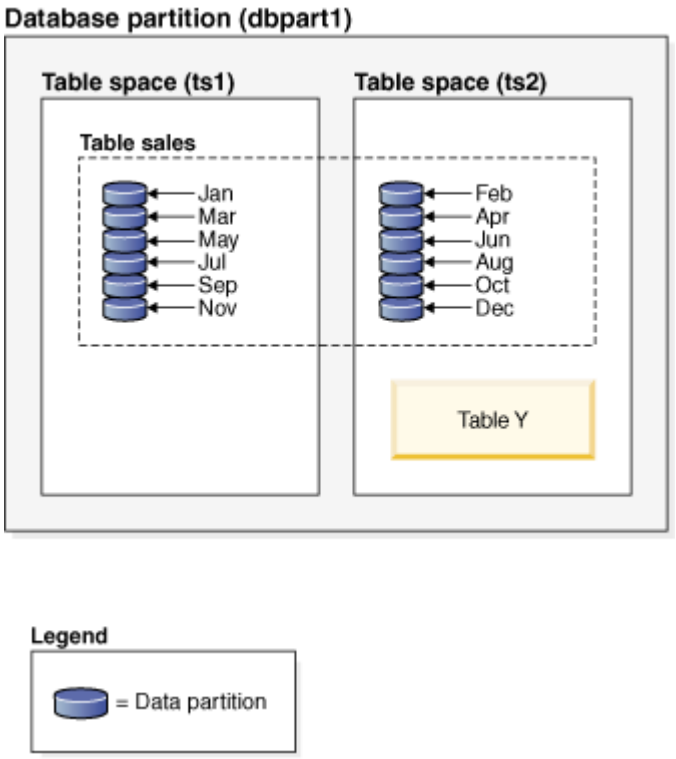


Figure 3. Demonstrating the table partitioning organization scheme where a table representing monthly sales data is partitioned into multiple data partitions. The table also spans two table spaces (ts1 and ts2).

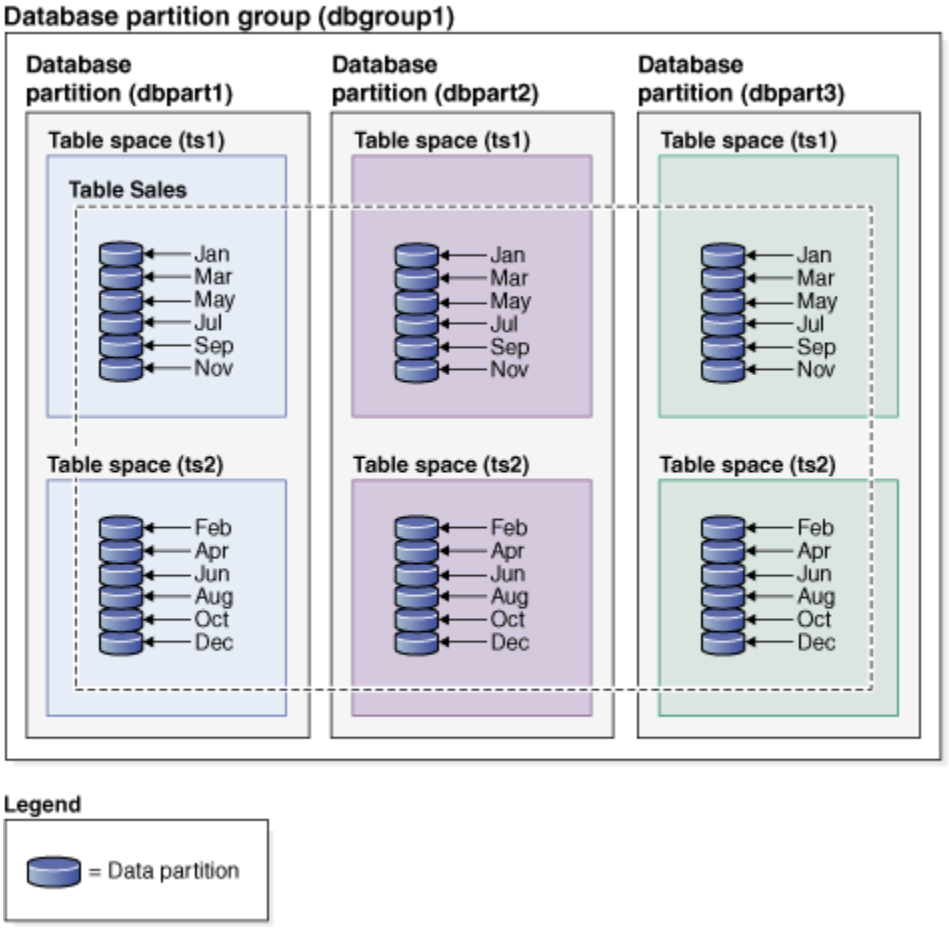


Figure 4. Demonstrating the complementary organization schemes of database partitioning and table partitioning. A table representing monthly sales data is partitioned into multiple data partitions, spanning two table spaces (ts1 and ts2) that are distributed across multiple database partitions (dbpart1, dbpart2, dbpart3) of a database partition group (dbgroup1).

The salient distinction between multidimensional clustering (MDC) and table partitioning is multi-dimension versus single dimension. MDC is suitable to cubes (that is, tables with multiple dimensions), and table partitioning works well if there is a single dimension which is central to the database design, such as a DATE column. MDC and table partitioning are complementary when both of these conditions are met. This is demonstrated in Figure 5 on page 14.

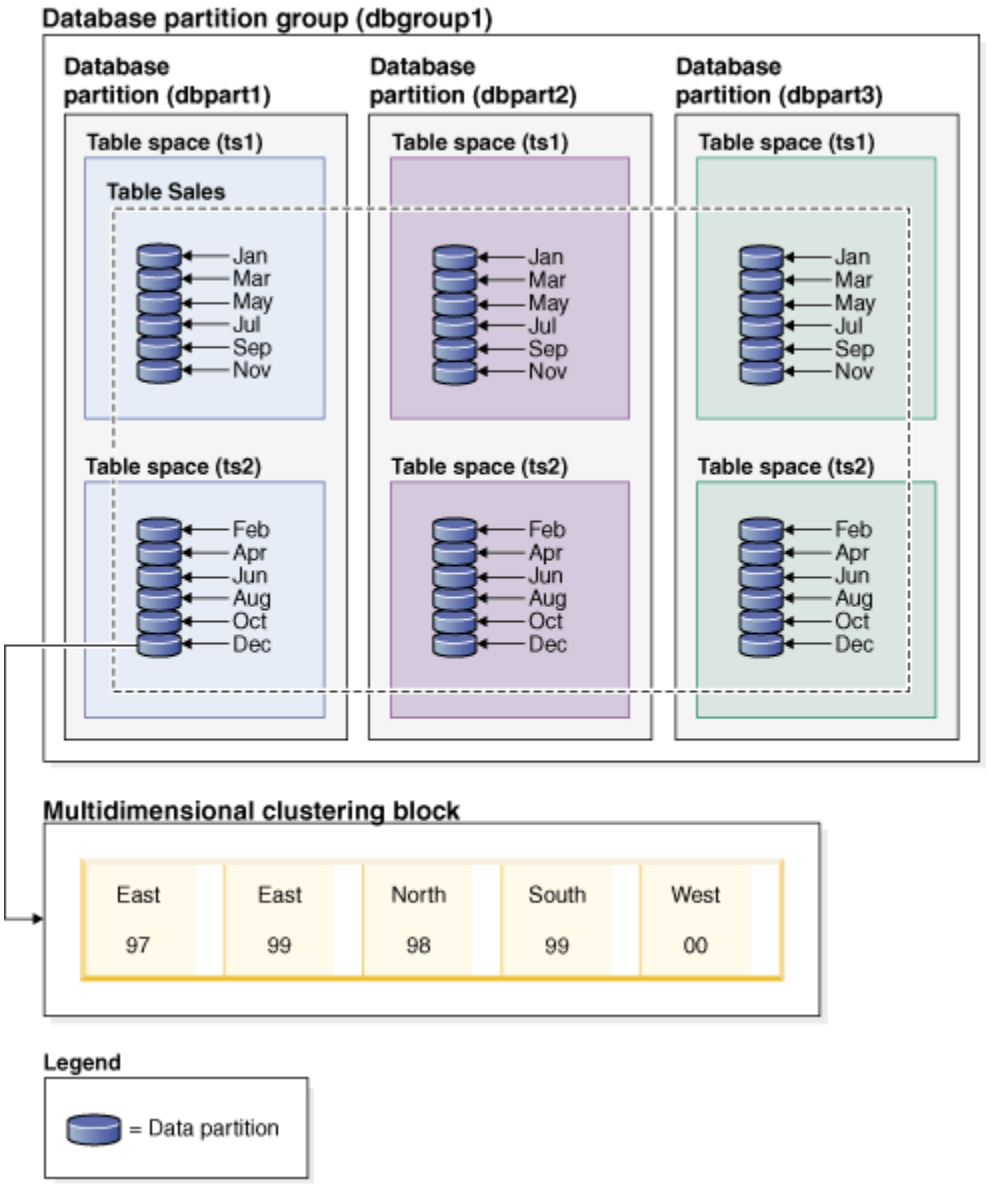


Figure 5. A representation of the database partitioning, table partitioning and multidimensional organization schemes where data from table SALES is not only distributed across multiple database partitions, partitioned across table spaces ts1 and ts2, but also groups rows with similar values on both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

There is another data organization scheme which cannot be used with any of the schemes that were listed previously. This scheme is ORGANIZE BY KEY SEQUENCE. It is used to insert each record into a row that was reserved for that record at the time of table creation (Range-clustered table).

Data organization terminology

Database partitioning
A data organization scheme in which table data is divided across multiple database partitions based on the hash values in one or more distribution key columns of the table, and based on the use of a distribution map of the database partitions. Data from a given table is distributed based on the specifications provided in the DISTRIBUTE BY HASH clause of the CREATE TABLE statement.

Database partition
A portion of a database on a database partition server consisting of its own user data, indexes, configuration file, and transaction logs. Database partitions can be logical or physical.

Table partitioning
A data organization scheme in which table data is divided across multiple data partitions according to values in one or more partitioning columns of the table. Data from a given table is partitioned into multiple storage objects based on the specifications provided in the PARTITION BY clause of the CREATE TABLE statement. These storage objects can be in different table spaces.

Data partition
A set of table rows, stored separately from other sets of rows, grouped by the specifications provided in the PARTITION BY RANGE clause of the CREATE TABLE statement.

Multidimensional clustering (MDC)
A table whose data is physically organized into blocks along one or more dimensions, or clustering keys, specified in the ORGANIZE BY DIMENSIONS clause.

Insert time clustering (ITC)
A table whose data is physically clustered based on row insert time, specified by the ORGANIZE BY INSERT TIME clause.

Benefits of each data organization scheme

Understanding the benefits of each data organization scheme can help you to determine the best approach when planning, designing, or reassessing your database system requirements. Table 2 on page 15 provides a high-level view of common customer requirements and shows how the various data organization schemes can help you to meet those requirements.

Table 2. Using table partitioning with the Database Partitioning Feature

Issue	Recommended scheme	Explanation
Data roll-out	Table partitioning	Uses detach to roll out large amounts of data with minimal disruption
Parallel query execution (query performance)	Database Partitioning Feature	Provides query parallelism for improved query performance
Data partition elimination (query performance)	Table partitioning	Provides data partition elimination for improved query performance
Maximization of query performance	Both	Maximum query performance when used together: query parallelism and data partition elimination are complementary
Heavy administrator workload	Database Partitioning Feature	Execute many tasks for each database partition

Table 3. Using table partitioning with MDC tables

Issue	Recommended scheme	Explanation
Data availability during roll-out	Table partitioning	Use the DETACH PARTITION clause to roll out large amounts of data with minimal disruption.
Query performance	Both	MDC is best for querying multiple dimensions. Table partitioning helps through data partition elimination.
Minimal reorganization	MDC	MDC maintains clustering, which reduces the need to reorganize.

Note: Table partitioning is now recommended over UNION ALL views.

Data organization schemes in Db2 and Informix databases
Table partitioning is a data organization scheme in which table data is divided across multiple storage objects called *data partitions* according to values in one or more table columns. Each data partition is stored separately. These storage objects can be in different table spaces, in the same table space, or a combination of both.

Table data is partitioned as specified in the PARTITION BY clause of the CREATE TABLE statement. The columns used in this definition are referred to as the table partitioning key columns. Db2 table partitioning maps to the data fragmentation approach to data organization offered by Informix Dynamic Server and Informix Extended Parallel Server.

The Informix approach

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

An Informix and Db2 database system comparison

Db2 database provides a rich set of complementary features that map directly to the Informix data organization schemes, making it relatively easy for customers to convert from the Informix syntax to the Db2 syntax. The Db2 database manager handles complicated Informix schemes using a combination of generated columns and the PARTITION BY RANGE clause of the CREATE TABLE statement. Table 4 on page 16 compares data organizations schemes used in Informix and Db2 database products.

Table 4. A mapping of all Informix and Db2 data organization schemes

Data organization scheme	Informix syntax	Db2 Version 9.1 syntax
• Informix: expression-based • Db2: table partitioning	FRAGMENT BY EXPRESSION	PARTITION BY RANGE
• Informix: round-robin • Db2: default	FRAGMENT BY ROUND ROBIN	No syntax: Db2 database manager automatically spreads data among containers
• Informix: range distribution • Db2: table partitioning	FRAGMENT BY RANGE	PARTITION BY RANGE
• Informix: system defined-hash • Db2: database partitioning	FRAGMENT BY HASH	DISTRIBUTE BY HASH
• Informix: HYBRID • Db2: database partitioning with table partitioning	FRAGMENT BY HYBRID	DISTRIBUTE BY HASH, PARTITION BY RANGE
• Informix: n/a • Db2: Multidimensional clustering	n/a	ORGANIZE BY DIMENSIONS

Examples

The following examples provide details on how to accomplish Db2 database equivalent outcomes for any Informix fragment by expression scheme.

Example 1: The following basic create table statement shows Informix fragmentation and the equivalent table partitioning syntax for a Db2 database system:

Informix syntax:

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

Db2 syntax:

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

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

The Db2 database system achieves the equivalent organization scheme to the Informix hybrid using a combination of the DISTRIBUTE BY and PARTITION BY clauses of the CREATE TABLE statement.

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

Informix syntax

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

Db2 syntax

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

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

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

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

Table partitioning applied to common application problems

The following sections discuss how to apply the various features of Db2 table partitioning to common application problems. In each section, particular attention is given to best practices for mapping various Informix fragmentation schemes into equivalent Db2 table partitioning schemes.

Considerations for creating simple data partition ranges

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

Examples

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

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

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

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

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

Notice that the Informix syntax provides an open ended range at the top and bottom to catch dates that are not in the expected range. The Db2 syntax can be modified to match the Informix syntax by adding ranges that make use of MINVALUE and MAXVALUE.

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

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

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

Partition by expression using generated columns

Although Db2 database does not directly support partitioning by expression, partitioning on a generated column is supported, making it possible to achieve the same result.

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

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

Examples

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

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

Example 2: In this example, the Db2 table is partitioned using a generated column:

```
    CREATE TABLE customer (
        cust_id     INT,
```

```
        cust_prov   CHAR(2),
        cust_prov_gen GENERATED ALWAYS AS (CASE
            WHEN cust_prov = 'AB' THEN 1
            WHEN cust_prov = 'BC' THEN 2
            WHEN cust_prov = 'MB' THEN 3
            ...
            WHEN cust_prov = 'YT' THEN 13
            ELSE 14 END))
        IN tbspace_ab, tbspace_bc, tbspace_mb, .... tbspace_remainder
        PARTITION BY RANGE (cust_prov_gen)
            (STARTING 1 ENDING 14 EVERY 1);
```

Here the expressions within the case statement match the corresponding expressions in the FRAGMENT BY EXPRESSION clause. The case statement maps each original expression to a number, which is stored in the generated column (cust_prov_gen in this example). This column is a real column stored on disk, so the table could occupy slightly more space than would be necessary if Db2 supported partition by expression directly. This example uses the short form of the syntax. Therefore, the table spaces in which to place the data partitions must be listed in the IN clause of the CREATE TABLE statement. Using the long form of the syntax requires a separate IN clause for each data partition.

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

Table partitioning keys

A table partitioning key is an ordered set of one or more columns in a table. The values in the table partitioning key columns are used to determine in which data partition each table row belongs.

To define the table partitioning key on a table use the CREATE TABLE statement with the PARTITION BY clause.

Choosing an effective table partitioning key column is essential to taking full advantage of the benefits of table partitioning. The following guidelines can help you to choose the most effective table partitioning key columns for your partitioned table.

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

Supported data types

Table 5 on page 20 shows the data types (including synonyms) that are supported for use as a table partitioning key column:

Table 5. Supported data types

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

Table 5. Supported data types (continued)

Data type column 1	Data type column 2
TIMESTAMP	CHAR VARYING
CHARACTER FOR BIT DATA	CHAR FOR BIT DATA
VARCHAR FOR BIT DATA	CHARACTER VARYING FOR BIT DATA
CHAR VARYING FOR BIT DATA	User defined types (distinct)

Unsupported data types

The following data types can occur in a partitioned table, but are not supported for use as a table partitioning key column:

- User defined types (structured)
- LONG VARCHAR
- LONG VARCHAR FOR BIT DATA
- BLOB
- BINARY LARGE OBJECT
- CLOB
- CHARACTER LARGE OBJECT
- DBCLOB
- LONG VARGRAPHIC
- REF
- Varying length string for C
- Varying length string for Pascal
- XML

If you choose to automatically generate data partitions using the EVERY clause of the CREATE TABLE statement, only one column can be used as the table partitioning key. If you choose to manually generate data partitions by specifying each range in the PARTITION BY clause of the CREATE TABLE statement, multiple columns can be used as the table partitioning key, as shown in the following example:

```
CREATE TABLE sales (year INT, month INT)
   PARTITION BY RANGE(year, month)
   (STARTING FROM (2001, 1) ENDING (2001,3) IN tbsp1,
   ENDING (2001,6) IN tbsp2, ENDING (2001,9)
   IN tbsp3, ENDING (2001,12) IN tbsp4,
   ENDING (2002,3) IN tbsp5, ENDING (2002,6)
   IN tbsp6, ENDING (2002,9) IN tbsp7,
   ENDING (2002,12) IN tbsp8)
```

This results in eight data partitions, one for each quarter in year 2001 and 2002.

Note:

1. When multiple columns are used as the table partitioning key, they are treated as a composite key (which are similar to composite keys in an index), in the sense that trailing columns are dependent on the leading columns. Each starting or ending value (all of the columns, together) must be specified in 512 characters or less. This limit corresponds to the size of the LOWVALUE and HIGHVALUE columns of the SYSCAT.DATAPARTITIONS catalog view. A starting or ending value specified with more than 512 characters will result in error SQL0636N, reason code 9.
2. Table partitioning is multicolumn not multidimension. In table partitioning, all columns used are part of a single dimension.

Generated columns

Generated columns can be used as table partitioning keys. This example creates a table with twelve data partitions, one for each month. All rows for January of any year will be placed in the first data partition, rows for February in the second, and so on.

Example 1

```
CREATE TABLE monthly_sales (sales_date date,
    sales_month int GENERATED ALWAYS AS (month(sales_date)))
        PARTITION BY RANGE (sales_month)
        (STARTING FROM 1 ENDING AT 12 EVERY 1);
```

Note:

1. You cannot alter or drop the expression of a generated column that is used in the table partitioning key. Adding a generated column expression on a column that is used in the table partitioning key is not permitted. Attempting to add, drop or alter a generated column expression for a column used in the table partitioning key results in error (SQL0270N rc=52).
2. Data partition elimination will not be used for range predicates if the generated column is not monotonic, or the optimizer can not detect that it is monotonic. In the presence of non-monotonic expressions, data partition elimination can only take place for equality or IN predicates. For a detailed discussion and examples of monotonicity see “Considerations when creating MDC or ITC tables” on page 39.

Load considerations for partitioned tables

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

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

General load behavior

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

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

issuing any ALTER TABLE ...ATTACH, DETACH, or ADD PARTITION operations while the load utility is running.

Handling of invalid rows

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

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

History file

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

Terminating a load operation

Terminating a load replace completely truncates all visible data partitions, terminating a load insert truncates all visible data partitions to their lengths before the load. Indexes are invalidated during a termination of an ALLOW READ ACCESS load operation that failed in the load copy phase. Indexes are also invalidated when terminating an ALLOW NO ACCESS load operation that touched the index (It is invalidated because the indexing mode is rebuild, or a key was inserted during incremental maintenance leaving the index in an inconsistent state). Loading data into multiple targets does not have any effect on load recovery operations except for the inability to restart the load operation from a consistency point taken during the load phase In this case, the SAVECOUNT load option is ignored if the target table is partitioned. This behavior is consistent with loading data into a MDC target table.

Generated columns

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

Data availability

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

Important: The ALLOW READ ACCESS parameter is deprecated and might be removed in a future release. For more details, see ALLOW READ ACCESS parameter in the LOAD command is deprecated.

The ingest utility also supports partitioned tables and is better suited to allow data concurrency and availability than the LOAD command with the ALLOW READ ACCESS parameter. It can move large amounts of data from files and pipes without locking the target table. In addition, data becomes accessible as soon as it is committed based on elapsed time or number of rows.

Data partition states

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

Error isolation

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

Other considerations

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

Replicated materialized query tables

A *materialized query table* is defined by a query that also determines the data in the table. Materialized query tables can be used to improve the performance of queries. If the database manager determines that a portion of a query can be resolved by using a materialized query table, the query might be rewritten to use the materialized query table.

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

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

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

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

DELETE or UPDATE statements that contain non-deterministic operations are not supported with replicated materialized query tables.

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

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

Table partitioning and multidimensional clustering tables
In a table that is both multidimensional clustered and data partitioned, columns can be used both in the table partitioning range-partition-spec and in the multidimensional clustering (MDC) key. A table that is both multidimensional clustered and partitioned can achieve a finer granularity of data partition and block elimination than could be achieved by either functionality alone.

There are also many applications where it is useful to specify different columns for the MDC key than those on which the table is partitioned. It should be noted that table partitioning is multicolumn, while MDC is multi-dimension.

Characteristics of a mainstream Db2 data warehouse

The following recommendations were focused on typical, mainstream warehouses that were new for Db2 V9.1. The following characteristics are assumed:

- The database runs on multiple machines or multiple AIX® logical partitions.
- Partitioned database environments are used (tables are created using the DISTRIBUTE BY HASH clause).
- There are four to fifty data partitions.
- The table for which MDC and table partitioning is being considered is a major fact table.
- The table has 100,000,000 to 100,000,000,000 rows.
- New data is loaded at various time frames: nightly, weekly, monthly.
- Daily ingest volume is 10 thousand to 10 million records.
- Data volumes vary: The biggest month is 5X the size of the smallest month. Likewise, the biggest dimensions (product line, region) have a 5X size range.
- 1 to 5 years of detailed data is retained.
- Expired data is rolled out monthly or quarterly.
- Tables use a wide range of query types. However, the workload is mostly analytical queries with the following characteristics, relative to OLTP workloads:
  - larger results sets with up to 2 million rows
  - most or all queries are hitting views, not base tables
- SQL clauses selecting data by ranges (BETWEEN clause), items in lists, and so on.

Characteristics of a mainstream Db2 V9.1 data warehouse fact table

A typical warehouse fact table, might use the following design:

- Create data partitions on the Month column.
- Define a data partition for each period you roll-out, for example, 1 month, 3 months.
- Create MDC dimensions on Day and on 1 to 4 additional dimensions. Typical dimensions are: product line and region.
- All data partitions and MDC clusters are spread across all database partitions.

MDC and table partitioning provide overlapping sets of benefits. The following table lists potential needs in your organization and identifies a recommended organization scheme based on the characteristics identified previously.

Table 6. Using table partitioning with MDC tables		
Issue	**Recommended scheme**	**Recommendation**
Data availability during roll-out	Table partitioning	Use the DETACH PARTITION clause to roll out large amounts of data with minimal disruption.
Query performance	Table partitioning and MDC	MDC is best for querying multiple dimensions. Table partitioning helps through data partition elimination.
Minimal reorganization	MDC	MDC maintains clustering, which reduces the need to reorganize.
Rollout a month or more of data during a traditional offline window	Table partitioning	Data partitioning addresses this need fully. MDC adds nothing and would be less suitable on its own.
Rollout a month or more of data during a micro-offline window (less than 1 minute)	Table partitioning	Data partitioning addresses this need fully. MDC adds nothing and would be less suitable on its own.
Rollout a month or more of data while keeping the table fully available for business users submitting queries without any loss of service.	MDC	MDC only addresses this need somewhat. Table partitioning would not be suitable due to the short period the table goes offline.
Load data daily (**LOAD** or **INGEST** command)	Table partitioning and MDC	MDC provides most of the benefit here. Table partitioning provides incremental benefits.
Load data "continually" (**LOAD** command with ALLOW READ ACCESS or **INGEST** command)	Table partitioning and MDC	MDC provides most of the benefit here. Table partitioning provides incremental benefits.
Query execution performance for "traditional BI" queries	Table partitioning and MDC	MDC is especially good for querying cubes/multiple dimensions. Table partitioning helps via partition elimination.
Minimize reorganization pain, by avoiding the need for reorganization or reducing the pain associated with performing the task	MDC	MDC maintains clustering which reduces the need to reorg. If MDC is used, data partitioning does not provide incremental benefits. However if MDC is not used, table partitioning helps reduce the need for reorg by maintaining some course grain clustering at the partition level.

Example 1:

Consider a table with key columns YearAndMonth and Province. A reasonable approach to planning this table might be to partition by date with 2 months per data partition. In addition, you might also organize by Province, so that all rows for a particular province within any two month date range are clustered together, as shown in Figure 6 on page 27.

```
CREATE TABLE orders (YearAndMonth INT, Province CHAR(2))
PARTITION BY RANGE (YearAndMonth)
```

```
(STARTING 9901 ENDING 9904 EVERY 2)
ORGANIZE BY (Province);
```

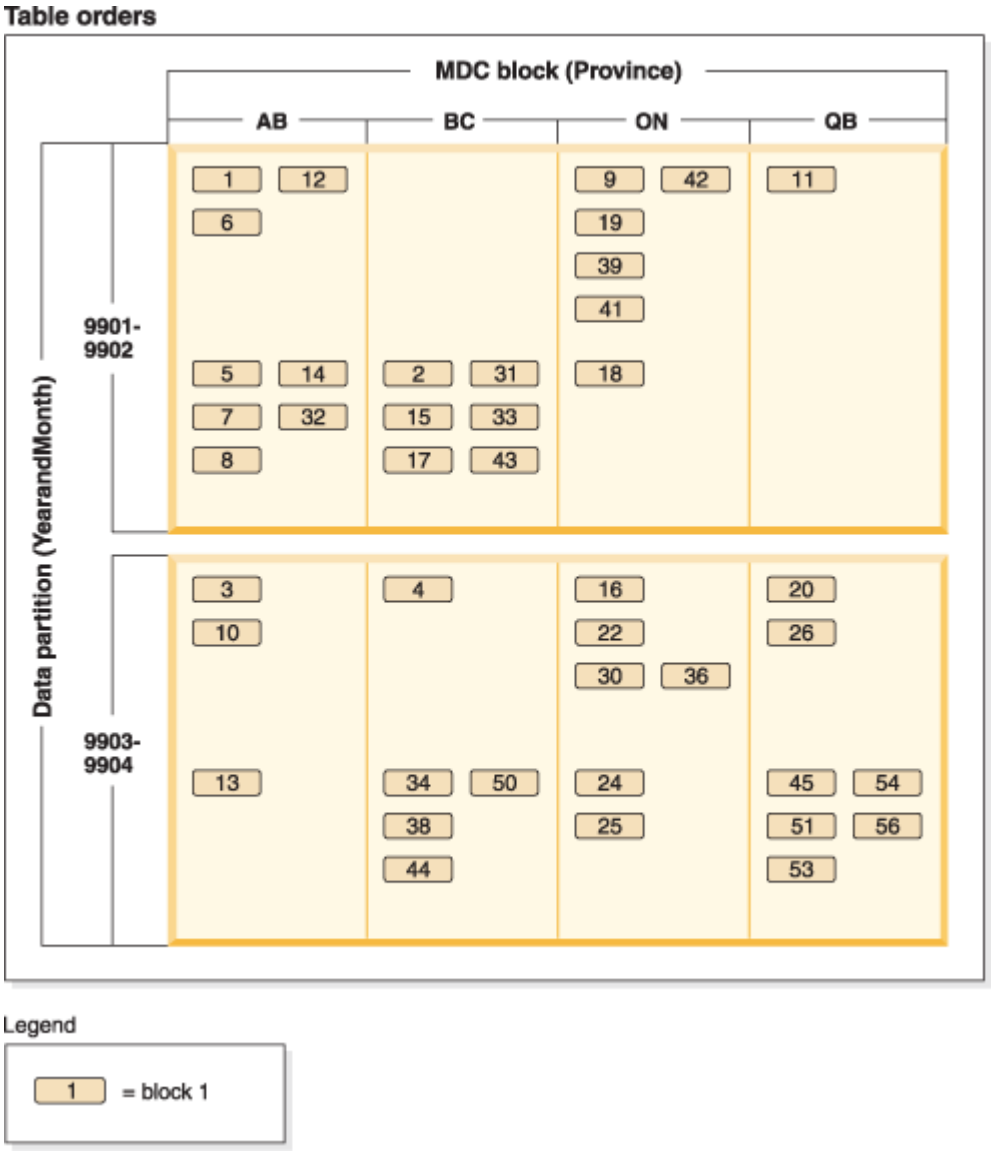


Figure 6. A table partitioned by YearAndMonth and organized by Province

Example 2:

Finer granularity can be achieved by adding YearAndMonth to the ORGANIZE BY DIMENSIONS clause, as shown in Figure 7 on page 28.

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

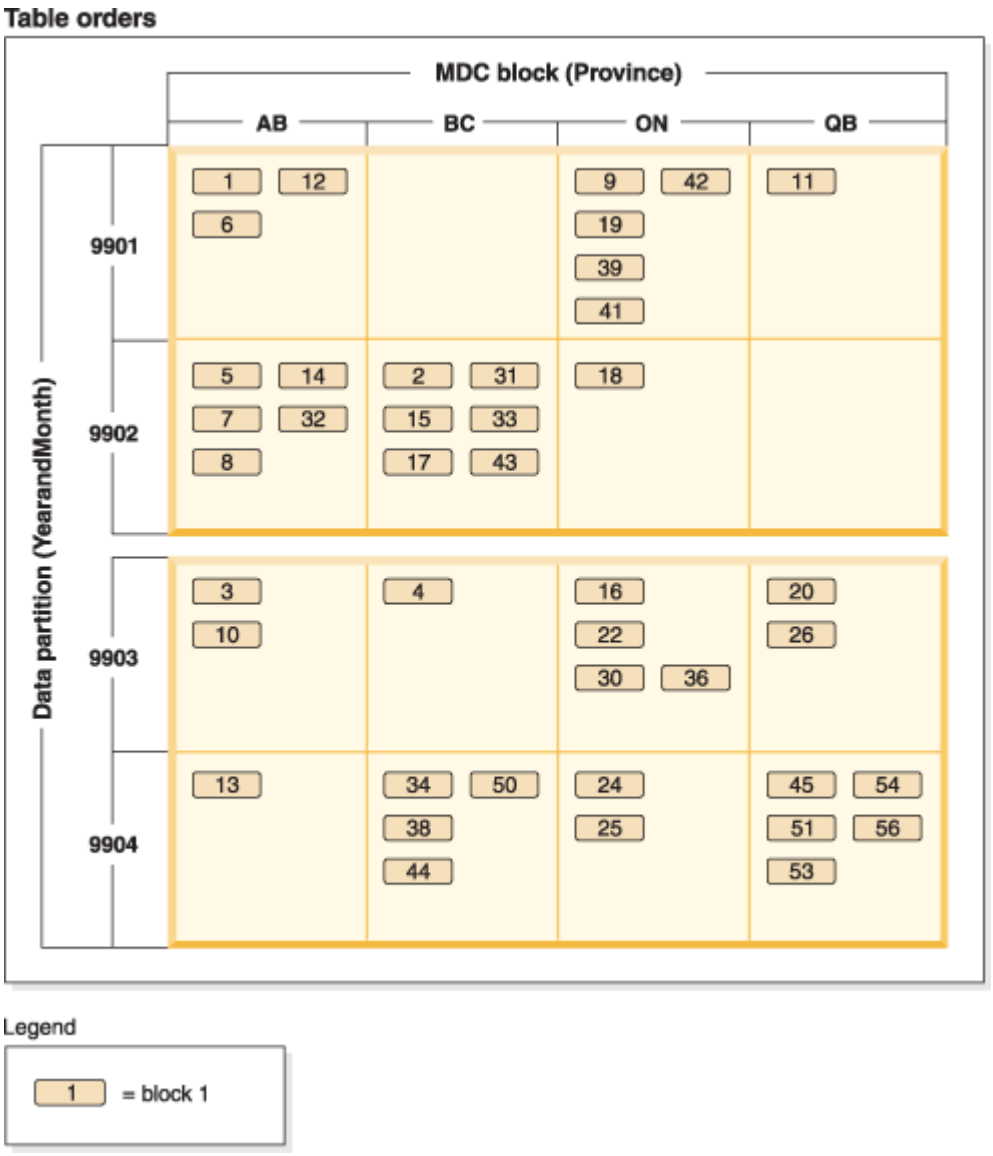


Figure 7. A table partitioned by YearAndMonth and organized by Province and YearAndMonth

In cases where the partitioning is such that there is only a single value in each range, nothing is gained by including the table partitioning column in the MDC key.

Considerations

- Compared to a basic table, both MDC tables and partitioned tables require more storage. These storage needs are additive but are considered reasonable given the benefits.
- If you choose not to combine table partitioning and MDC functionality in your partitioned database environment, table partitioning is best in cases where you can confidently predict the data distribution, which is generally the case for the types of systems discussed here. Otherwise, MDC should be considered.
- For a data-partitioned MDC table created with Db2 Version 9.7 Fix Pack 1 or later releases, the MDC block indexes on the table are partitioned. For a data-partitioned MDC table created with Db2 V9.7 or earlier releases, the MDC block indexes on the table are nonpartitioned.

Table partitioning in a Db2 pureScale environment
You can use table partitioning in Db2 pureScale to divide large table objects between multiple partitions for better performance.

You can use table partitioning in Db2 pureScale tables; this includes tables that use the PARTITION BY RANGE clause. In addition, the commands associated with table partitioning can be used in a Db2 pureScale environment.

This means, for example, that all of the following operations are supported:

- The roll-in and roll-out partition operations available through the ALTER TABLE statement
- The PARTITIONED and NOT PARTITIONED clauses for the CREATE INDEX statement
- For partitioned indexes, the ON DATA PARTITION clause of the REORG TABLE and REORG INDEXES ALL statements

In addition, the MON_GET_PAGE_ACCESS_INFO table function has been updated to work with partitioned tables. All existing monitoring functions that operate on data partitions will work with Db2 pureScale tables.

If you are already using the Db2 pureScale Feature, you can use table partitioning to help resolve page contention issues. By spreading contention out over a larger range, you can reduce data page contention; similarly, you can reduce contention with index pages by using partitioned indexes.

Note: From a Db2 pureScale performance perspective, the amount of memory used depends on the number of table partitions and the number of indexes. The memory resource used for the partitioning on the member comes from the **dbheap** configuration parameter. On the CF, the memory resource is defined by the **cf_sca_sz** configuration parameter.

Range-clustered tables

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

An algorithm is used to associate a record key value with the location of a specific table row. This approach provides exceptionally fast access to specific table rows. The algorithm does not use hashing, because hashing does not preserve key-value order. Preserving this order eliminates the need to reorganize the table data over time.

Each record key value in the table must be:

- Unique
- Not null
- An integer (SMALLINT, INTEGER, or BIGINT)
- Monotonically increasing
- Within a predetermined set of ranges based on each column in the key. (If necessary, use the ALLOW OVERFLOW option on the CREATE TABLE statement to allow rows with key values that are outside of the defined range of values.)

In addition to direct access to specific table rows, there are other advantages to using range-clustered tables.

- Less maintenance is required. A secondary structure, such as a B+ tree index, which would need to be updated after every insert, update, or delete operation, does not exist.
- Less logging is required for RCTs, when compared to similarly-sized regular tables with B+ tree indexes.
- Less buffer pool memory is required. There is no additional memory required to store a secondary structure, such as a B+ tree index.

Space for an RCT is pre-allocated and reserved for use by the table even when records do not yet exist. Consequently, range-clustered tables have no need for free space control records (FSCR). At table creation time, there are no records in the table; however, the entire range of pages is pre-allocated.

Preallocation is based on the record size and the maximum number of records to be stored. If a variable-length field (such as VARCHAR) is defined, the maximum length of the field is used, and the overall record size is of fixed length. This can result in less than optimal use of space. If key values are sparse, the unused space has a negative impact on range scan performance. Range scans must visit all possible rows within a range, even rows that do not yet contain data.

If a schema modification on a range-clustered table is required, the table must be recreated with a new schema name and then populated with the data from the old table. For example, if a table's ranges need to be altered, create a table with new ranges and populate it with data from the old table.

If an RCT allows overflow records, and a new record has a key value that falls outside of the defined range of values, the record is placed in an overflow area, which is dynamically allocated. As more records are added to this overflow area, operations against the table that involve the overflow area require more processing time. The larger the overflow area, the more time is required to access it. If this becomes a problem, consider reducing the size of the overflow area by exporting the data to a new RCT with wider ranges.

Restrictions on range-clustered tables

There are contexts in which range-clustered tables cannot be used, and there are certain utilities that cannot operate on range-clustered tables.

The following restrictions apply to range-clustered tables:

- Range-clustered tables cannot be specified in a Db2 pureScale environment (SQLSTATE 42997).
- Partitioned tables cannot be range-clustered tables.
- Declared temporary tables and created temporary tables cannot be range-clustered tables.
- Automatic summary tables (AST) cannot be range-clustered tables.
- The load utility is not supported. Data can be inserted into a range-clustered table through the import utility or through a parallel insert application.
- The **REORG** utility is not supported. Range-clustered tables that are defined with the DISALLOW OVERFLOW option do not need to be reorganized. Range-clustered tables that are defined with the ALLOW OVERFLOW option cannot have the data in this overflow region reorganized.
- The DISALLOW OVERFLOW clause on the CREATE TABLE statement cannot be specified if the table is a range-clustered materialized query table.
- The design advisor will not recommend range-clustered tables.
- Multidimensional clustering and clustering indexes are incompatible with range-clustered tables.
- Value and default compression are not supported.
- Reverse scans on range-clustered tables are not supported.
- The **REPLACE** parameter on the **IMPORT** command is not supported.
- The WITH EMPTY TABLE option on the ALTER TABLE...ACTIVATE NOT LOGGED INITIALLY statement is not supported.

Multi-dimensional clustered (MDC) tables

Multidimensional clustering tables

Multidimensional clustering (MDC) provides an elegant method for clustering data in tables along multiple dimensions in a flexible, continuous, and automatic way. MDC can significantly improve query performance.

In addition, MDC can significantly reduce the overhead of data maintenance, such as reorganization and index maintenance operations during insert, update, and delete operations. MDC is primarily intended for data warehousing and large database environments, but it can also be used in online transaction processing (OLTP) environments.

Related information

Best practices: Managing data growth

Comparison of regular and MDC tables

Regular tables have indexes that are record-based. Any clustering of the indexes is restricted to a single dimension. Prior to Version 8, the database manager supported only single-dimensional clustering of data, through clustering indexes. Using a clustering index, the database manager attempts to maintain the physical order of data on pages in the key order of the index when records are inserted and updated in the table.

Clustering indexes greatly improve the performance of range queries that have predicates containing the key (or keys) of the clustering index. Performance is improved with a good clustering index because only a portion of the table needs to be accessed, and more efficient prefetching can be performed.

Data clustering using a clustering index has some drawbacks, however. First, because space is filled up on data pages over time, clustering is not guaranteed. An insert operation will attempt to add a record to a page nearby to those having the same or similar clustering key values, but if no space can be found in the ideal location, it will be inserted elsewhere in the table. Therefore, periodic table reorganizations may be necessary to re-cluster the table and to set up pages with additional free space to accommodate future clustered insert requests.

Second, only one index can be designated as the "clustering" index, and all other indexes will be unclustered, because the data can only be physically clustered along one dimension. This limitation is related to the fact that the clustering index is record-based, as all indexes have been prior to Version 8.1.

Third, because record-based indexes contain a pointer for every single record in the table, they can be very large in size.

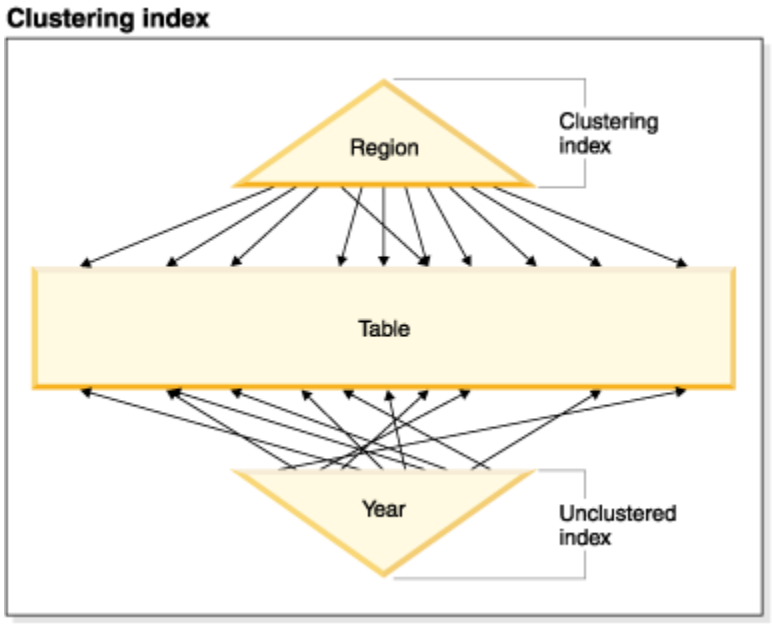


Figure 8. A regular table with a clustering index

The table in Figure 8 on page 31 has two record-based indexes defined on it:

- A clustering index on "Region"
- Another index on "Year"

The "Region" index is a clustering index which means that as keys are scanned in the index, the corresponding records should be found for the most part on the same or neighboring pages in the table. In contrast, the "Year" index is unclustered which means that as keys are scanned in that index, the corresponding records will likely be found on random pages throughout the table. Scans on the clustering

index will exhibit better I/O performance and will benefit more from sequential prefetching, the more clustered the data is to that index.

MDC introduces indexes that are block-based. "Block indexes" point to blocks or groups of records instead of to individual records. By physically organizing data in an MDC table into blocks according to clustering values, and then accessing these blocks using block indexes, MDC is able not only to address all of the drawbacks of clustering indexes, but to provide significant additional performance benefits.

First, MDC enables a table to be physically clustered on more than one key, or dimension, simultaneously. With MDC, the benefits of single-dimensional clustering are therefore extended to multiple dimensions, or clustering keys. Query performance is improved where there is clustering of one or more specified dimensions of a table. Not only will these queries access only those pages having records with the correct dimension values, these qualifying pages will be grouped into blocks, or extents.

Second, although a table with a clustering index can become unclustered over time, in most cases an MDC table is able to maintain and guarantee its clustering over all dimensions automatically and continuously. This eliminates the need to frequently reorganize MDC tables to restore the physical order of the data. While record order within blocks is always maintained, the physical ordering of blocks (that is, from one block to another, in a block index scan) is not maintained on inserts (or even on the initial load, in some cases).

Third, in MDC the clustering indexes are block-based. These indexes are drastically smaller than regular record-based indexes, so take up much less disk space and are faster to scan.

Choosing MDC table dimensions

After you have decided to work with multidimensional clustering tables, the dimensions that you choose will depend not only on the type of queries that will use the tables and benefit from block-level clustering, but even more importantly on the amount and distribution of your actual data.

Queries that will benefit from MDC

The first consideration when choosing clustering dimensions for your table is the determination of which queries will benefit from clustering at a block level. Typically, there will be several candidates when choosing dimensions based on the queries that make up the work to be done on the data. The ranking of these candidates is important. Columns that are involved in equality or range predicate queries, and especially columns with low cardinalities, show the greatest benefit from clustering dimensions. Consider creating dimensions for foreign keys in an MDC fact table involved in star joins with dimension tables. Keep in mind the performance benefits of automatic and continuous clustering on more than one dimension, and of clustering at the extent or block level.

There are many queries that can take advantage of multidimensional clustering. Examples of such queries follow. In some of these examples, assume that there is an MDC table t1 with dimensions c1, c2, and c3. In the other examples, assume that there is an MDC table mdctable with dimensions color and nation.

Example 1:

```
SELECT .... FROM t1 WHERE c3 < 5000
```

This query involves a range predicate on a single dimension, so it can be internally rewritten to access the table using the dimension block index on c3. The index is scanned for block identifiers (BIDs) of keys having values less than 5000, and a mini-relational scan is applied to the resulting set of blocks to retrieve the actual records.

Example 2:

```
SELECT .... FROM t1 WHERE c2 IN (1,2037)
```

This query involves an IN predicate on a single dimension, and can trigger block index based scans. This query can be internally rewritten to access the table using the dimension block index on c2. The index is scanned for BIDs of keys having values of 1 and 2037, and a mini-relational scan is applied to the resulting set of blocks to retrieve the actual records.

Example 3:

```
SELECT * FROM MDCTABLE WHERE COLOR='BLUE' AND NATION='USA'
```

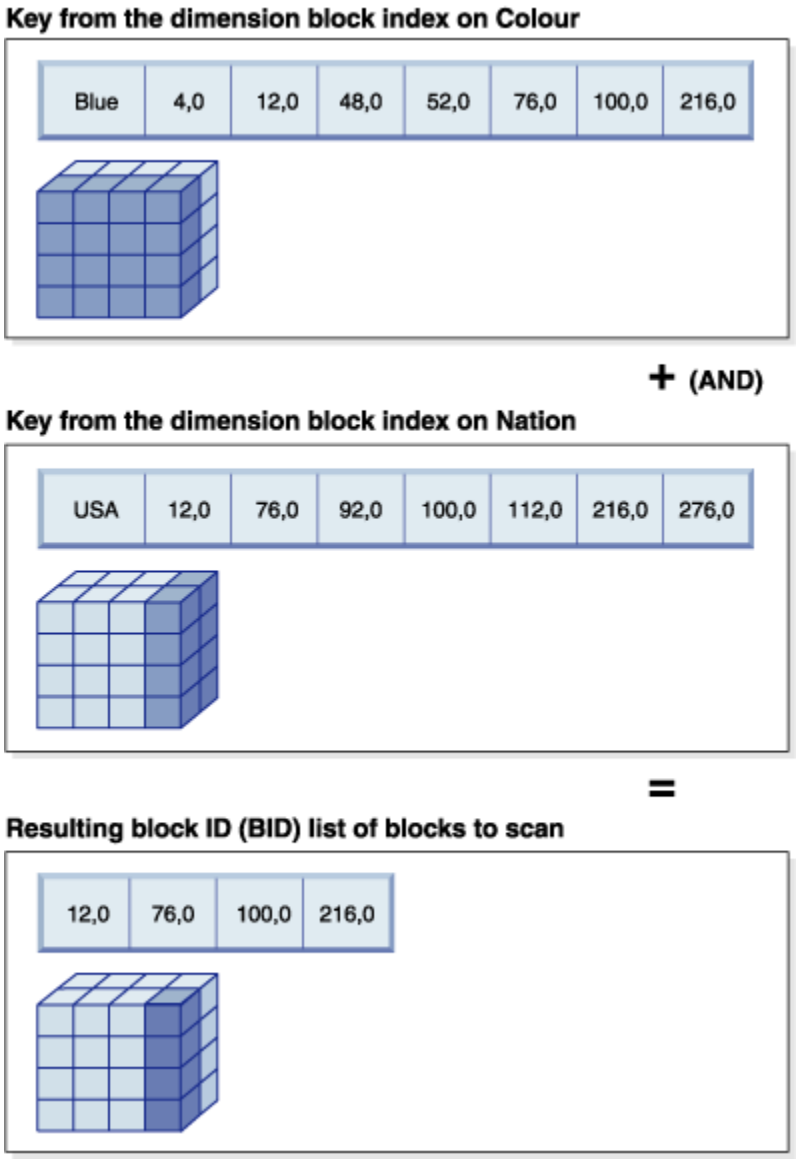


Figure 9. A query request that uses a logical AND operation with two block indexes

To carry out this query request, the following is done (and is shown in Figure 9 on page 33):

- A dimension block index lookup is done: one for the Blue slice and another for the USA slice.
- A block logical AND operation is carried out to determine the intersection of the two slices. That is, the logical AND operation determines only those blocks that are found in both slices.
- A mini-relation scan of the resulting blocks in the table is carried out.

Example 4:

```
SELECT ... FROM t1
   WHERE c2 > 100 AND c1 = '16/03/1999' AND c3 > 1000 AND c3 < 5000
```

This query involves range predicates on c2 and c3 and an equality predicate on c1, along with a logical AND operation. This can be internally rewritten to access the table on each of the dimension block indexes:

- A scan of the c2 block index is done to find BIDs of keys having values greater than 100

- A scan of the c3 block index is done to find BIDs of keys having values between 1000 and 5000
- A scan of the c1 block index is done to find BIDs of keys having the value '16/03/1999'.

A logical AND operation is then done on the resulting BIDs from each block scan, to find their intersection, and a mini-relational scan is applied to the resulting set of blocks to find the actual records.

Example 5:

```
SELECT * FROM MDCTABLE WHERE COLOR='BLUE' OR NATION='USA'
```

To carry out this query request, the following is done:

- A dimension block index lookup is done: one for each slice.
- A logical OR operation is done to find the union of the two slices.
- A mini-relation scan of the resulting blocks in the table is carried out.

Example 6:

```
SELECT .... FROM t1 WHERE c1 < 5000 OR c2 IN (1,2,3)
```

This query involves a range predicate on the c1 dimension, an IN predicate on the c2 dimension, and a logical OR operation. This can be internally rewritten to access the table on the dimension block indexes c1 and c2. A scan of the c1 dimension block index is done to find values less than 5000 and another scan of the c2 dimension block index is done to find values 1, 2, and 3. A logical OR operation is done on the resulting BIDs from each block index scan, then a mini-relational scan is applied to the resulting set of blocks to find the actual records.

Example 7:

```
SELECT .... FROM t1 WHERE c1 = 15 AND c4 < 12
```

This query involves an equality predicate on the c1 dimension and another range predicate on a column that is not a dimension, along with a logical AND operation. This can be internally rewritten to access the dimension block index on c1, to get the list of blocks from the slice of the table having value 15 for c1. If there is a RID index on c4, an index scan can be done to retrieve the RIDs of records having c4 less than 12, and then the resulting list of blocks undergoes a logical AND operation with this list of records. This intersection eliminates RIDs not found in the blocks having c1 of 15, and only those listed RIDs found in the blocks that qualify are retrieved from the table.

If there is no RID index on c4, then the block index can be scanned for the list of qualifying blocks, and during the mini-relational scan of each block, the predicate c4 < 12 can be applied to each record found.

Example 8:

Given a scenario where there are dimensions for color, year, nation and a row ID (RID) index on the part number, the following query is possible.

```
SELECT * FROM MDCTABLE WHERE COLOR='BLUE' AND PARTNO < 1000
```

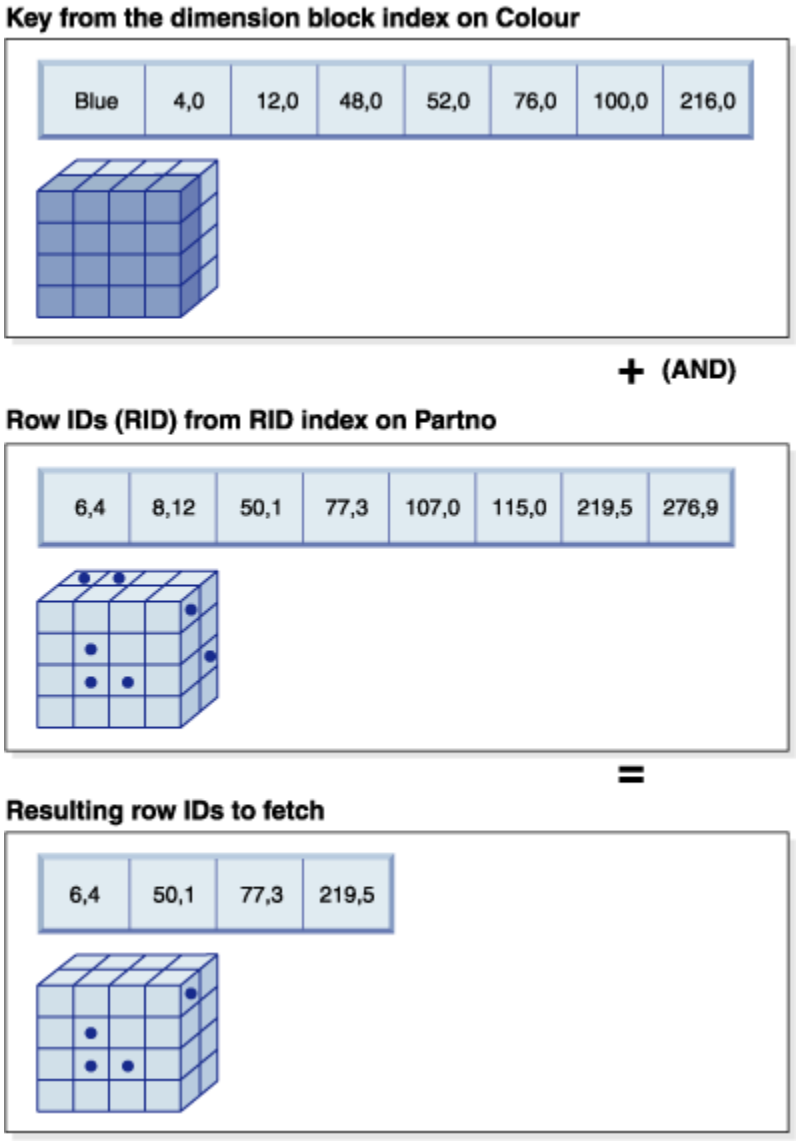


Figure 10. A query request that uses a logical AND operation on a block index and a row ID (RID) index

To carry out this query request, the following is done (and is shown in Figure 10 on page 35):

- A dimension block index lookup and a RID index lookup are done.
- A logical AND operation is used with the blocks and RIDs to determine the intersection of the slice and those rows meeting the predicate condition.
- The result is only those RIDs that also belong to the qualifying blocks.

Example 9:

```
SELECT * FROM MDCTABLE WHERE COLOR='BLUE' OR PARTNO < 1000
```

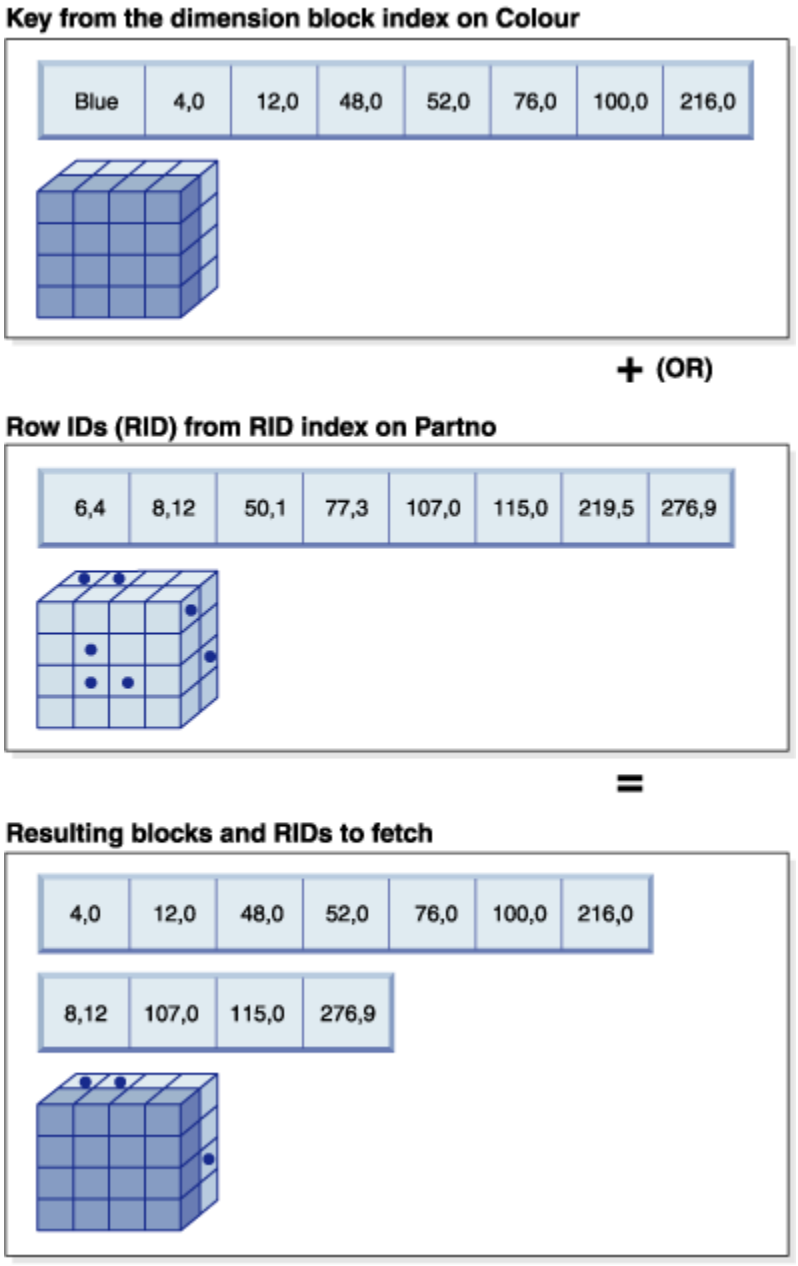


Figure 11. How block index and row ID using a logical OR operation works

To carry out this query request, the following is done (and is shown in Figure 11 on page 36):

- A dimension block index lookup and a RID index lookup are done.
- A logical OR operation is used with the blocks and RIDs to determine the union of the slice and those rows meeting the predicate condition.
- The result is all of the rows in the qualifying blocks, plus additional RIDs that fall outside the qualifying blocks that meet the predicate condition. A mini-relational scan of each of the blocks is performed to retrieve their records, and the additional records outside these blocks are retrieved individually.

Example 10:

```
SELECT ... FROM t1 WHERE c1 < 5 OR c4 = 100
```

This query involves a range predicate on dimension c1, an equality predicate on a non-dimension column c4, and a logical OR operation. If there is a RID index on the c4 column, this might be internally rewritten to do a logical OR operation using the dimension block index on c1 and the RID index on c4. If there is no

index on c4, a table scan might be chosen instead, because all records must be checked. The logical OR operation uses a block index scan on c1 for values less than 4, and a RID index scan on c4 for values of 100. A mini-relational scan is performed on each block that qualifies, because all records within those blocks will qualify, and any additional RIDs for records outside of those blocks are retrieved as well.

Example 11:

```
SELECT .... FROM t1,d1,d2,d3
   WHERE t1.c1 = d1.c1 and d1.region = 'NY'
      AND t2.c2 = d2.c3 and d2.year='1994'
      AND t3.c3 = d3.c3 and d3.product='basketball'
```

This query involves a star join. In this example, t1 is the fact table and it has foreign keys c1, c2, and c3, corresponding to the primary keys of d1, d2, and d3, the dimension tables. The dimension tables do not need to be MDC tables. Region, year, and product are columns of the dimension tables that can be indexed using regular or block indexes (if the dimension tables are MDC tables). When accessing the fact table on c1, c2, and c3 values, block index scans of the dimension block indexes on these columns can be done, followed by a logical AND operation using the resulting BIDs. When there is a list of blocks, a mini-relational scan can be done on each block to get the records.

Density of cells

The choices made for the appropriate dimensions and for the extent size are of **critical** importance to MDC design. These factors determine the table's expected cell density. They are important because an extent is allocated for every existing cell, regardless of the number of records in the cell. The right choices will take advantage of block-based indexing and multidimensional clustering, resulting in performance gains. The goal is to have densely-filled blocks to get the most benefit from multidimensional clustering, and to get optimal space utilization.

Thus, a very important consideration when designing a multidimensional table is the expected density of cells in the table, based on present and anticipated data. You can choose a set of dimensions, based on query performance, that cause the potential number of cells in the table to be very large, based on the number of possible values for each of the dimensions. The number of possible cells in the table is equal to the Cartesian product of the cardinalities of each of the dimensions. For example, if you cluster the table on dimensions Day, Region and Product and the data covers 5 years, you might have `1821 days * 12 regions * 5 products = 109 260` different possible cells in the table. Any cell that contains only a few records still requires an entire block of pages to store its records. If the block size is large, this table might end up being much larger than it needs to be.

There are several design factors that can contribute to optimal cell density:

- Varying the number of dimensions.
- Varying the granularity of one or more dimensions.
- Varying the block (extent) size and page size of the table space.

Carry out the following steps to achieve the best design possible:

1. Identify candidate dimensions.

   Determine which queries will benefit from block-level clustering. Examine the potential workload for columns which have some or all of the following characteristics:

   - Range and equality of any IN-list predicates
   - Roll-in or roll-out of data
   - Group-by and order-by clauses
   - Join clauses (especially in star schema environments).
2. Estimate the number of cells.

   Identify how many potential cells are possible in a table organized along a set of candidate dimensions. Determine the number of unique combinations of the dimension values that occur in the data. If the table exists, an exact number can be determined for the current data by selecting the

number of distinct values in each of the columns that will be dimensions for the table. Alternatively, an approximation can be determined if you only have the statistics for a table, by multiplying the column cardinalities for the dimension candidates.

Note: If your table is in a partitioned database environment, and the distribution key is not related to any of the dimensions considered, determine an average amount of data per cell by taking all of the data and dividing by the number of database partitions.

3. Estimate the space occupancy or density.

   On average, consider that each cell has one partially-filled block where only a few rows are stored. There will be more partially-filled blocks as the number of rows per cell becomes smaller. Also, note that on average (assuming little or no data skew), the number of records per cell can be found by dividing the number of records in the table by the number of cells. However, if your table is in a partitioned database environment, consider how many records there are per cell on each database partition, because blocks are allocated for data on a database partition basis. When estimating the space occupancy and density in a partitioned database environment, consider the average number of records per cell on each database partition, not across the entire table.

   There are several ways to improve the density:

   - Reduce the block size so that partially-filled blocks take up less space.

     Reduce the size of each block by making the extent size appropriately small. Each cell that has a partially-filled block, or that contains only one block with few records on it, wastes less space. The trade-off, however, is that for those cells having many records, more blocks are needed to contain them. This increases the number of block identifiers (BIDs) for these cells in the block indexes, making these indexes larger and potentially resulting in more inserts and deletes to these indexes as blocks are more quickly emptied and filled. It also results in more small groupings of clustered data in the table for these more populated cell values, versus a smaller number of larger groupings of clustered data.

   - Reduce the number of cells by reducing the number of dimensions, or by increasing the granularity of the cells with a generated column.

     You can roll up one or more dimensions to a coarser granularity to give it a lower cardinality. For example, you can continue to cluster the data in the previous example on Region and Product, but replace the dimension of Day with a dimension of YearAndMonth. This gives cardinalities of 60 (12 months times 5 years), 12, and 5 for YearAndMonth, Region, and Product, with a possible number of cells of 3600. Each cell then holds a greater range of values and is less likely to contain only a few records.

     Take into account predicates commonly used on the columns involved, such as whether many are on Month of Date, or Quarter, or Day. This affects the desirability of changing the granularity of the dimension. If, for example, most predicates are on particular days and you have clustered the table on Month, Db2 can use the block index on YearAndMonth to quickly narrow down which months contain the required days and access only those associated blocks. When scanning the blocks, however, the Day predicate must be applied to determine which days qualify. However, if you cluster on Day (and Day has high cardinality), the block index on Day can be used to determine which blocks to scan, and the Day predicate only has to be reapplied to the first record of each cell that qualifies. In this case, it might be better to consider rolling up one of the other dimensions to increase the density of cells, as in rolling up the Region column, which contains 12 different values, to Regions West, North, South and East, using a user-defined function.

Considerations when creating MDC or ITC tables

There are many factors to consider when creating MDC or ITC tables. Decisions on how to create, place, and use your MDC or ITC tables can be influenced by your current database environment (for example, whether you have a partitioned database or not), and by your choice of dimensions.

Moving data from existing tables to MDC tables

To improve query performance and reduce the requirements of data maintenance operations in a data warehouse or large database environment, you can move data from regular tables into multidimensional clustering (MDC) tables. To move data from an existing table to an MDC table:

1. export your data,
2. drop the original table (optional),
3. create a multidimensional clustering (MDC) table (using the CREATE TABLE statement with the ORGANIZE BY DIMENSIONS clause),
4. load the MDC table with your data.

An ALTER TABLE procedure called SYSPROC.ALTOBJ can be used to carry out the translation of data from an existing table to an MDC table. The procedure is called from the Db2 Design Advisor. The time required to translate the data between the tables can be significant and depends on the size of the table and the amount of data that needs to be translated.

The ALTOBJ procedure runs the following steps when altering a table:

1. drop all dependent objects of the table,
2. rename the table,
3. create the table with the new definition,
4. recreate all dependent objects of the table,
5. transform existing data in the table into the data required in the new table. That is, the selecting of data from the old table and loading that data into the new one where column functions can be used to transform from an old data type to a new data type.

Moving data from existing tables to ITC tables

To reduce the requirements of data maintenance operations, you can move data from regular tables into insert time clustering (ITC) tables. To move data from an existing table to an ITC table use the online table move stored procedure.

The ExampleBank scenario shows how data from an existing table is moved into an ITC table. The scenario also shows how convenient reclaiming space is when using ITC tables. For more information, see the Related concepts links.

MDC Advisor feature on the Db2 Design Advisor

The Db2 Design Advisor (**db2advis**) has an MDC feature. This feature recommends clustering dimensions for use in an MDC table, including coarsifications on base columns in order to improve workload performance. The term *coarsification* refers to a mathematical expression to reduce the cardinality (the number of distinct values) of a clustering dimension. A common example is coarsification by date, week of the date, month of the date, or quarter of the year.

A requirement to use the MDC feature of the Db2 Design Advisor is the existence of at least several extents of data within the database. The Db2 Design Advisor uses the data to model data density and cardinality.

If the database does not have data in the tables, the Db2 Design Advisor does not recommend MDC, even if the database contains empty tables but has a mocked up set of statistics to imply a populated database.

The recommendation includes identifying potential generated columns that define coarsification of dimensions. The recommendation does not include possible block sizes. The extent size of the table

space is used when making recommendations for MDC tables. The assumption is that the recommended MDC table is created in the same table space as the existing table, and therefore has the same extent size. The recommendations for MDC dimensions change depending on the extent size of the table space, because the extent size affects the number of records that can fit into a block or cell. The extent size directly affects the density of the cells.

Only single-column dimensions, and not composite-column dimensions, are considered, although single or multiple dimensions might be recommended for the table. The MDC feature recommends coarsifications for most supported data types with the goal of reducing the cardinality of cells in the resulting MDC solution. The data type exceptions include: CHAR, VARCHAR, GRAPHIC, and VARGRAPHIC data types. All supported data types are cast to INTEGER and are coarsified through a generated expression.

The goal of the MDC feature of the Db2 Design Advisor is to select MDC solutions that result in improved performance. A secondary goal is to keep the storage expansion of the database constrained to a modest level. A statistical method is used to determine the maximum storage expansion on each table.

The analysis operation within the advisor includes not only the benefits of block index access but also the effect of MDC on insert, update, and delete operations against dimensions of the table. These actions on the table have the potential to cause records to be moved between cells. The analysis operation also models the potential performance effect of any table expansion resulting from the organization of data along particular MDC dimensions.

The MDC feature is run by using the `-m <advise type>` flag on the db2advis utility. The "C" advise type is used to indicate multidimensional clustering tables. The advise types are: "I" for index, "M" for materialized query tables, "C" for MDC, and "P" for partitioned database environment. The advise types can be used in combination with each other.

Note: The Db2 Design Advisor does not explore tables that are less than 12 extents in size.

The advisor analyzes both MQTs and regular base tables when coming up with recommendations.

The output from the MDC feature includes:

- Generated column expressions for each table for coarsified dimensions that appear in the MDC solution.
- An ORGANIZE BY DIMENSIONS clause recommended for each table.

The recommendations are reported both to stdout and to the ADVISE tables that are part of the explain facility.

MDC tables and partitioned database environments

Multidimensional clustering can be used in a partitioned database environment. In fact, MDC can complement a partitioned database environment. A partitioned database environment is used to distribute data from a table across multiple physical or logical database partitions to:

- take advantage of multiple machines to increase processing requests in parallel,
- increase the physical size of the table beyond the limits of a single database partition,
- improve the scalability of the database.

The reason for distributing a table is independent of whether the table is an MDC table or a regular table. For example, the rules for the selection of columns to make up the distribution key are the same. The distribution key for an MDC table can involve any column, whether those columns make up part of a dimension of the table or not.

If the distribution key is identical to a dimension from the table, then each database partition contains a different portion of the table. For instance, if our example MDC table is distributed by color across two database partitions, then the Color column is used to divide the data. As a result, the Red and Blue slices might be found on one database partition and the Yellow slice on the other. If the distribution key is not identical to the dimensions from the table, then each database partition has a subset of data from each slice. When choosing dimensions and estimating cell occupancy, note that on average the total amount of data per cell is determined by taking all of the data and dividing by the number of database partitions.

MDC tables with multiple dimensions

If you know that certain predicates are heavily used in queries, you can cluster the table on the columns involved. You can do this by using the ORGANIZE BY DIMENSIONS clause.

Example 1:

```
CREATE TABLE T1 (c1 DATE, c2 INT, c3 INT, c4 DOUBLE)
   ORGANIZE BY DIMENSIONS (c1, c3, c4)
```

The table in Example 1 is clustered on the values within three columns forming a logical cube (that is, having three dimensions). The table can now be logically sliced up during query processing on one or more of these dimensions such that only the blocks in the appropriate slices or cells are processed by the relational operators involved. The size of a block (the number of pages) is the extent size of the table.

MDC tables with dimensions based on more than one column

Each dimension can be made up of one or more columns. As an example, you can create a table that is clustered on a dimension containing two columns.

Example 2:

```
CREATE TABLE T1 (c1 DATE, c2 INT, c3 INT, c4 DOUBLE)
   ORGANIZE BY DIMENSIONS (c1, (c3, c4))
```

In Example 2, the table is clustered on two dimensions, c1 and (c3,c4). Thus, in query processing, the table can be logically sliced up on either the c1 dimension, or on the composite (c3, c4) dimension. The table has the same number of blocks as the table in Example 1, but one less dimension block index. In Example 1, there are three dimension block indexes, one for each of the columns c1, c3, and c4. In Example 2, there are two dimension block indexes, one on the column c1 and the other on the columns c3 and c4. The main difference between the two approaches is that, in Example 1, queries involving c4 can use the dimension block index on c4 to quickly and directly access blocks of relevant data. In Example 2, c4 is a second key part in a dimension block index, so queries involving c4 involve more processing. However, in Example 2 there is one less block index to maintain and store.

The Db2 Design Advisor does not make recommendations for dimensions containing more than one column.

MDC tables with column expressions as dimensions

Column expressions can also be used for clustering dimensions. The ability to cluster on column expressions is useful for rolling up dimensions to a coarser granularity, such as rolling up an address to a geographic location or region, or rolling up a date to a week, month, or year. To implement the rolling up of dimensions in this way, you can use generated columns. This type of column definition allows the creation of columns using expressions that can represent dimensions. In Example 3, the statement creates a table clustered on one base column and two column expressions.

Example 3:

```
CREATE TABLE T1(c1 DATE, c2 INT, c3 INT, c4 DOUBLE,
   c5 DOUBLE GENERATED ALWAYS AS (c3 + c4),
   c6 INT GENERATED ALWAYS AS (MONTH(C1)))
      ORGANIZE BY DIMENSIONS (c2, c5, c6)
```

In Example 3, column c5 is an expression based on columns c3 and c4, and column c6 rolls up column c1 to a coarser granularity in time. The statement clusters the table based on the values in columns c2, c5, and c6.

Range queries on generated column dimensions

Range queries on a generated column dimension require monotonic column functions. Expressions must be monotonic to derive range predicates for dimensions on generated columns. If you create a dimension on a generated column, queries on the base column are able to take advantage of the block index on the

generated column to improve performance, with one exception. For range queries on the base column (date, for example) to use a range scan on the dimension block index, the expression used to generate the column in the CREATE TABLE statement must be monotonic. Although a column expression can include any valid expression (including user-defined functions (UDFs)), if the expression is non-monotonic, only equality or IN predicates are able to use the block index to satisfy the query when these predicates are on the base column.

As an example, assume that you create an MDC table with dimensions on the generated column month, where `month = INTEGER (date)/100`. For queries on the dimension (month), block index scans can be done. For queries on the base column (date), block index scans can also be done to narrow down which blocks to scan, and then apply the predicates on date to the rows in those blocks only.

The compiler generates additional predicates to be used in the block index scan. For example, with the query:

```
SELECT * FROM MDCTABLE WHERE DATE > "1999-03-03" AND DATE < "2000-01-15"
```

the compiler generates the additional predicates: "month >= 199903" and "month <= 200001" which can be used as predicates for a dimension block index scan. When scanning the resulting blocks, the original predicates are applied to the rows in the blocks.

A non-monotonic expression allows equality predicates to be applied to that dimension. A good example of a non-monotonic function is MONTH( ) as seen in the definition of column c6 in Example 3. If the c1 column is a date, timestamp, or valid string representation of a date or timestamp, then the function returns an integer value in the range of 1 to 12. Even though the output of the function is deterministic, it actually produces output similar to a step function (that is, a cyclic pattern):

```
MONTH(date('01/05/1999')) = 1
MONTH(date('02/08/1999')) = 2
MONTH(date('03/24/1999')) = 3
MONTH(date('04/30/1999')) = 4
...
MONTH(date('12/09/1999')) = 12
MONTH(date('01/18/2000')) = 1
MONTH(date('02/24/2000')) = 2
...
```

Although date in this example is continually increasing, MONTH(date) is not. More specifically, it is not guaranteed that whenever date1 is larger than date2, MONTH(date1) is greater than or equal to MONTH(date2). It is this condition that is required for monotonicity. This non-monotonicity is allowed, but it limits the dimension in that a range predicate on the base column cannot generate a range predicate on the dimension. However, a range predicate on the expression is fine, for example, `where month(c1) between 4 and 6`. This can use the index on the dimension in the typical way, with a starting key of 4 and a stop key of 6.

To make this function monotonic, include the year as the high-order part of the month. There is an extension to the INTEGER built-in function to help in defining a monotonic expression on date. INTEGER(date) returns an integer representation of the date, which then can be divided to find an integer representation of the year and month. For example, `INTEGER(date('2000/05/24'))` returns `20000524`, and therefore `INTEGER(date('2000/05/24'))/100 = 200005`. The function INTEGER(date)/100 is monotonic.

Similarly, the built-in functions DECIMAL and BIGINT also have extensions so that you can derive monotonic functions. DECIMAL(timestamp) returns a decimal representation of a timestamp, and this can be used in monotonic expressions to derive increasing values for month, day, hour, minute, and so on. BIGINT(date) returns a big integer representation of the date, similar to INTEGER(date).

The database manager determines the monotonicity of an expression, where possible, when creating the generated column for the table, or when creating a dimension from an expression in the dimensions clause. Certain functions can be recognized as monotonicity-preserving, such as DAYS( ) or YEAR( ). Also, various mathematical expressions such as division, multiplication, or addition of a column and a constant are monotonicity-preserving. Where Db2 determines that an expression is not monotonicity-preserving,

or if it cannot determine this, the dimension supports only the use of equality predicates on its base column.

Load considerations for MDC and ITC tables

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

After you delete a set of data, for example, all the data for a month, you can use the load utility to roll in the next month of data and it can reuse the blocks that were emptied after the (committed) deletion. You can also choose to use the MDC rollout feature with deferred cleanup. After the rollout, which is also a deletion, is committed, the blocks are not free and cannot yet be reused. A background process is invoked to maintain the record ID (RID) based indexes. When the maintenance is complete, the blocks are freed and can be reused. For insert time clustering (ITC) tables, blocks that are not in use are reused where possible before the table is extended. This includes blocks that were reclaimed. Rollout is not supported on ITC tables.

When loading data into MDC tables, the input data can be either sorted or unsorted. If unsorted, and the table has more than one dimension, consider doing the following:

- Increase the **util_heap_sz** configuration parameter.

  To improve the performance of the load utility when loading MDC tables, increase the **util_heap_sz** database configuration parameter value. The mdc-load algorithm performs better when more memory is available to the utility. This reduces disk I/O during the clustering of data that is performed during the load phase. If the **LOAD** command is being used to load several MDC tables concurrently, **util_heap_sz** must be increased accordingly.
- Increase the value given with the **DATA BUFFER** clause of the **LOAD** command.

  Increasing this value affects a single load request. The utility heap size must be large enough to accommodate the possibility of multiple concurrent load requests. Beginning in version 9.5, the value of the **DATA BUFFER** parameter of the **LOAD** command can temporarily exceed **util_heap_sz** if more memory is available in the system.
- Ensure the page size used for the buffer pool is the same as the largest page size for the temporary table space.

  During the load phase, extra logging for the maintenance of the block map is performed. There are approximately two extra log records per extent allocated. To ensure good performance, the **logbufsz** database configuration parameter must be set to a value that takes this into account.

The following restrictions apply when loading data into MDC or ITC tables:

- The **SAVECOUNT** parameter in the **LOAD** command is not supported.
- The `totalfreespace` file type modifier is not supported since these tables manage their own free space.
- The `anyorder` file type modifier is required for MDC or ITC tables. If a load is executed into an MDC or ITC table without the `anyorder` modifier, it is explicitly enabled by the utility.

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

- If the table included a unique key before the load operation and duplicate records are loaded into the table, the original record remains and the new records are deleted during the delete phase.
- If the table did not include a unique key prior to the load operation and both a unique key and duplicate records are loaded into the table, only one of the records with the unique key is loaded and the others are deleted during the delete phase.

  Note: There is no explicit technique for determining which record is loaded and which is deleted.

Load begins at a block boundary, so it is best used for data belonging to new cells, for the initial populating of a table, and for loading additional data into ITC tables.

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

Logging considerations for MDC and ITC tables
Index maintenance and logging is reduced when dimensions and therefore block indexes are used, as compared to cases where RID indexes are used.

The database manager removes the BID from the block indexes only when the last record in an entire block is deleted. This index operation is also logged at this time. Similarly, the database manager inserts a BID into the block index only when a record is inserted into a new block. That record must be the first record of a logical cell or an insert to a logical cell of blocks that are currently full. This index operation is also logged at this time.

Because blocks can be 2 - 256 pages of records, this block index maintenance and logging is relatively small. Inserts and deletes to the table and to RID indexes are still logged. For roll out deletions, the deleted records are not logged. Instead, the pages that contain the records are made to look empty by reformatting parts of the pages. The changes to the reformatted parts are logged, but the records themselves are not logged.

Block indexes for MDC and ITC tables
Dimension block indexes are created when you define dimensions for a multidimensional clustering (MDC) table. A composite block index is created when you define multiple dimensions. If you define only one dimension for your MDC table, or if your table is an insert time clustering (ITC) table, the database manager creates only one block index, which serves as both the dimension block index and the composite block index. If your MDC or ITC table is partitioned, the block index is also partitioned.

If you create an MDC table that has dimensions on column A and on (column A, column B), the database manager creates a dimension block index on column A and a dimension block index on (column A, column B). Because a composite block index is a block index of all the dimensions in the table, the dimension block index on (column A, column B) also serves as the composite block index.

The composite block index for an MDC table is used in query processing to access data with specific dimension values. The order of key parts in the composite block index has no effect on insert processing, but might affect its use or applicability for query processing. The order of key parts is determined by the order of columns in the ORGANIZE BY DIMENSIONS clause when the MDC table is created. Multicolumn dimensions in the ORGANIZE BY DIMENSION clause take precedence when there is a duplicate. For example, if a table is created by using the following statement, the composite block index is created on columns (c4, c3, c1, c2).

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

Although c1 is specified twice in the ORGANIZE BY DIMENSIONS clause, it is used only once as a key part for the composite block index; (c3, c1) replaces (c1). The following example shows you how to create a table whose composite block index has a column order of (c1, c2, c3, c4):

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

Block indexes for MDC tables

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

The MDC table shown in Figure 12 on page 45 is physically organized such that records having the same "Region" and "Year" values are grouped together into separate blocks, or extents. An extent is a set of contiguous pages on disk, so these groups of records are clustered on physically contiguous data pages. Each table page belongs to exactly one block, and all blocks are of equal size (that is, an equal number of pages). The size of a block is equal to the extent size of the table space, so that block boundaries line up with extent boundaries. In this case, two block indexes are created, one for the "Region" dimension, and another for the "Year" dimension. These block indexes contain pointers only to the blocks in the table. A scan of the "Region" block index for all records having "Region" equal to "East" will find two blocks that qualify. All records, and only those records, having "Region" equal to "East" will be found in these two blocks, and will be clustered on those two sets of contiguous pages or extents. At the same time, and completely independently, a scan of the "Year" index for records between 1999 and 2000 will find three blocks that qualify. A data scan of each of these three blocks will return all records and only those records

that are between 1999 and 2000, and will find these records clustered on the sequential pages within each of the blocks.

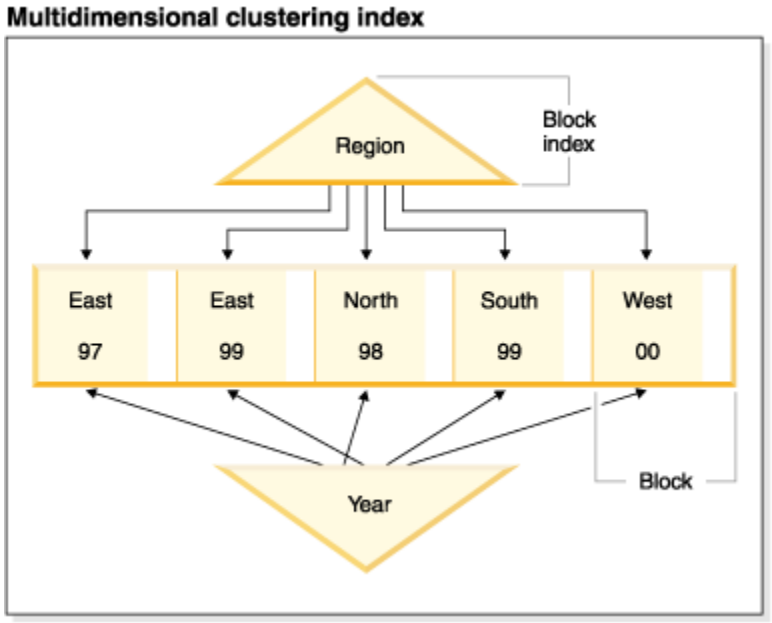


Figure 12. A multidimensional clustering table

In addition to these clustering improvements, MDC tables provide the following benefits:

- Probes and scans of block indexes are much faster due to their incredibly small size in relation to record-based indexes
- Block indexes and the corresponding organization of data allows for fine-grained "database partition elimination", or selective table access
- Queries that utilize the block indexes benefit from the reduced index size, optimized prefetching of blocks, and guaranteed clustering of the corresponding data
- Reduced locking and predicate evaluation is possible for some queries
- Block indexes have much less overhead associated with them for logging and maintenance because they only need to be updated when adding the first record to a block, or removing the last record from a block
- Data rolled in can reuse the contiguous space left by data previously rolled out.

Note: An MDC table defined with even just a single dimension can benefit from these MDC attributes, and can be a viable alternative to a regular table with a clustering index. This decision should be based on many factors, including the queries that make up the workload, and the nature and distribution of the data in the table. Refer to "Choosing MDC table dimensions" on page 32 and "Considerations when creating MDC or ITC tables" on page 39.

When you create a table, you can specify one or more keys as dimensions along which to cluster the data. Each of these MDC dimensions can consist of one or more columns similar to regular index keys. A *dimension block index* will be automatically created for each of the dimensions specified, and it will be used by the optimizer to quickly and efficiently access data along each dimension. A *composite block index* will also automatically be created, containing all columns across all dimensions, and will be used to maintain the clustering of data over insert and update activity. A composite block index will only be created if a single dimension does not already contain all the dimension key columns. The composite block index may also be selected by the optimizer to efficiently access data that satisfies values from a subset, or from all, of the column dimensions.

Note: The usefulness of this index during query processing depends on the order of its key parts. The key part order is determined by the order of the columns encountered by the parser when parsing the

dimensions specified in the ORGANIZE BY DIMENSIONS clause of the CREATE TABLE statement. Refer to “Block indexes for MDC and ITC tables” on page 44 for more information.

Block indexes are structurally the same as regular indexes, except that they point to blocks instead of records. Block indexes are smaller than regular indexes by a factor of the block size multiplied by the average number of records on a page. The number of pages in a block is equal to the extent size of the table space, which can range from 2 to 256 pages. The page size can be 4 KB, 8 KB, 16 KB, or 32 KB.

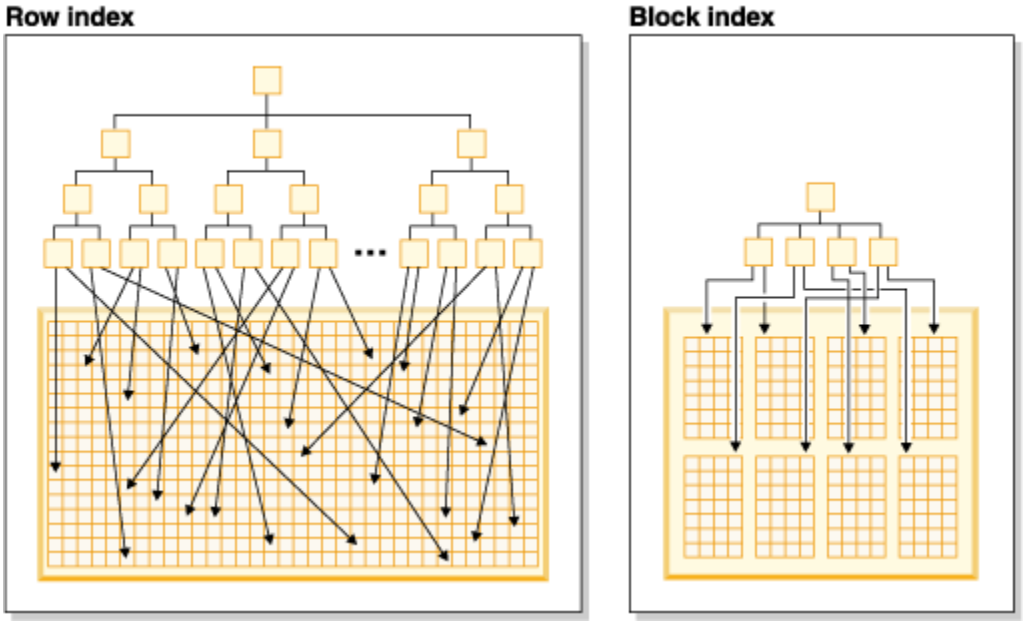


Figure 13. How row indexes differ from block indexes

As seen in Figure 13 on page 46, in a block index there is a single index entry for each block compared to a single entry for each row. As a result, a block index provides a significant reduction in disk usage and significantly faster data access.

In an MDC table, every unique combination of dimension values form a logical *cell*, which may be physically made up of one or more blocks of pages. The logical cell will only have enough blocks associated with it to store the records having the dimension values of that logical cell. If there are no records in the table having the dimension values of a particular logical cell, no blocks will be allocated for that logical cell. The set of blocks that contain data having a particular dimension key value is called a *slice*.

An MDC table can be partitioned. The block index on a partitioned MDC table can be either nonpartitioned or partitioned:

- For a partitioned MDC table created with Db2 Version 9.7 Fix Pack 1 or later releases, the block indexes on the table are partitioned.
- For a partitioned MDC table created with Db2 V9.7 or earlier releases, the block indexes on the table are nonpartitioned.

Nonpartitioned block index are supported after upgrading the database to Db2 V9.7 Fix Pack 1 or later releases.

Scenario: Multidimensional clustered (MDC) tables

As a scenario of how to work with an MDC table, we will imagine an MDC table called "Sales" that records sales data for a national retailer. The table is clustered along the dimensions "YearAndMonth" and "Region". Records in the table are stored in blocks, which contain enough consecutive pages on disk to fill an extent.

In Figure 14 on page 47, a block is represented by a rectangle, and is numbered according to the logical order of allocated extents in the table. The grid in the diagram represents the logical database partitioning

of these blocks, and each square represents a logical cell. A column or row in the grid represents a slice for a particular dimension. For example, all records containing the value 'South-central' in the "Region" column are found in the blocks contained in the slice defined by the 'South-central' column in the grid. In fact, each block in this slice also only contains records having 'South-central' in the "Region" field. Thus, a block is contained in this slice or column of the grid if and only if it contains records having 'South-central' in the "Region" field.

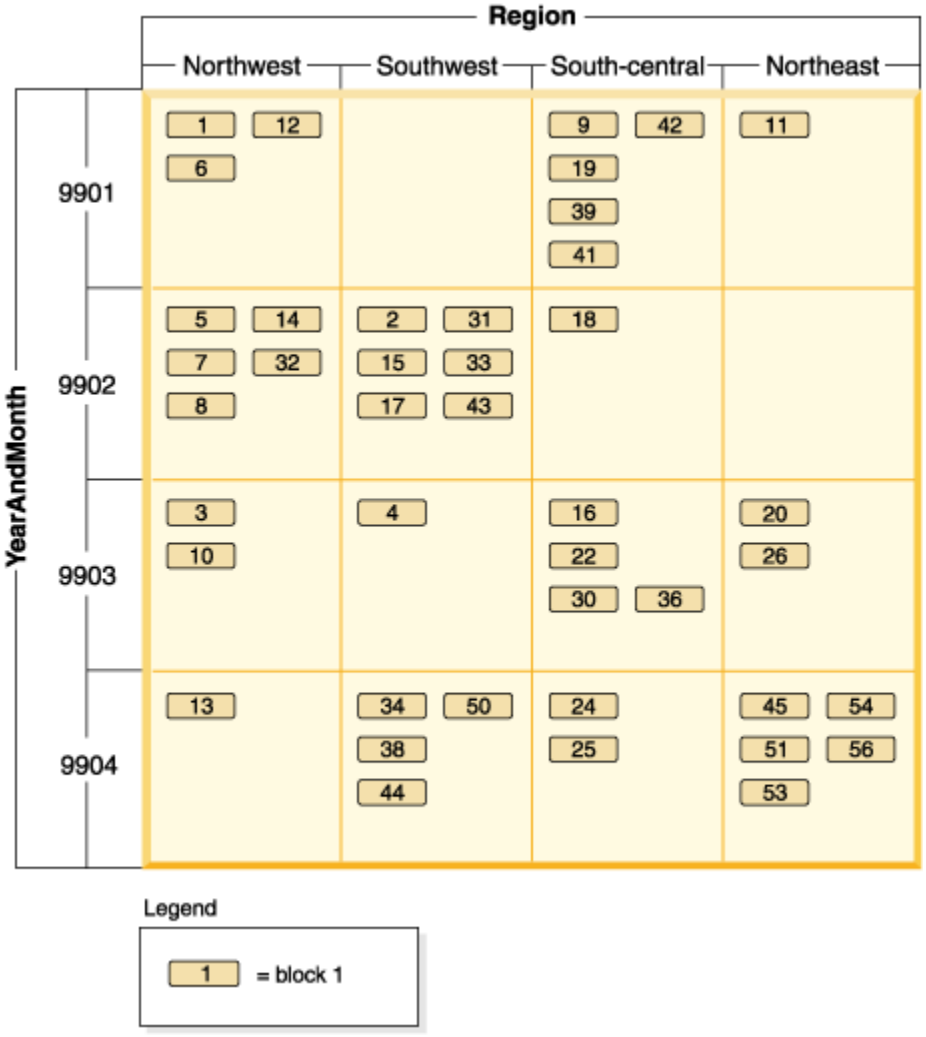


Figure 14. Multidimensional table with dimensions of 'Region' and 'YearAndMonth' that is called Sales

To determine which blocks comprise a slice, or equivalently, which blocks contain all records having a particular dimension key value, a dimension block index is automatically created for each dimension when the table is created.

In Figure 15 on page 48, a dimension block index is created on the "YearAndMonth" dimension, and another on the "Region" dimension. Each dimension block index is structured in the same manner as a traditional RID index, except that at the leaf level the keys point to a block identifier (BID) instead of a record identifier (RID). A RID identifies the location of a record in the table by a physical page number and a slot number - the slot on the page where the record is found. A BID represents a block by the physical page number of the first page of that extent, and a dummy slot (0). Because all pages in the block are physically consecutive starting from that one, and we know the size of the block, all records in the block can be found using this BID.

A slice, or the set of blocks containing pages with all records having a particular key value in a dimension, will be represented in the associated dimension block index by a BID list for that key value.

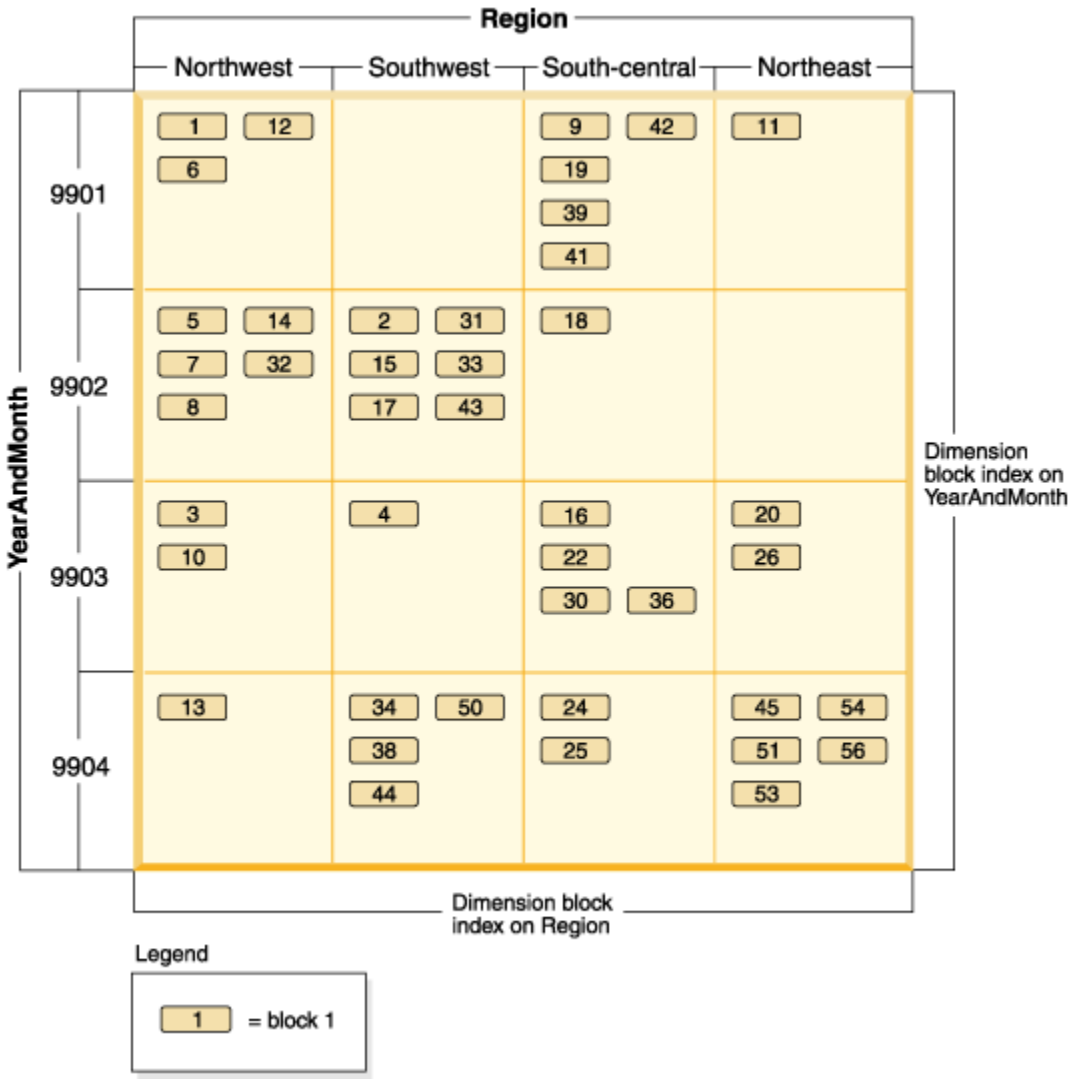


Figure 15. Sales table with dimensions of 'Region' and 'YearAndMonth' showing dimension block indexes

Figure 16 on page 48 shows how a key from the dimension block index on "Region" would appear. The key is made up of a key value, namely 'South-central', and a list of BIDs. Each BID contains a block location. In Figure 16 on page 48, the block numbers listed are the same that are found in the 'South-central' slice found in the grid for the Sales table (see Figure 14 on page 47).

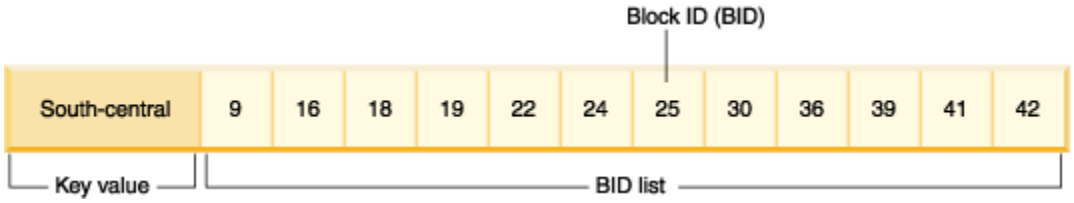


Figure 16. Key from the dimension block index on 'Region'

Similarly, to find the list of blocks containing all records having '9902' for the "YearAndMonth" dimension, look up this value in the "YearAndMonth" dimension block index, shown in Figure 17 on page 49.

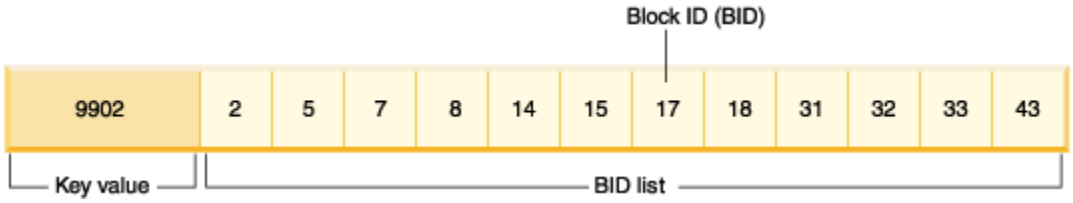


Figure 17. Key from the dimension block index on 'YearAndMonth'

Block indexes and query performance for MDC tables

Scans on any of the block indexes of an MDC table provide clustered data access, because each block identifier (BID) corresponds to a set of sequential pages in the table that is guaranteed to contain data having the specified dimension value. Moreover, dimensions or slices can be accessed independently from each other through their block indexes without compromising the cluster factor of any other dimension or slice. This provides the multidimensionality of multidimensional clustering.

Queries that take advantage of block index access can benefit from a number of factors that improve performance.

- Because block indexes are so much smaller than regular indexes, a block index scan is very efficient.
- Prefetching of data pages does not rely on sequential detection when block indexes are used. The Db2 database manager looks ahead in the index, prefetching blocks of data into memory using big-block I/O, and ensuring that the scan does not incur I/O costs when data pages are accessed in the table.
- The data in the table is clustered on sequential pages, optimizing I/O and localizing the result set to a selected portion of the table.
- If a block-based buffer pool is used, and the block size is equal to the extent size, MDC blocks are prefetched from sequential pages on disk into sequential pages in memory, further increasing the positive effect of clustering on performance.
- The records from each block are retrieved using a mini-relational scan of its data pages, which is often faster than scanning data through RID-based retrieval.

Queries can use block indexes to narrow down a portion of the table having a particular dimension value or range of values. This provides a fine-grained form of "database partition elimination", that is, block elimination. This can translate into better concurrency for the table, because other queries, loads, inserts, updates and deletes may access other blocks in the table without interacting with this query's data set.

If the Sales table is clustered on three dimensions, the individual dimension block indexes can also be used to find the set of blocks containing records which satisfy a query on a subset of all of the dimensions of the table. If the table has dimensions of "YearAndMonth", "Region" and "Product", this can be thought of as a logical cube, as illustrated in Figure 18 on page 50.

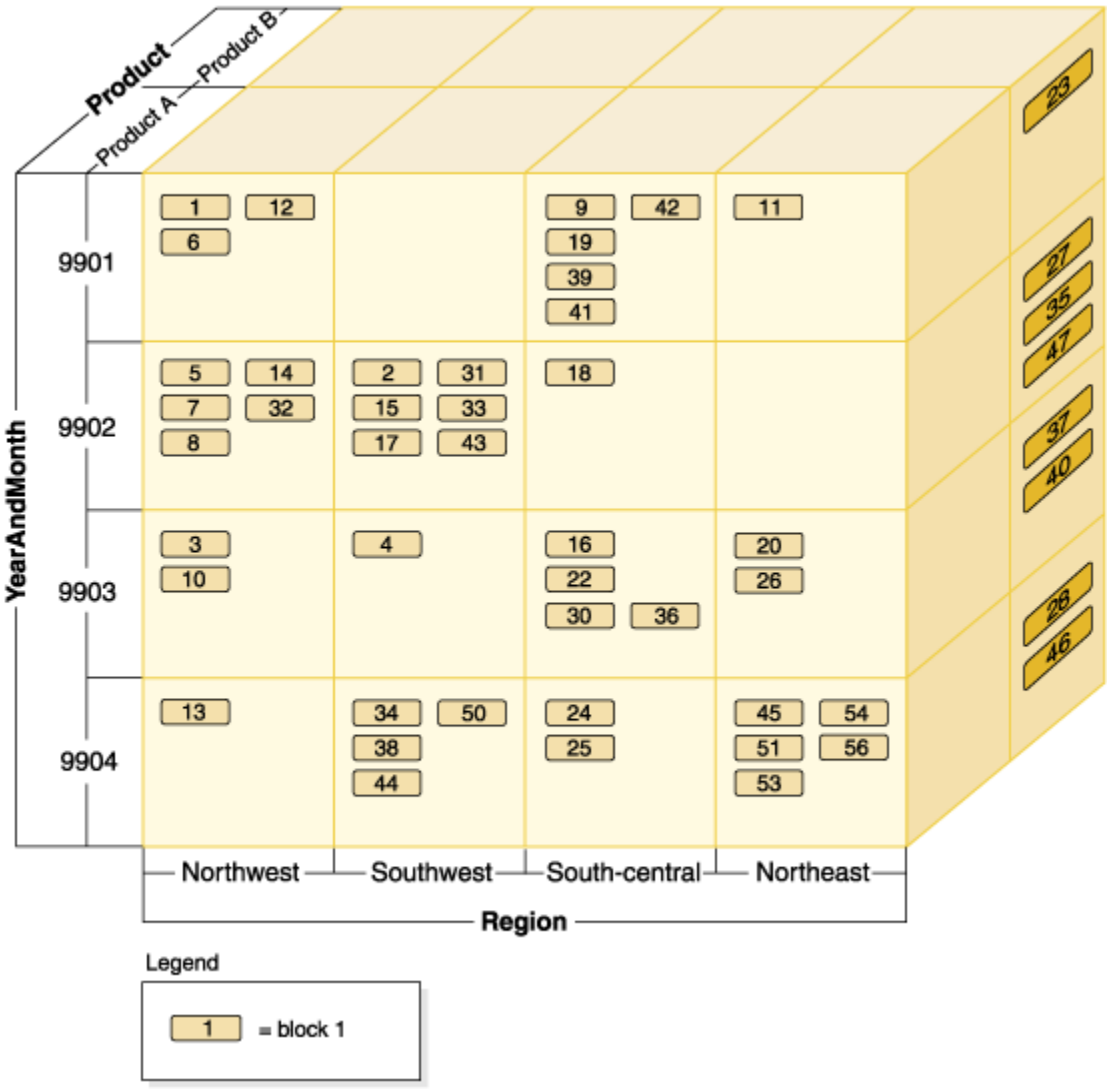


Figure 18. Multidimensional table with dimensions of 'Region', 'YearAndMonth', and 'Product'

Four block indexes will be created for the MDC table shown in Figure 18 on page 50: one for each of the individual dimensions, "YearAndMonth", "Region", and "Product"; and another with all of these dimension columns as its key. To retrieve all records having a "Product" equal to "ProductA" and "Region" equal to "Northeast", the database manager would first search for the ProductA key from the "Product" dimension block index. (See Figure 19 on page 50.) The database manager then determines the blocks containing all records having "Region" equal to "Northeast", by looking up the "Northeast" key in the "Region" dimension block index. (See Figure 20 on page 50.)

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

Figure 19. Key from dimension block index on 'Product'

Northeast	11	20	23	26	27	28	35	37	40	45	46	47	51	53	54	56

Figure 20. Key from dimension block index on 'Region'

Block index scans can be combined through the use of the logical AND and logical OR operators and the resulting list of blocks to scan also provides clustered data access.

Using the previous example, in order to find the set of blocks containing all records having both dimension values, you have to find the intersection of the two slices. This is done by using the logical AND operation on the BID lists from the two block index keys. The common BID values are 11, 20, 26, 45, 54, 51, 53, and 56.

The following example illustrates how to use the logical OR operation with block indexes to satisfy a query having predicates that involve two dimensions. Figure 21 on page 51 assumes an MDC table where the two dimensions are "Colour" and "Nation". The goal is to retrieve all those records in the MDC table that meet the conditions of having "Colour" of "blue" or having a "Nation" name "USA".

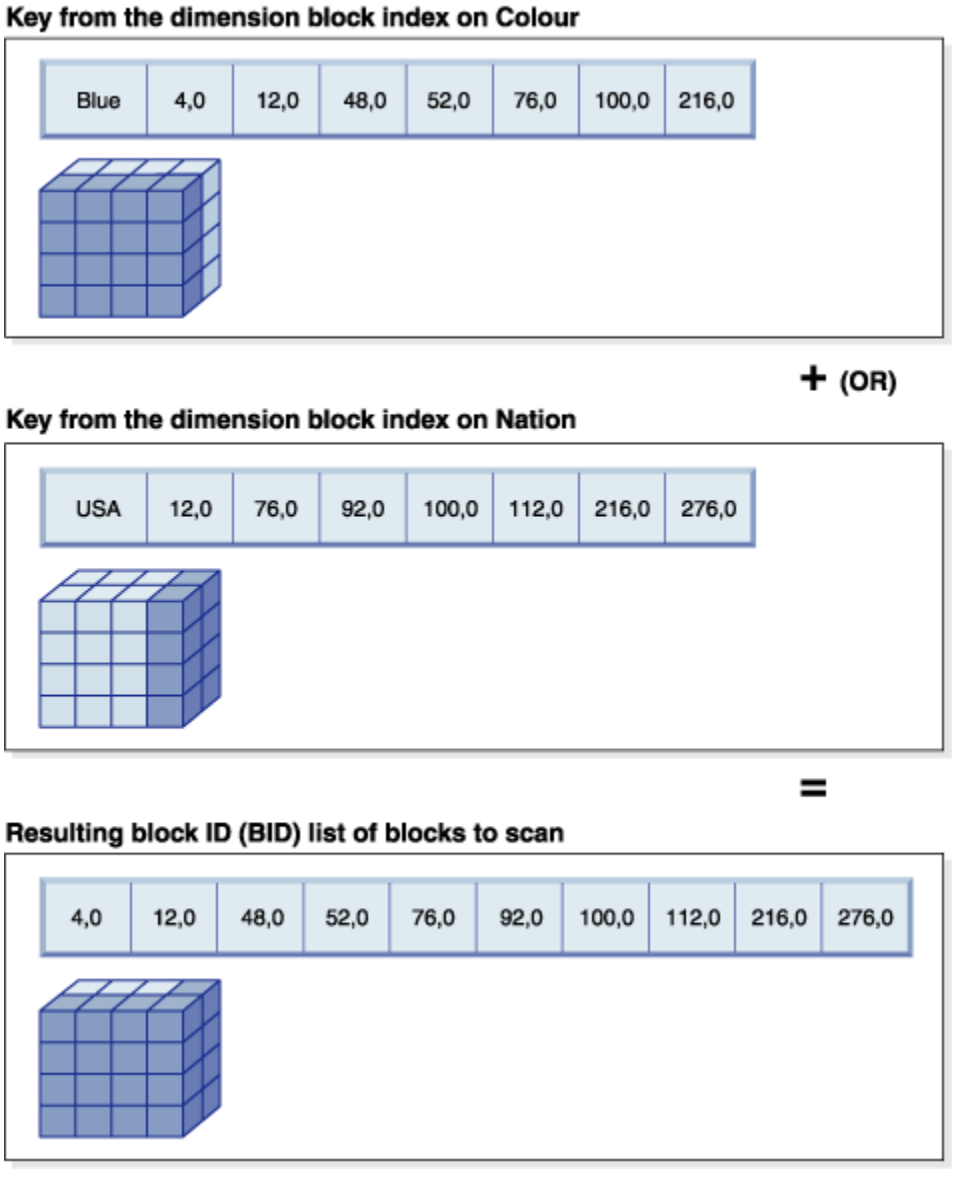


Figure 21. How the logical OR operation can be used with block indexes

This diagram shows how the result of two separate block index scans are combined to determine the range of values that meet the predicate restrictions. (The numbers indicate record identifiers (RIDs), slot fields.)

Based on the predicates from the SELECT statement, two separate dimension block index scans are done; one for the blue slice, and another for the USA slice. A logical OR operation is done in memory in order to find the union of the two slices, and determine the combined set of blocks found in both slices (including the removal of duplicate blocks).

Once the database manager has list of blocks to scan, the database manager can do a mini-relational scan of each block. Prefetching of the blocks can be done, and will involve just one I/O per block, as each block is stored as an extent on disk and can be read into the buffer pool as a unit. If predicates need to be applied to the data, dimension predicates need only be applied to one record in the block, because all records in the block are guaranteed to have the same dimension key values. If other predicates are present, the database manager only needs to check these on the remaining records in the block.

MDC tables also support regular RID-based indexes. Both RID and block indexes can be combined using a logical AND operation, or a logical OR operation, with the index. Block indexes provide the optimizer with additional access plans to choose from, and do not prevent the use of traditional access plans (RID scans, joins, table scans, and others). Block index plans will be costed by the optimizer along with all other possible access plans for a particular query, and the most inexpensive plan will be chosen.

The Db2 Design Advisor can help to recommend RID-based indexes on MDC tables, or to recommend MDC dimensions for a table.

Maintaining clustering automatically during INSERT operations

Automatic maintenance of data clustering in an MDC table is ensured using the composite block index. It is used to dynamically manage and maintain the physical clustering of data along the dimensions of the table over the course of INSERT operations.

A key is found in this composite block index only for each of those logical cells of the table that contain records. This block index is therefore used during an INSERT to quickly and efficiently determine if a logical cell exists in the table, and only if so, determine exactly which blocks contain records having that cell's particular set of dimension values.

When an insert occurs:

- The composite block index is probed for the logical cell corresponding to the dimension values of the record to be inserted.
- If the key of the logical cell is found in the index, its list of block ID (BIDs) gives the complete list of blocks in the table having the dimension values of the logical cell. (See Figure 22 on page 52.) This limits the numbers of extents of the table to search for space to insert the record.
- If the key of the logical cell is not found in the index; or, if the extents containing these values are full, a new block is assigned to the logical cell. If possible, the reuse of an empty block in the table occurs first before extending the table by another new extent of pages (a new block).

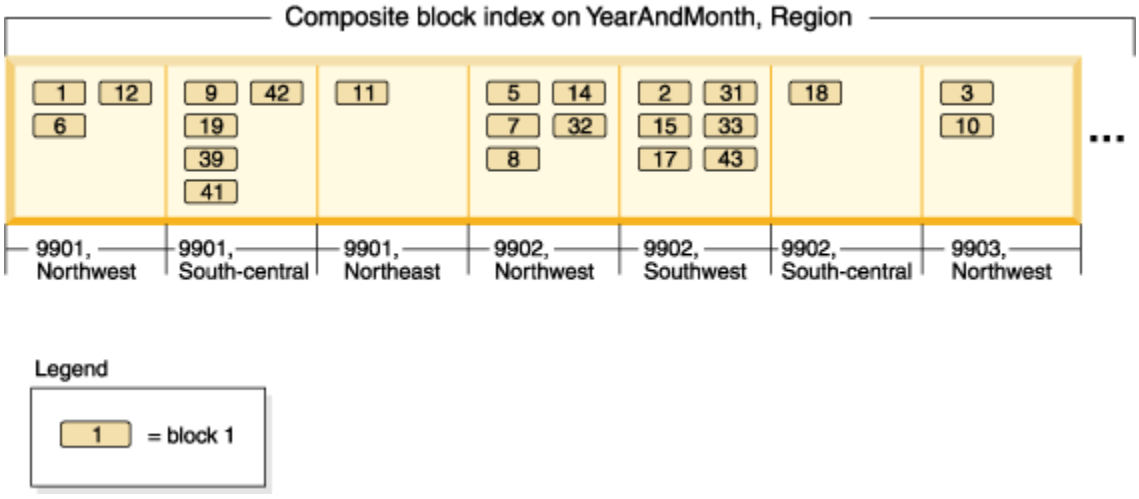


Figure 22. Composite block index on 'YearAndMonth', 'Region'

Data records having particular dimension values are guaranteed to be found in a set of blocks that contain only and all the records having those values. Blocks are made up of consecutive pages on disk. As a result, access to these records is sequential, providing clustering. This clustering is automatically maintained over time by ensuring that records are only inserted into blocks from cells with the record's

dimension values. When existing blocks in a logical cell are full, an empty block is reused or a new block is allocated and added to the set of blocks for that logical cell. When a block is emptied of data records, the block ID (BID) is removed from the block indexes. This disassociates the block from any logical cell values so that it can be reused by another logical cell in the future. Thus, cells and their associated block index entries are dynamically added and removed from the table as needed to accommodate only the data that exists in the table. The composite block index is used to manage this, because it maps logical cell values to the blocks containing records having those values.

Because clustering is automatically maintained in this way, reorganization of an MDC table is never needed to re-cluster data. However, reorganization can still be used to reclaim space. For example, if cells have many sparse blocks where data could fit on fewer blocks, or if the table has many pointer-overflow pairs, a reorganization of the table would compact records belonging to each logical cell into the minimum number of blocks needed, as well as remove pointer-overflow pairs.

The following example illustrates how the composite block index can be used for query processing. If you want to find all records in the table in Figure 22 on page 52 having "Region" of 'Northwest' and "YearAndMonth" of '9903', the database manager would look up the key value 9903, Northwest in the composite block index, as shown in Figure 23 on page 53. The key is made up a key value, namely '9903, Northwest', and a list of BIDs. You can see that the only BIDs listed are 3 and 10, and indeed there are only two blocks in the Sales table containing records having these two particular values.

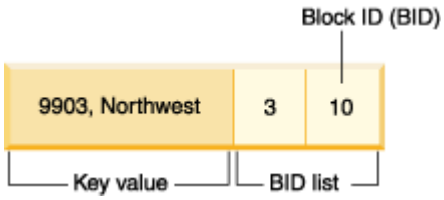


Figure 23. Key from composite block index on 'YearAndMonth', 'Region'

To illustrate the use of the composite block index during insert, take the example of inserting another record with dimension values 9903 and Northwest. The database manager would look up this key value in the composite block index and find BIDs for blocks 3 and 10. These blocks contain all records and the only records having these dimension key values. If there is space available, the database manager inserts the new record into one of these blocks. If there is no space on any pages in these blocks, the database manager allocates a new block for the table, or uses a previously emptied block in the table. Note that, in this example, block 48 is currently not in use by the table. The database manager inserts the record into the block and associates this block to the current logical cell by adding the BID of the block to the composite block index and to each dimension block index. See Figure 24 on page 53 for an illustration of the keys of the dimension block indexes after the addition of Block 48.

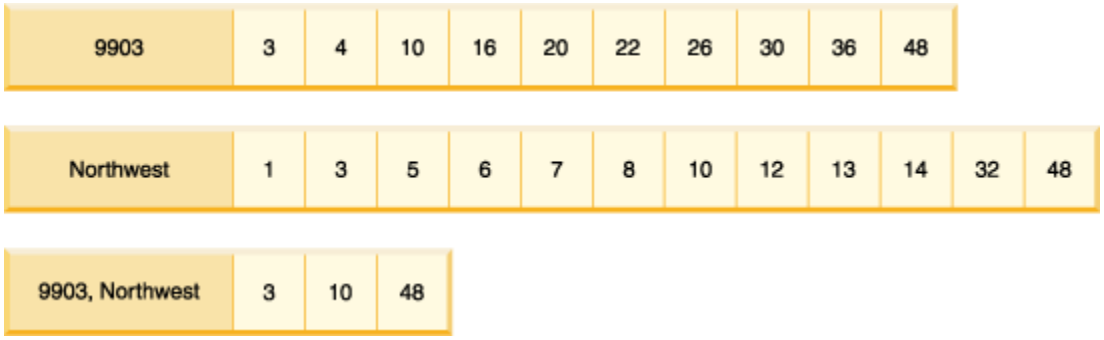


Figure 24. Keys from the dimension block indexes after addition of Block 48

Block maps for MDC and ITC tables

For MDC tables, when a block is emptied, it is disassociated from its current logical cell values by removing its BID from the block indexes. The block can then be reused by another logical cell. For ITC tables, all blocks are associated with a single cell. Freeing a block within a cell means it can be reused by a subsequent insert. This reuse reduces the need to extend the table with new blocks.

When a new block is needed, previously emptied blocks need to be found quickly without having to search the table for them.

The block map is a structure used to facilitate locating empty blocks in the MDC or ITC table. The block map is stored as a separate object:

- In SMS, as a separate .BKM file
- In DMS, as a new object descriptor in the object table.

The block map is an array containing an entry for each block of the table. Each entry comprises a set of status bits for a block.

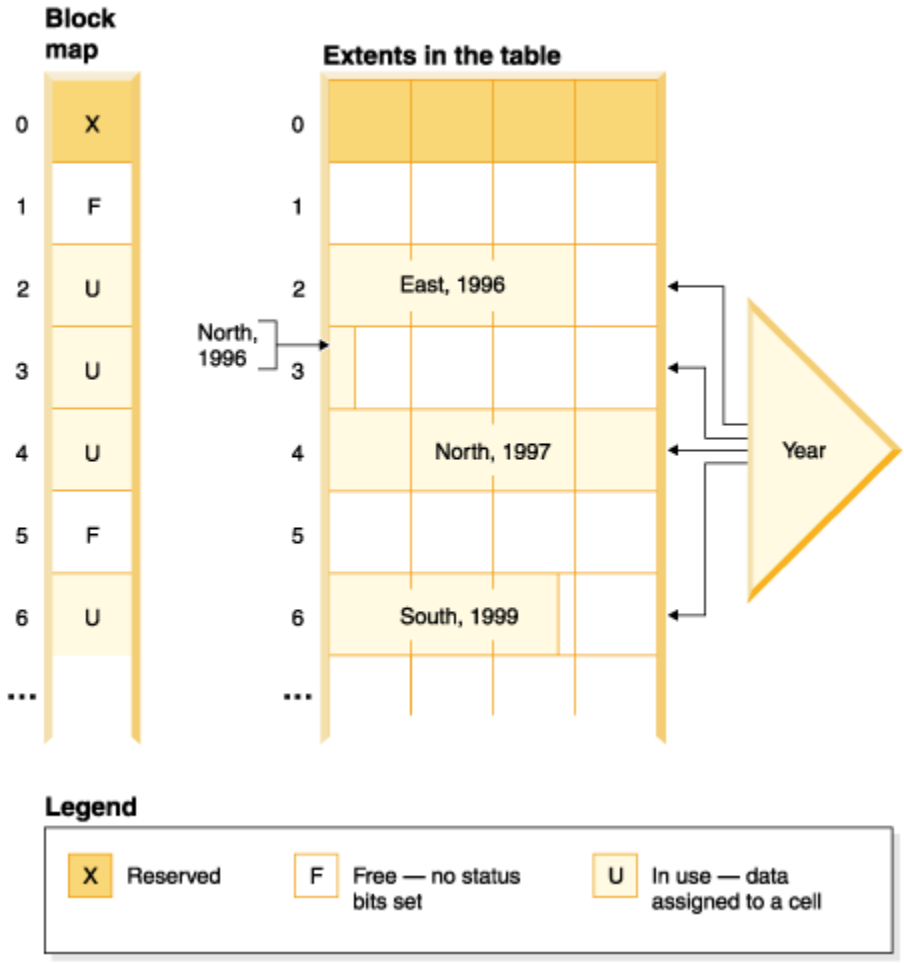


Figure 25. How a block map works

In Figure 25 on page 54, the left side shows the block map array with different entries for each block in the table. The right side shows how each extent of the table is being used: some are free, most are in use, and records are only found in blocks marked in use in the block map. For simplicity, only one of the two dimension block indexes is shown in the diagram.

Note:

1. There are pointers in the block index only to blocks which are marked IN USE in the block map.
2. The first block is reserved. This block contains system records for the table.

Free blocks are found easily for use in a cell, by scanning the block map for FREE blocks, that is, blocks without any bits set.

Table scans also use the block map to access only extents currently containing data. Any extents not in use do not need to be included in the table scan at all. To illustrate, a table scan in this example (Figure 25 on page 54) would start from the third extent (extent 2) in the table, skipping the first reserved extent

and the subsequent empty extent, scan blocks 2, 3 and 4 in the table, skip the next extent (not touching the data pages of that extent), and then continue scanning from there.

Deleting from MDC and ITC tables

When a record is deleted in an MDC or ITC table, if it is not the last record in the block, the database manager merely deletes the record and removes its RID from any record-based indexes defined on the table.

When a delete removes the last record in a block, the database manager frees the block. The block is freed by changing the IN_USE status bit and removing the BID of the block from all block indexes. If there are record-based indexes as well, the RID is removed from them.

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

Updates to MDC and ITC tables

In an MDC table, updates of non-dimension values are done in place just as they are done with regular tables. If the update of a record in an MDC or ITC table causes the record to grow in length and it no longer fits on the page, another page with sufficient space is found.

The search for this new page begins within the same block. If there is no space in that block, the algorithm to insert a new record is used to find a page in the logical cell with enough space. There is no need to update the block indexes, unless no space is found in the cell and a new block needs to be added to the cell.

For an ITC table, if there is insufficient room in the block to place the updated row, the row is moved to a new block. This move causes the row to no longer be clustered with rows that were inserted at a similar time.

Considerations for MDC tables only

Updates of dimension values are treated as a delete of the current record followed by an insert of the changed record, because the record is changing the logical cell to which it belongs. If the deletion of the current record causes a block to be emptied, the block index needs to be updated. Similarly, if the insert of the new record requires it to be inserted into a new block, the block index needs to be updated.

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

Considerations for MDC and ITC tables

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

When you are reusing blocks that were recently made empty, a conditional Z lock on the block must be used to ensure that it is not currently being scanned by a UR scanner.

Multidimensional and insert time clustering extent management

Freeing data extents from within the multidimensional (MDC) or insert time clustering (ITC) table is done through the reorganization of the table.

Within an MDC and ITC table, a block map tracks all the data extents belonging to a table and indicates which blocks and extents have data on them and which do not. Blocks with data are marked as being "in use". Whenever deletions on MDC or ITC tables, or rollouts on MDC tables happen, block entries with the block map are no longer marked "in use" but rather are freed for reuse by the table.

However, these blocks and extents cannot be used by other objects within the table space. You can release these free data extents from the table through the reorganization of the table. You can use the **REORG TABLE** command with the **RECLAIM EXTENTS** parameter so the table is available and online to

your users while space is reclaimed. The freeing of extents from the MDC or ITC table is only supported for tables in DMS table spaces.

The **REORG TABLE** command uses the **RECLAIM EXTENTS** parameter to free extents from exclusive use by the MDC or ITC table and makes the space available for use by other database objects within the table space.

The option also allows for your control of concurrent access to the MDC or ITC table while the extents are being freed. Write access is the default, read access and no access are also choices to control concurrent access.

If the MDC or ITC table is also range or database partitioned, by default the freeing of extents occurs on all data or database partitions. You can run the command to free extents only on a specific partition by specifying a partition name (for data partitions) or a partition number (for database partitions).

Both the **REORG TABLE** command and the db2Reorg API can be used to free extents.

Automatic support is available to make the freeing of extents part of your automatic maintenance activities for the database. To enable a reorganization to free extents in an MDC or ITC table, the **auto_maint**, **auto_tbl_maint**, and **auto_reorg** database configuration parameters must all have a value of ON. The configuring of these database configuration parameters can be carried out using the command line. On a Db2 instance where the database partitioning feature is enabled, the configuring of the parameters must be issued on the catalog partition.

A maintenance policy controls when an automatic reorganization of an MDC or ITC table takes place to free unused extents. The Db2 system stored procedures AUTOMAINT_SET_POLICY and AUTOMAINT_SET_POLICYFILE are used to set this maintenance policy. XML is used to store the automated maintenance policy.

Table partitioning and multidimensional clustering tables

In a table that is both multidimensional clustered and data partitioned, columns can be used both in the table partitioning range-partition-spec and in the multidimensional clustering (MDC) key. A table that is both multidimensional clustered and partitioned can achieve a finer granularity of data partition and block elimination than could be achieved by either functionality alone.

There are also many applications where it is useful to specify different columns for the MDC key than those on which the table is partitioned. It should be noted that table partitioning is multicolumn, while MDC is multi-dimension.

Characteristics of a mainstream Db2 data warehouse

The following recommendations were focused on typical, mainstream warehouses that were new for Db2 V9.1. The following characteristics are assumed:

- The database runs on multiple machines or multiple AIX logical partitions.
- Partitioned database environments are used (tables are created using the DISTRIBUTE BY HASH clause).
- There are four to fifty data partitions.
- The table for which MDC and table partitioning is being considered is a major fact table.
- The table has 100,000,000 to 100,000,000,000 rows.
- New data is loaded at various time frames: nightly, weekly, monthly.
- Daily ingest volume is 10 thousand to 10 million records.
- Data volumes vary: The biggest month is 5X the size of the smallest month. Likewise, the biggest dimensions (product line, region) have a 5X size range.
- 1 to 5 years of detailed data is retained.
- Expired data is rolled out monthly or quarterly.
- Tables use a wide range of query types. However, the workload is mostly analytical queries with the following characteristics, relative to OLTP workloads:

  - larger results sets with up to 2 million rows
  - most or all queries are hitting views, not base tables
- SQL clauses selecting data by ranges (BETWEEN clause), items in lists, and so on.

Characteristics of a mainstream Db2 V9.1 data warehouse fact table

A typical warehouse fact table, might use the following design:

- Create data partitions on the Month column.
- Define a data partition for each period you roll-out, for example, 1 month, 3 months.
- Create MDC dimensions on Day and on 1 to 4 additional dimensions. Typical dimensions are: product line and region.
- All data partitions and MDC clusters are spread across all database partitions.

MDC and table partitioning provide overlapping sets of benefits. The following table lists potential needs in your organization and identifies a recommended organization scheme based on the characteristics identified previously.

Table 7. Using table partitioning with MDC tables

Issue	Recommended scheme	Recommendation
Data availability during roll-out	Table partitioning	Use the DETACH PARTITION clause to roll out large amounts of data with minimal disruption.
Query performance	Table partitioning and MDC	MDC is best for querying multiple dimensions. Table partitioning helps through data partition elimination.
Minimal reorganization	MDC	MDC maintains clustering, which reduces the need to reorganize.
Rollout a month or more of data during a traditional offline window	Table partitioning	Data partitioning addresses this need fully. MDC adds nothing and would be less suitable on its own.
Rollout a month or more of data during a micro-offline window (less than 1 minute)	Table partitioning	Data partitioning addresses this need fully. MDC adds nothing and would be less suitable on its own.
Rollout a month or more of data while keeping the table fully available for business users submitting queries without any loss of service.	MDC	MDC only addresses this need somewhat. Table partitioning would not be suitable due to the short period the table goes offline.
Load data daily (**LOAD** or **INGEST** command)	Table partitioning and MDC	MDC provides most of the benefit here. Table partitioning provides incremental benefits.
Load data "continually" (**LOAD** command with ALLOW READ ACCESS or **INGEST** command)	Table partitioning and MDC	MDC provides most of the benefit here. Table partitioning provides incremental benefits.
Query execution performance for "traditional BI" queries	Table partitioning and MDC	MDC is especially good for querying cubes/multiple dimensions. Table partitioning helps via partition elimination.

Table 7. Using table partitioning with MDC tables (continued)		
Issue	**Recommended scheme**	**Recommendation**
Minimize reorganization pain, by avoiding the need for reorganization or reducing the pain associated with performing the task	MDC	MDC maintains clustering which reduces the need to reorg. If MDC is used, data partitioning does not provide incremental benefits. However if MDC is not used, table partitioning helps reduce the need for reorg by maintaining some course grain clustering at the partition level.

Example 1:

Consider a table with key columns YearAndMonth and Province. A reasonable approach to planning this table might be to partition by date with 2 months per data partition. In addition, you might also organize by Province, so that all rows for a particular province within any two month date range are clustered together, as shown in Figure 26 on page 59.

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

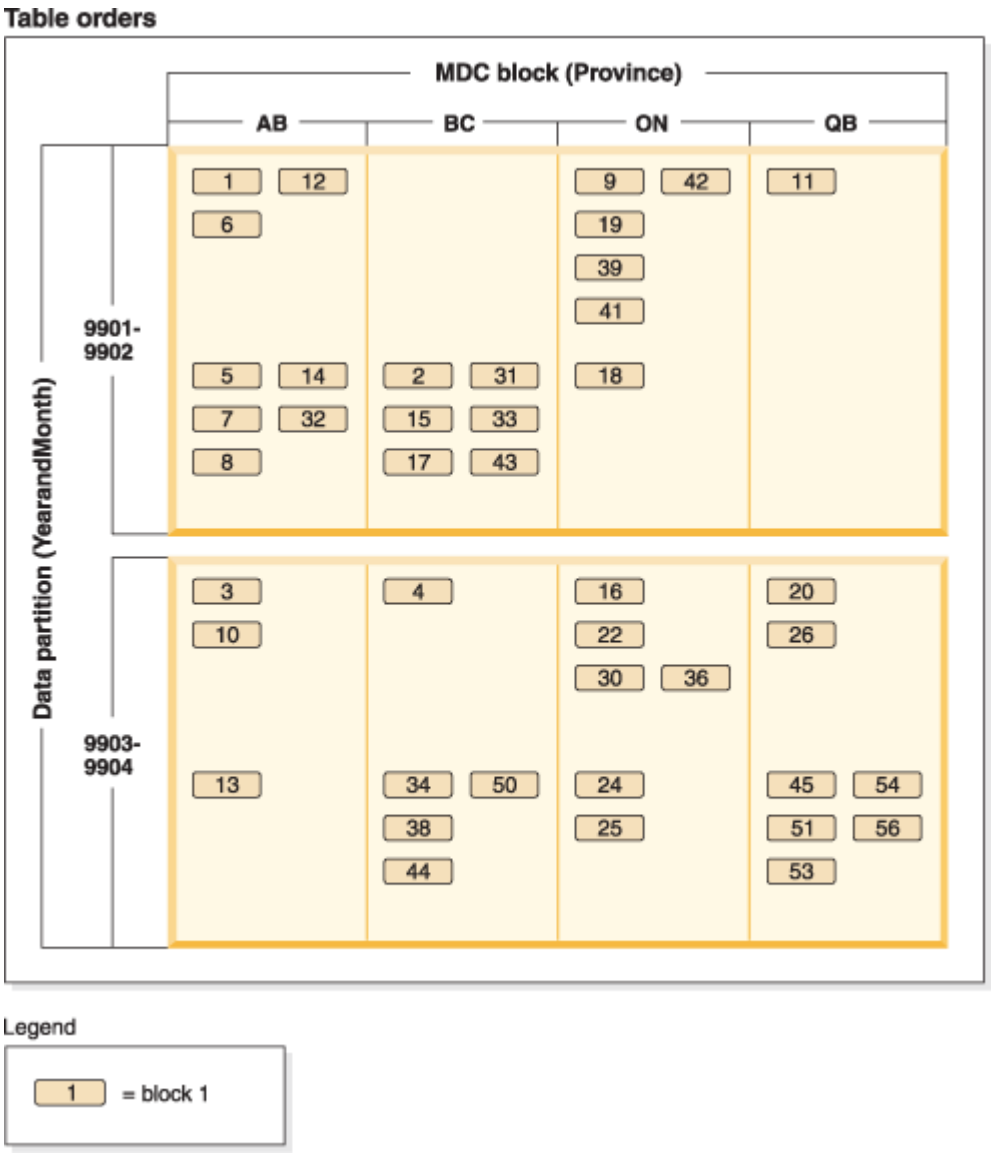


Figure 26. A table partitioned by YearAndMonth and organized by Province

Example 2:

Finer granularity can be achieved by adding YearAndMonth to the ORGANIZE BY DIMENSIONS clause, as shown in Figure 27 on page 60.

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

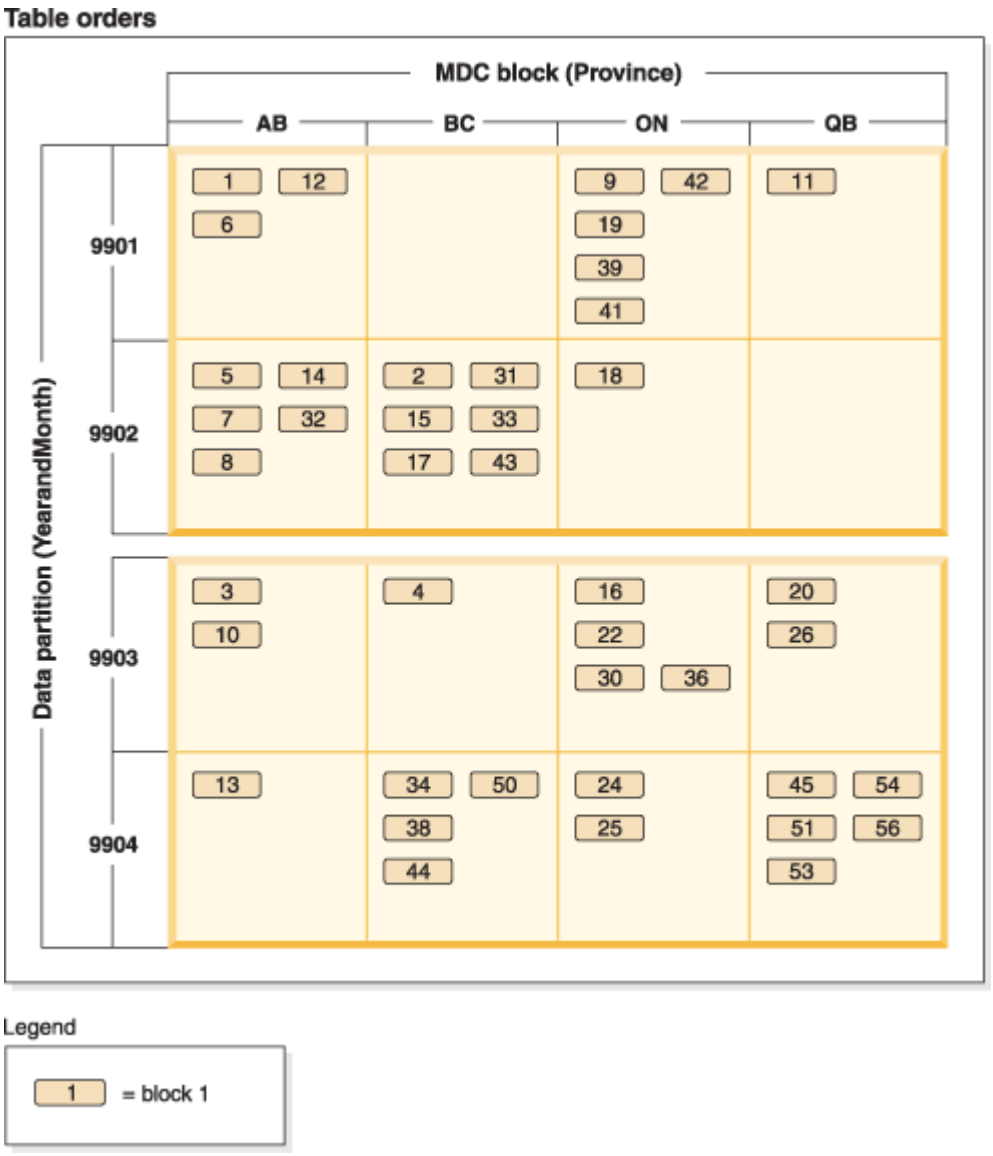


Figure 27. A table partitioned by YearAndMonth and organized by Province and YearAndMonth

In cases where the partitioning is such that there is only a single value in each range, nothing is gained by including the table partitioning column in the MDC key.

Considerations

- Compared to a basic table, both MDC tables and partitioned tables require more storage. These storage needs are additive but are considered reasonable given the benefits.
- If you choose not to combine table partitioning and MDC functionality in your partitioned database environment, table partitioning is best in cases where you can confidently predict the data distribution, which is generally the case for the types of systems discussed here. Otherwise, MDC should be considered.
- For a data-partitioned MDC table created with Db2 Version 9.7 Fix Pack 1 or later releases, the MDC block indexes on the table are partitioned. For a data-partitioned MDC table created with Db2 V9.7 or earlier releases, the MDC block indexes on the table are nonpartitioned.

Parallel database systems

Parallelism

Components of a task, such as a database query, can be run in parallel to dramatically enhance performance. The nature of the task, the database configuration, and the hardware environment, all determine how the Db2 database product will perform a task in parallel.

These factors are interrelated. Consider them all when you work on the physical and logical design of a database. The following types of parallelism are supported by the Db2 database system:

- I/O
- Query
- Utility

Input/output parallelism

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

Query parallelism

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

Interquery parallelism refers to the ability of the database to accept queries from multiple applications at the same time. Each query runs independently of the others, but the database manager runs all of them at the same time. Db2 database products have always supported this type of parallelism.

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

Intrapartition parallelism

Intrapartition parallelism refers to the ability to break up a query into multiple parts. Some Db2 utilities also perform this type of parallelism.

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

Figure 28 on page 62 shows a query that is broken into three pieces that can be run in parallel, with the results returned more quickly than if the query were run in serial fashion. The pieces are copies of each other. To use intrapartition parallelism, you must configure the database appropriately. You can choose the degree of parallelism or let the system do it for you. The degree of parallelism represents the number of pieces of a query running in parallel.

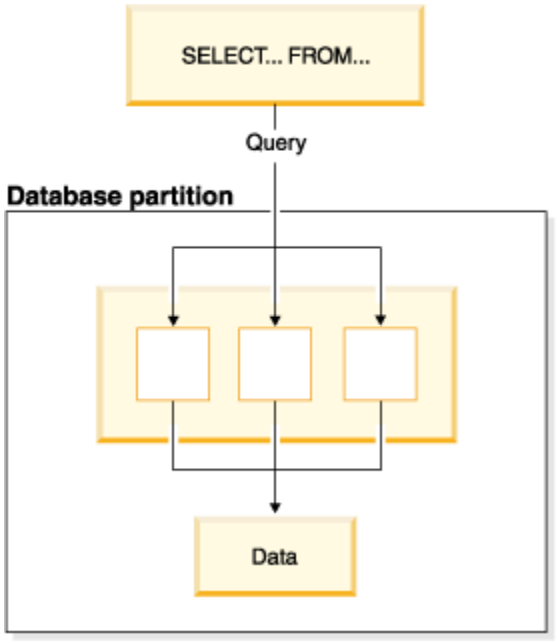


Figure 28. Intrapartition parallelism

Interpartition parallelism

Interpartition parallelism refers to the ability to break up a query into multiple parts across multiple partitions of a partitioned database, on one machine or multiple machines. The query is run in parallel. Some Db2 utilities also perform this type of parallelism.

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

Figure 29 on page 63 shows a query that is broken into three pieces that can be run in parallel, with the results returned more quickly than if the query were run in serial fashion on a single database partition.

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

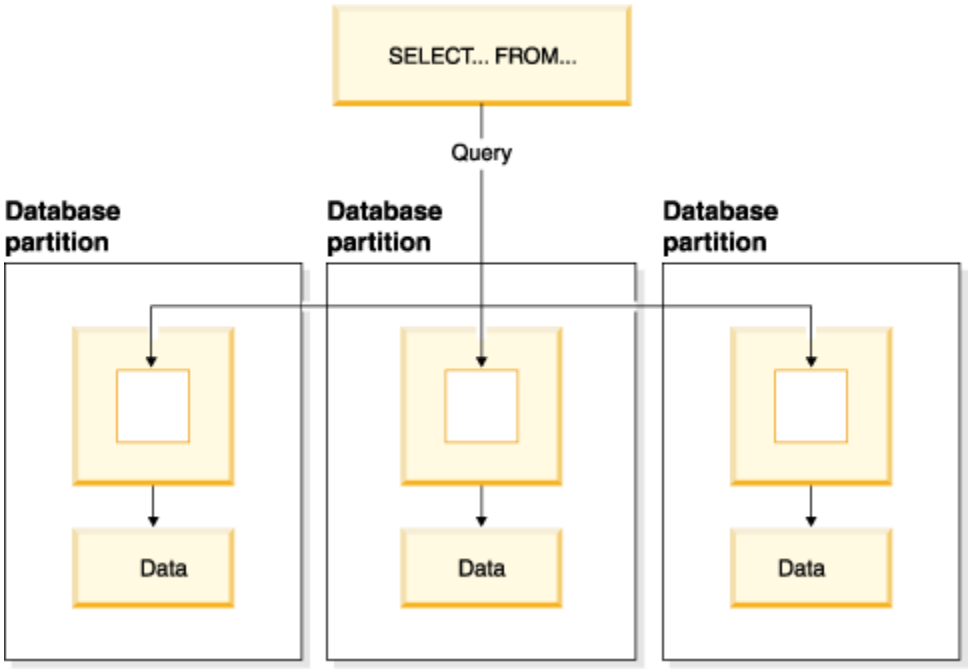


Figure 29. Interpartition parallelism

Simultaneous intrapartition and interpartition parallelism

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

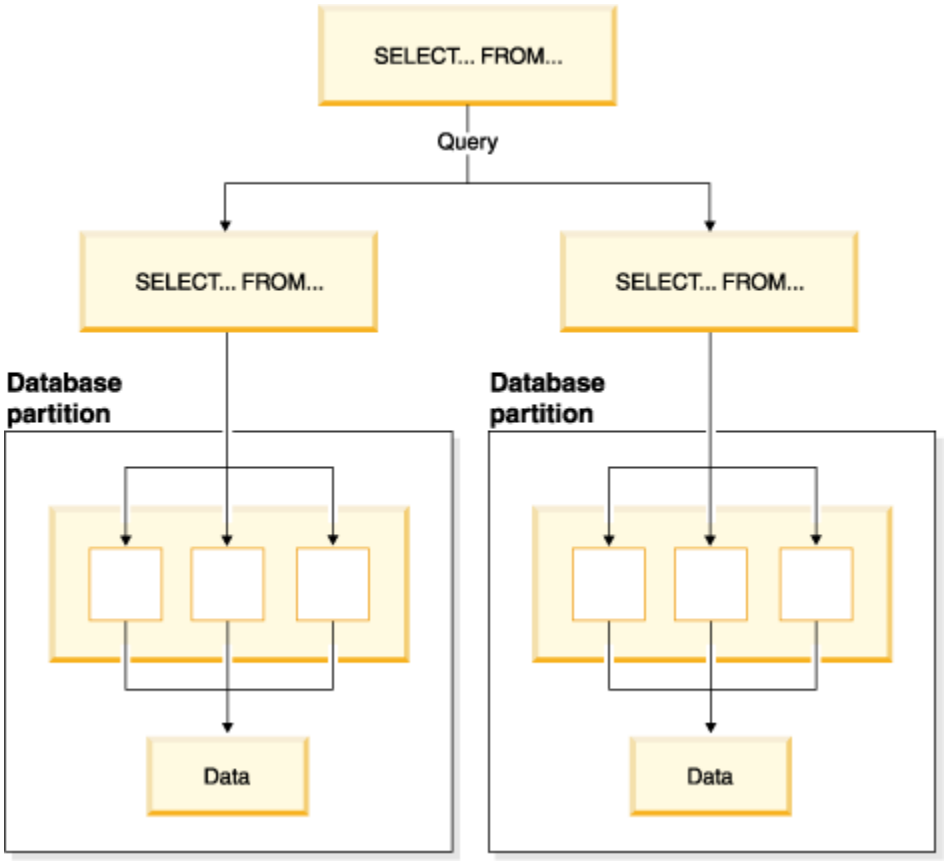


Figure 30. Simultaneous interpartition and intrapartition parallelism

Utility parallelism

Db2 utilities can take advantage of intrapartition parallelism. They can also take advantage of interpartition parallelism; where multiple database partitions exist, the utilities run in each of the database partitions in parallel.

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

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

During index creation, the scanning and subsequent sorting of the data occurs in parallel. The Db2 system exploits both I/O parallelism and intrapartition parallelism when creating an index. This helps to speed up index creation when a CREATE INDEX statement is issued, during restart (if an index is marked invalid), and during the reorganization of data.

Backing up and restoring data are heavily I/O-bound tasks. The Db2 system exploits both I/O parallelism and intrapartition parallelism when performing backup and restore operations. Backup exploits I/O parallelism by reading from multiple table space containers in parallel, and asynchronously writing to multiple backup media in parallel.

Partitioned database environments

A partitioned database environment is a database installation that supports the distribution of data across database partitions.

- A *database partition* is a part of a database that consists of its own data, indexes, configuration files, and transaction logs. A partitioned database environment is a database installation that supports the distribution of data across database partitions.
- A *single-partition database* is a database having only one database partition. All data in the database is stored in that single database partition. In this case, database partition groups, although present, provide no additional capability.
- A *multi-partition database* is a database with two or more database partitions. Tables can be located in one or more database partitions. When a table is in a database partition group consisting of multiple database partitions, some of its rows are stored in one database partition, and other rows are stored in other database partitions.

Usually, a single database partition exists on each physical machine, and the processors on each system are used by the database manager at each database partition to manage its part of the total data in the database.

Because data is distributed across database partitions, you can use the power of multiple processors on multiple physical machines to satisfy requests for information. Data retrieval and update requests are decomposed automatically into sub-requests, and executed in parallel among the applicable database partitions. The fact that databases are split across database partitions is transparent to users issuing SQL statements.

User interaction occurs through one database partition, known as the *coordinator partition* for that user. The coordinator partition runs on the same database partition as the application, or in the case of a remote application, the database partition to which that application is connected. Any database partition can be used as a coordinator partition.

The database manager allows you to store data across several database partitions in the database. This means that the data is physically stored across more than one database partition, and yet can be accessed as though it were located in the same place. Applications and users accessing data in a multi-partition database are unaware of the physical location of the data.

Although the data is physically split, it is used and managed as a logical whole. Users can choose how to distribute their data by declaring distribution keys. Users can also determine across which and over how many database partitions their data is distributed by selecting the table space and the associated

database partition group in which the data is to be stored. Suggestions for distribution and replication can be done using the Db2 Design Advisor. In addition, an updatable distribution map is used with a hashing algorithm to specify the mapping of distribution key values to database partitions, which determines the placement and retrieval of each row of data. As a result, you can spread the workload across a multi-partition database for large tables, and store smaller tables on one or more database partitions. Each database partition has local indexes on the data it stores, resulting in increased performance for local data access.

Note: You are not restricted to having all tables divided across all database partitions in the database. The database manager supports *partial declustering*, which means that you can divide tables and their table spaces across a subset of database partitions in the system.

An alternative to consider when you want tables to be positioned on each database partition, is to use materialized query tables and then replicate those tables. You can create a materialized query table containing the information that you need, and then replicate it to each database partition.

A non-root installation of a Db2 database product does not support database partitioning. Do not manually update the `db2nodes.cfg` file. A manual update returns an error (SQL6031N).

Related information

Best practices: Managing data growth

Database partition and processor environments

This section provides an overview of both single database partition and multiple database partition configurations. The former include single processor (uniprocessor) and multiple processor (SMP) configurations, and the latter include database partitions with one processor (MPP) or multiple processors (cluster of SMPs), and logical database partitions.

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

Single database partition on a single processor

This environment is made up of memory and disk, but contains only a single CPU (see Figure 31 on page 65). The database in this environment serves the needs of a department or small office, where the data and system resources (including a single processor or CPU) are managed by a single database manager.

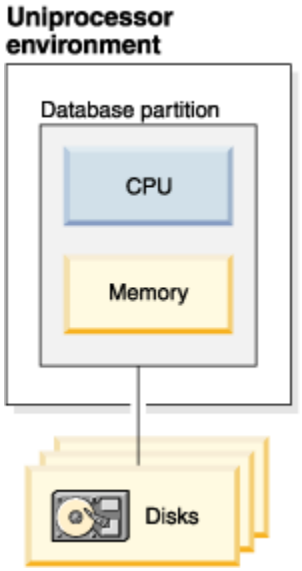


Figure 31. Single database partition on a single processor

Capacity and scalability

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

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

Single database partition with multiple processors

This environment is typically made up of several equally powerful processors within the same machine (see Figure 32 on page 66), and is called a *symmetric multiprocessor (SMP)* system. Resources, such as disk space and memory, are shared.

With multiple processors available, different database operations can be completed more quickly. Db2 database systems can also divide the work of a single query among available processors to improve processing speed. Other database operations, such as loading data, backing up and restoring table spaces, and creating indexes on existing data, can take advantage of multiple processors.

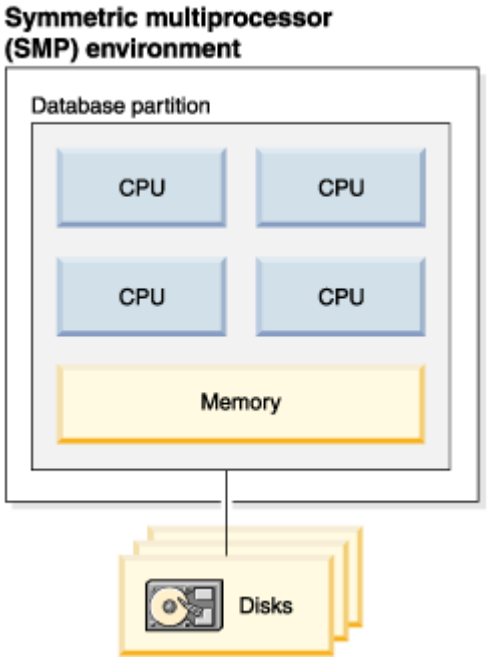


Figure 32. Single partition database symmetric multiprocessor environment

Capacity and scalability

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

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

Multiple database partition configurations

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

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

Database partitions with one processor

In this environment, there are many database partitions. Each database partition resides on its own machine, and has its own processor, memory, and disks (Figure 33 on page 67). All the machines are connected by a communications facility. This environment is referred to by many different names, including: cluster, cluster of uniprocessors, massively parallel processing (MPP) environment, and shared-nothing configuration. The latter name accurately reflects the arrangement of resources in this environment. Unlike an SMP environment, an MPP environment has no shared memory or disks. The MPP environment removes the limitations introduced through the sharing of memory and disks.

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

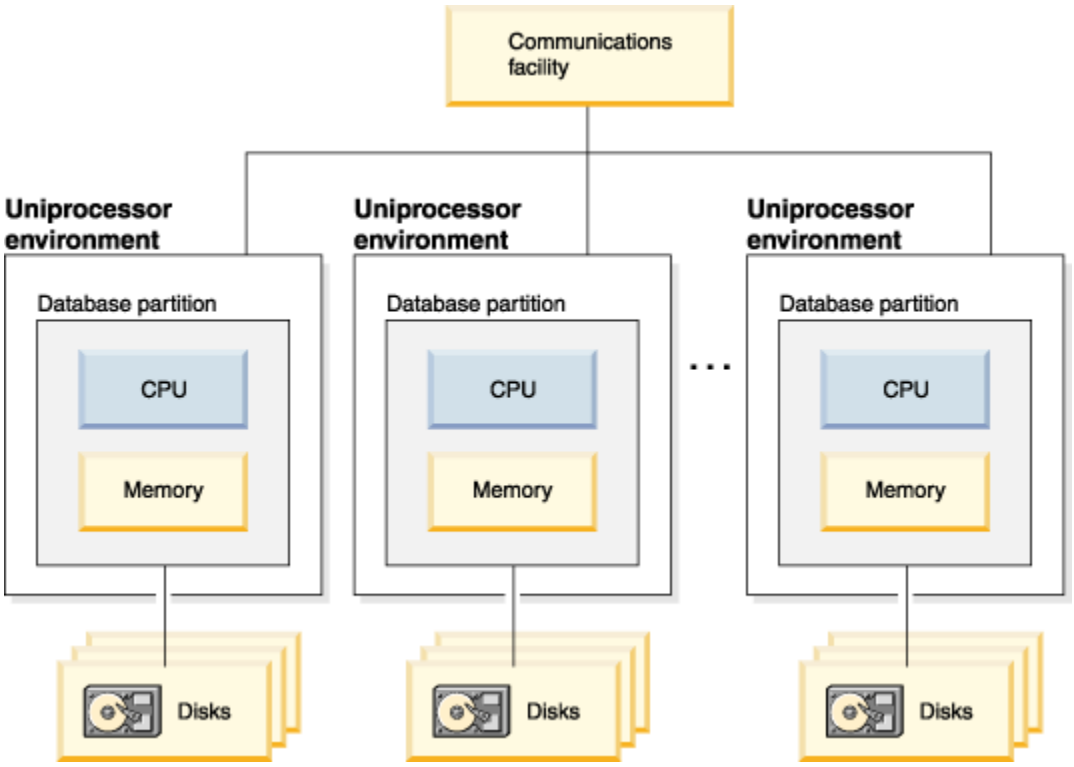


Figure 33. Massively parallel processing (MPP) environment

Capacity and scalability

In this environment you can add more database partitions to your configuration. On some platforms the maximum number is 512 database partitions. However, there might be practical limits on managing a high number of machines and instances.

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

Database partitions with multiple processors

An alternative to a configuration in which each database partition has a single processor, is a configuration in which each database partition has multiple processors. This is known as an *SMP cluster* (Figure 34 on page 68).

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

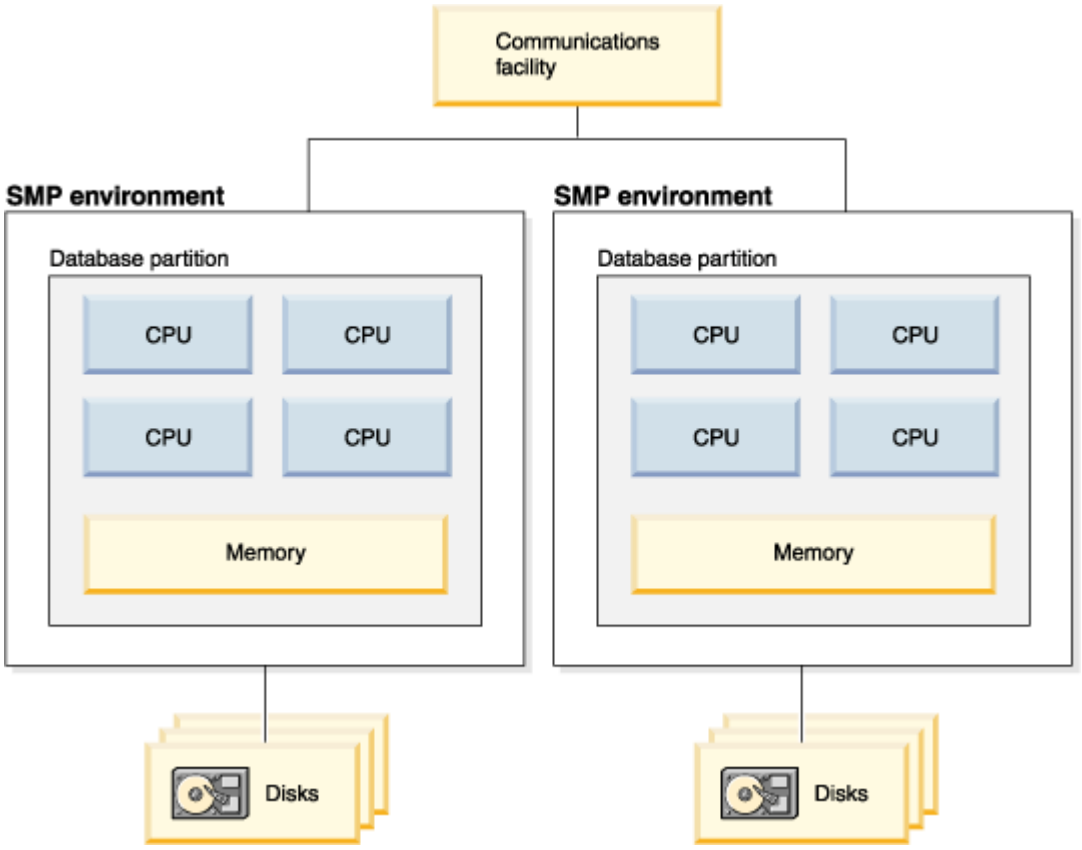


Figure 34. Several symmetric multiprocessor (SMP) environments in a cluster

Capacity and scalability

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

Logical database partitions

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

Logical database partitions provide scalability. Multiple database managers running on multiple logical partitions can make fuller use of available resources than a single database manager can. Figure 35 on page 69 illustrates the fact that you can gain more scalability on an SMP machine by adding more database partitions; this is particularly true for machines with many processors. By distributing the database, you can administer and recover each database partition separately.

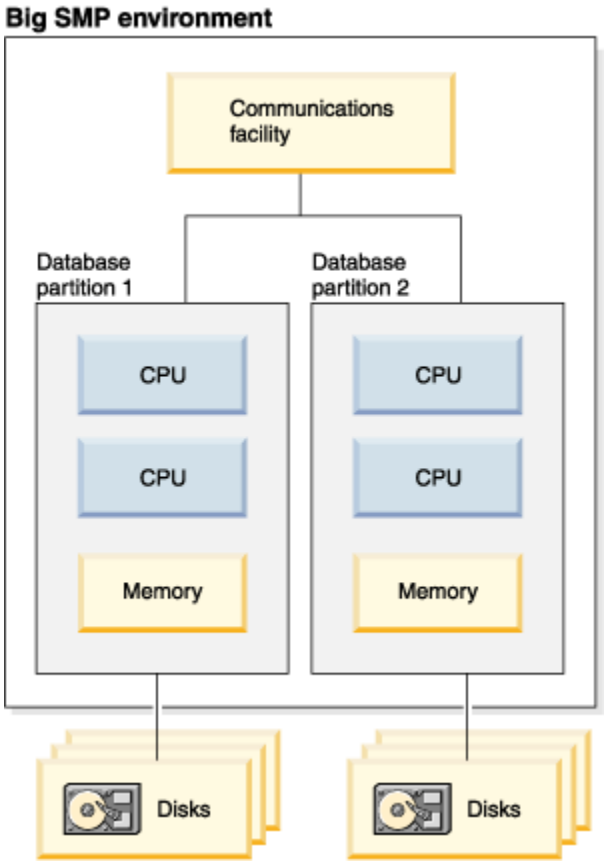


Figure 35. Partitioned database with symmetric multiprocessor environment

Figure 36 on page 70 illustrates that you can multiply the configuration shown in Figure 35 on page 69 to increase processing power.

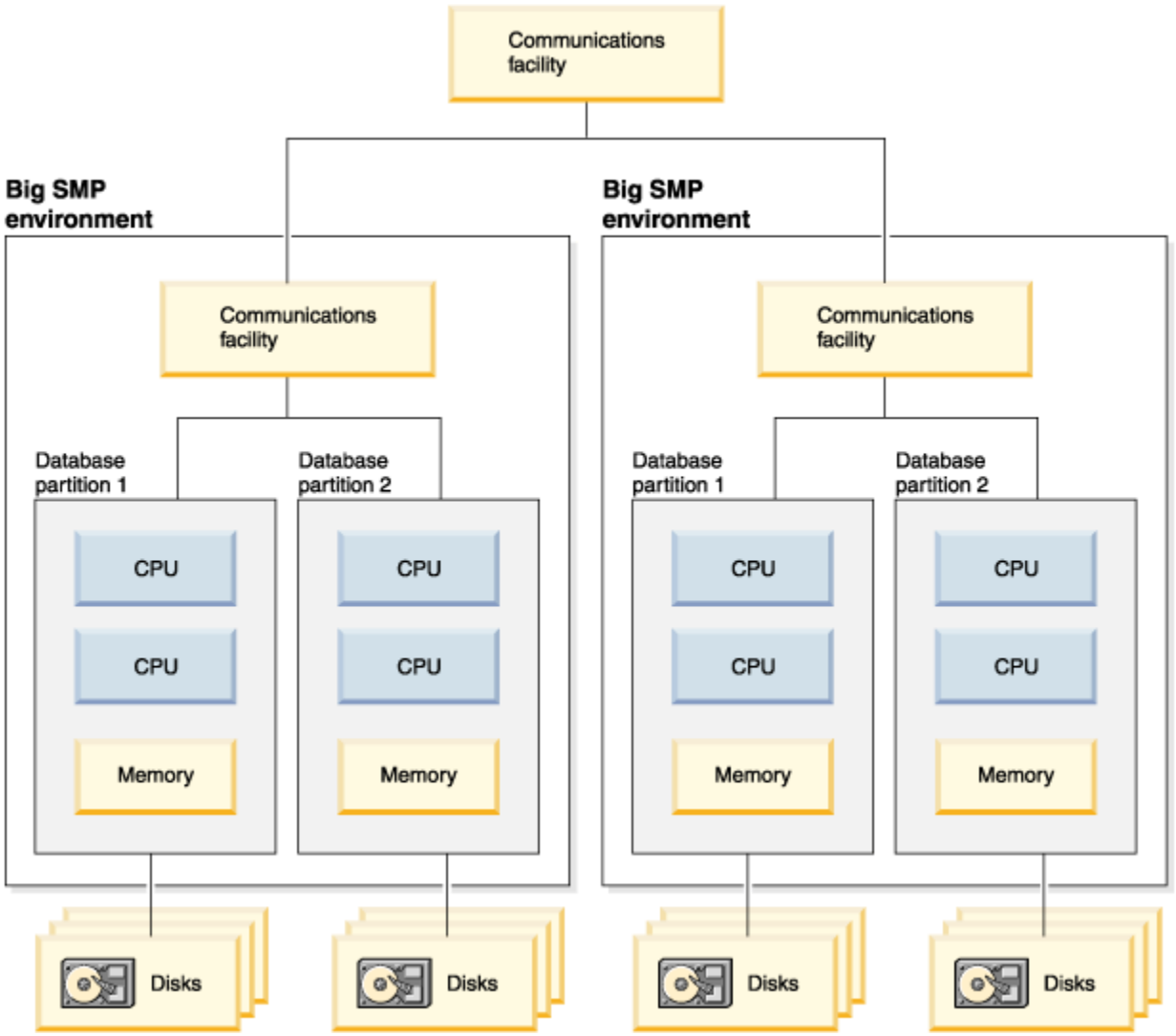


Figure 36. Partitioned database with symmetric multiprocessor environments clustered together

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

Summary of parallelism best suited to each hardware environment

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

Table 8. Types of Possible Parallelism in Each Hardware Environment

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

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

Hardware Environment	I/O Parallelism	Intra-Query Parallelism	
		Intra-Partition Parallelism	Inter-Partition Parallelism
Multiple Database Partitions, One Processor (MPP)	Yes	No [1]	Yes
Multiple Database Partitions, Multiple Processors (cluster of SMPs)	Yes	Yes	Yes
Logical Database Partitions	Yes	Yes	Yes

[1] There can be an advantage to setting the degree of parallelism (using one of the configuration parameters) to some value greater than one, even on a single processor system, especially if your queries are not fully using the CPU (for example, if they are I/O bound).

Chapter 2. Installation considerations

Installation prerequisites

Installing Db2 (Windows)

This task describes how to download and install the Db2 Community Edition for Windows. Db2 Community Edition is a full-featured server installation. After a three-month trial period, you can apply either a *Standard* or *Advanced* license to receive product support and additional functionality.

Before you begin

Before you download and install Db2 Community Edition:

- If you are planning on setting up a partitioned database environment, refer to Setting up a partitioned database environment.
- Ensure that your system meets installation, memory, and disk requirements.
- If you are planning to use LDAP to register the Db2 server in Windows operating systems Active Directory, extend the directory schema before you install. Otherwise, you must manually register the node and catalog the databases. For more information, see Extending the Active Directory Schema for LDAP directory services (Windows).
- You must have a local Administrator user account with the recommended user rights to perform the installation. In Db2 database servers where LocalSystem can be used as the DAS and Db2 instance user and you are not using the database partitioning feature, a non-administrator user with elevated privileges can perform the installation.
- Although not mandatory, it is recommended that you close all programs so that the installation program can update any files on the computer without requiring a reboot.
- Installing Db2 products from a virtual drive or an unmapped network drive (such as *hostname* *sharename* in Windows Explorer) is not supported. Before you attempt to install Db2 products, you must map the network drive to a Windows drive letter (for example, Z:).

Restrictions

- You cannot have more than one instance of the **Db2 Setup** wizard that is running in any user account.
- The Db2 copy name and the instance name cannot start with a numeric value.The Db2 copy name is limited to 64 English characters that consists of the characters A-Z, a-z and 0-9.
- The Db2 copy name and the instance name must be unique among all Db2 copies.
- The use of XML features is restricted to a database that has only one database partition.
- If you installed one of the following products, then no other Db2 database product can be installed in the same path:
  - IBM® Data Server Runtime Client
  - IBM Data Server Driver Package
- The **Db2 Setup** wizard fields do not accept non-English characters.
- If you enable extended security on Windows or higher, the users must belong to the DB2ADMNS or DB2USERS group to run local Db2 commands and applications. This setup is required because of an extra security feature (User Access Control) that limits the privileges that local administrators have by default. If users do not belong to one of these groups, they will not have read access to local Db2 configuration or application data.

Procedure

Use the **Db2 Setup** wizard to define your installation and install your Db2 database product on your system.

To install Db2 Community Edition on a Windows machine:

1. Go to the ***Db2 Download*** page and sign in with your IBM account credentials.
2. Choose your contact option (e.g., email), review the Terms and Conditions, and then click **Continue**.
3. From the ***Downloads*** page that appears, locate the *IBM® Db2 11.5 Edition for Windows® on AMD64 and Intel® EM64T systems* option and click the associated **Download now** link.
4. From the pop-up box that appears, choose **Save File**.

The file is downloaded to your browser's default location on your Windows server machine.

5. Log on to the system with the local Administrator account.
6. Using Windows Explorer, navigate to the `v11.5_ntx64_dec.zip` file that you downloaded and extract it the desired location on your server.
7. Expand the extracted folder, `\v11.5_ntx64_dec\SERVER_DEC\image`, right-click the **setup.exe** file and select **Run as administrator** to start the Db2 Setup Launchpad..

Attention: You must never start the installation package(".msi" file) present in the Db2 installation media directly. Installation must always be started by **`setup.exe`** with full administrative privileges.

8. If a **User Account Control** window is displayed, click **Yes**.
9. Click **Install a Product**.
10. Scroll to the end of the **DB2 Version 11.5.0.0 Server Editions** and click **Install New**.
11. Click **Next** to begin the installation.
12. Review the software licence agreement, click the option to accept, and then click **Next**.
13. Select the **Typical** install option and then click **Next**.
14. Select the option, **Install DB2 Server Edition on this computer and save my settings in a response file** and then click **Next**.
15. Accept the default install path and then click **Next**.
16. Select the option, **Do no autostart the IBM SSH server** at system startup and then click **Next**.
17. From the ***Domain*** drop-down list box, select **None- use local user account** as the domain name, enter and confirm a password, and then click **Next**.
18. Accept ***DB2*** as the default instance and then click **Next**.
19. Ensure that the **Set up your DB2 server to send notifications** check box is NOT SELECTED and then click **Next**.
20. Select the option, **enable operating system security for DB2 objects**, and then click **Next**.
21. When the ***Start copying files*** window is displayed, click **Finish**.
22. When the setup is complete, click **Finish**.

Results

Your Db2 database product is installed, by default, in the *`Program_Files`*`\IBM\sqllib` directory, where *Program_Files* represents the location of the `Program Files` directory.

If you are installing on a system where this directory is already being used, the Db2 database product installation path has *_xx* added to it. The *xx* are digits, starting at `01` and increasing, depending on how many Db2 copies you installed.

You can also specify your own Db2 database product installation path.

What to do next

- Verify your installation. You can use the db2val tool for verification.

- Perform the necessary post-installation tasks.

For information about errors that are encountered during installation, review the installation log file that is located in the `My Documents\DB2LOG\` directory. The log file uses the following format: `DB2-`*`ProductAbrrev-DateTime`*`.log`, for example, `DB2-ESE-Tue Apr 04 17_04_45 2012.log`.

If this is a new Db2 product installation on Windows 64–bit, and you use a 32–bit OLE DB provider, you must manually register the IBMDADB2 DLL. To register this DLL, run the following command:

```
c:\windows\SysWOW64\regsvr32 /s c:\Program_Files\IBM\SQLLIB\bin\ibmdadb2.dll
```

where *Program_Files* represents the location of the `Program Files` directory.

IBM Data Studio can be installed by running the **Db2 Setup** wizard.

Preparing the environment for a partitioned Db2 server (Windows)
This topic describes the steps required to prepare your Windows environment for a partitioned installation of the Db2 database product.

Before you begin
When you add a new machine as a partition in a partitioned database environment, the new machine must:

- Have the same operating system version as the instance owning machine.
- Have the same CPU architecture (x32 bit or x64 bit) as the instance owning machine.

If the new machine does not meet these requirements, adding the partition might fail.

Procedure

To prepare your Windows environment for installation:

1. Ensure that the primary computer and participating computers belong to the same Windows domain.

   Check the domain to which the computer belongs by using the **System Properties** dialog, accessible through the **Control Panel**.
2. Ensure that time and date settings on the primary computer and participating computers are consistent.

   To be considered consistent, the difference in GMT time between all computers must be no greater than one hour.

   System date and time can be modified using the **Date/Time Properties** dialog, accessible through the **Control Panel**. You can use the **`max_time_diff`** configuration parameter to change this restriction. The default is `max_time_diff = 60`, which allows a difference of less than 60 minutes.
3. Ensure that each computer object that participates in the partitioned database environment has the "Trust computer for delegation" privilege flagged.

   You can verify that the "Trust computer for delegation" check box on the **General** tab of each computer's account **Properties** dialog box in the **Active Directory Users and Computers** console is checked.
4. Ensure that all participating computers can communicate with each other using TCP/IP:
   a) On one participating computer, enter the **`hostname`** command, which will return the hostname of the computer.
   b) On another participating computer, enter the following command:

      ```
      ping hostname
      ```

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

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

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

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

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

If you are planning to use multiple network adapters, you can specify which adapter to use to communicate between database partition servers. Use the**db2nchg** command to specify the netname field in the db2nodes.cfg file after the installation is complete.

5. During the installation you will be asked to provide a Db2 Administration Server user account.

   This is a local or domain user account that will be used by the Db2 Administration Server (DAS). The DAS is an administration service used to support the GUI tools and assist with administration tasks. You can define a user now or have the **Db2 Setup** wizard create one for you. If you want to create a new domain user using the **Db2 Setup** wizard, the account used to perform the installation must have authority to create domain users.
6. On the primary computer, where you will install the instance-owning partition, you must have a domain user account that belongs to the local *Administrators* group.

   You will log on as this user when you install Db2 database products. You must add the same user account to the local *Administrators* group on each participating computer. This user must have the *Act as part of the operating system* user right.
7. Ensure that all computers in the instance have the database directory on the same local drive letter.

   You can check this condition by running the **GET DATABASE CONFIGURATION** command and verifying the value of the **dftdbpath** DBM configuration parameter.
8. During the installation you will be asked to provide a domain user account to be associated with the Db2 instance.

   Every Db2 instance has one user assigned. The Db2 database system logs on with this user name when the instance is started. You can define a user now, or you can have the **Db2 Setup** wizard create a new domain user for you.

   When adding a new node to a partitioned environment the Db2 copy name must be the same on all computers.

   If you want to create a new domain user using the **Db2 Setup** wizard, the account used to perform the installation must have authority to create domain users. The instance user domain account must belong to the local *Administrators* group on all the participating computers and will be granted the following user rights:

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

   If extended security was selected, the account must also be a member of the DB2ADMNS group. The DB2ADMNS group already has these privileges so the privileges are already explicitly added to the account.

Fast communications manager (Windows)

In multiple member environments, each member has a pair of FCM daemons to support communication between members that is related to agent requests. One daemon is for sending communications, and the other is for receiving. These daemons and supporting infrastructure are activated when an instance is started. FCM communication is also used for agents working within the same member; this type of communication is also known as intra-member communication.

You can specify the number of FCM message buffers by using the **fcm_num_buffers** database manager configuration parameter. You can specify the number of FCM channels by using the **fcm_num_channels** database manager configuration parameter. By default, the **fcm_num_buffers** and **fcm_num_channels** database manager configuration parameters are set to AUTOMATIC. If the setting is AUTOMATIC, which is the recommended setting, the FCM monitors resource usage and adjusts resources to accommodate workload demand.

An overview of installing Db2 database servers (Linux and UNIX)

This topic outlines the steps for installing your Db2 server product on AIX, and Linux.

Procedure

To install your Db2 server product:

1. Review your Db2 product prerequisites.
2. Review Db2 upgrade information if applicable.
3. On all platforms, except for Linux on x86_32, you must install a 64-bit kernel before proceeding with the installation, otherwise the installation will fail.
4. Download an installation image and extract the file.
5. Install your Db2 database product using:
   - the **Db2 Setup** wizard.
   - a silent installation with a response file.
6. Install your Db2 product using one of the available methods:
   - The Db2 Setup wizard
   - A silent installation using a response file
   - Payload file deployment

   For Db2 servers, you can use the Db2 Setup wizard to perform installation and configuration tasks, such as:
   - Selecting Db2 installation type (typical, compact, or custom).
   - Selecting Db2 product installation location.
   - Install the languages that you can specify later as the default language for the product interface and messages.
   - Install or upgrade the IBM Tivoli® System Automation for Multiplatforms (Linux and AIX).
   - Setting up a Db2 instance.
   - Setting up the Db2 Administration Server (including DAS user setup).
   - Setting up the Db2 Text Search server.
   - Setting up Administration contact and health monitor notification.
   - Setting up and configuring your instance setup and configuration (including instance user setup).
   - Setting up Informix data source support.
   - Preparing the Db2 tools catalog.
   - Creating response files.
7. If you installed a Db2 server using a method other than the Db2 Setup wizard, post-installation configuration steps are required.

Db2 installation methods

You can install Db2 database products in multiple methods. Each installation method is suited for specific circumstances.

The following table shows the installation methods that are available by operating system.

Table 9. Installation method by operating system.		
Installation method	**Windows**	**Linux or UNIX**
Db2 Setup wizard	Yes	Yes
Response file installation	Yes	Yes
db2_install command	No	Yes
Payload file deployment	No	Yes

The following list describes Db2 installation methods.

Db2 Setup wizard
The **Db2 Setup** wizard is a GUI installer available on Linux, UNIX, and Windows operating systems. The **Db2 Setup** wizard provides an easy-to-use interface for installing Db2 database products and executing initial setup and configuration tasks.

The **Db2 Setup** wizard can also create Db2 instances and response files that can be used to duplicate this installation on other workstations.

Note: For non-root installations on Linux and UNIX operating systems, only one Db2 instance can exist. The **Db2 Setup** wizard automatically creates the non-root instance.

On Linux and UNIX operating systems, to install a Db2 product by using the Db2 Setup wizard, you require an X Window System (X11) to display the graphical user interface (GUI). To display the GUI on your local workstation, the X Window System software must be installed and running. You must also set the DISPLAY variable to the IP address of the workstation you use to install the Db2 product (export DISPLAY=*<ip-address>*:0.0). For example, export DISPLAY=192.168.1.2:0.0. For more information, see this article.

Db2 Docker image
Docker containers provide you with a virtualized run-time environment from which to run Db2, without impacting your existing operating system. You can now pull a Docker image of Db2 Community Edition from Docker Hub and install it in a Docker container on Windows 10, Mac OS 10.10 or higher, an a number of Linux distributions. See Installing Db2 Community Edition with Docker for more information.

Response file installation
A response file is a text file that contains setup and configuration values. The file is read by the **Db2 Setup** program and the installation is executed according to the values that were specified.

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

Another advantage to response files is that they provide access to parameters that cannot be set by using the **Db2 Setup** wizard.

On Linux and UNIX operating systems, if you embed the Db2 installation image in your own application, your application might receive installation progress information and prompts from the installer in computer-readable form. This behavior is controlled by the **INTERACTIVE** response file keyword.

A response file can be created in a number of ways:

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

Using the Db2 Setup wizard.
The **Db2 Setup** wizard can create a response file based on the selections you make as you proceed through the **Db2 Setup** wizard. Your selections are recorded in a response file that you can save to a location on your system. If you select a partitioned database installation, two

response files are generated, one for the instance-owning computer and one for participating computers.

One benefit of this installation method is that you can create a response file without performing an installation. This feature can be useful to capture the options that are required to install the Db2 database product. The response file can be used later to install the Db2 database product according to the exact options you specified.

You can export a client or server profile with the **db2cfexp** command to save your client or server configuration. Import the profile by using the **db2cfimp** command. A client or server profile that is exported with the **db2cfexp** command can also be imported during a response file installation by using the **CLIENT_IMPORT_PROFILE** keyword.

Export the client or server profile after you install and catalog any data sources.

Customizing the sample response files that are provided for each Db2 database product.
An alternative to using the response file generator or the **Db2 Setup** wizard to create a response file is to manually modify a sample response file. Sample response files are provided on the Db2 database installation media. The sample response files provide details about all the valid keywords for each product.

db2_install command (Linux and UNIX operating systems only)
The **db2_install** command installs all components for the Db2 database product you specify with the English interface support. You can select additional languages to support with the **-L** parameter. You cannot select or clear components.

Although the **db2_install** command installs all components for the Db2 database product you specify, it does not install user and group creation, instance creation, or configuration. This method of installation might be preferred in cases where configuration is to be done after installation. To configure your Db2 database product while you install it, you can use the **Db2 Setup** wizard.

On Linux and UNIX operating systems, if you embed the Db2 installation image in your own application, your application might receive installation progress information and prompts from the installer in computer-readable form.

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

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

This method is not supported for Db2 pureScale installation.

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

Note: Db2 database product installations are no longer operating system packages on Linux and UNIX. As a result, you can no longer use operating system commands for installation. Any existing scripts that you use to interface and query with Db2 database product installations must change.

Installing Db2 servers using the Db2 Setup wizard (Linux and UNIX)
This task describes how to download and install the Db2 Community Edition for Linux and UNIX operating systems. Db2 Community Edition is a full-featured server installation. After a three-month trial period, you can apply either a Standard or Advanced license to receive product support and additional functionality.

Before you begin

Before you start the **Db2 Setup** wizard:

- If you are planning on setting up a partitioned database environment, refer to "Setting up a partitioned database environment" in *Installing Db2 Servers*.
- Ensure that your system meets installation, memory, and disk requirements.
- Ensure that you installed a supported browser.

- You can install a Db2 database server by using either root or non-root authority. For more information about non-root installation, see Non-root installation overview (Linux and UNIX). When Ubuntu Linux 18.04 LTS is installed, **root** login is disabled by default. Follow these steps to enable root user access:

  1. From the terminal window, enter this command:

     ```
     sudo passwd root
     ```

  2. Enter the password for the administrative user account when the prompt **[sudo] password for [AdminUser]:** is displayed.
  3. Press ENTER.
  4. Create a new password for the root user when the prompt **Enter new Unix Password** is displayed. Re-enter the same password in the next prompt.

  You should see this message:

  ```
  passwd: password updated successfully
  ```

  If this message is not displayed, go through the steps again, making any corrections suggested by the error message or messages that are returned.
- The Db2 database product image must be available. You can obtain a Db2 installation image by downloading an installation image from Passport Advantage®.
- If you are installing a non-English version of a Db2 database product, you must have the appropriate National Language Packages.
- The Db2 Setup wizard is a graphical installer. To display the graphical user interface (GUI) on your local workstation, the X Window System (X11) software must be installed and running. You must also set the DISPLAY variable to the IP address of the workstation you use to install the Db2 product:

  ```
  export DISPLAY=<ip-address>:0.0
  ```

  For example,

  ```
  export DISPLAY=192.168.1.2:0.0
  ```

  For more information, see this developerWorks article.
- If you are using the security software in your environment, you must manually create required Db2 users before you start the **Db2 Setup** wizard.

Prerequisites

Before you begin, ensure that you have access to the following software on your server machine:

- **Ubuntu Linux 18.04 LTS (Desktop)*** – The latest long-term support (LTS) version of the Ubuntu Linux operating system.
- **Db2 Community Edition Version 11.5.0.0** – Free, fully functional version of Db2 intended for development, test, and small production environments.
- **libaio1** – Linux kernel asynchronous input/output (I/O) shared library. (Enables a single application thread to overlap I/O operations with other processing, by providing an interface for submitting one or more I/O requests in a single system call without waiting for completion.)
- **binutils** – GNU Binary Utilities shared library. (A set of programming tools for creating and managing binary programs, object files, libraries, profile data, and assembly source code.)
- **liblogger-syslog-perl** – An interface to the UNIX program that sends messages to the system logger, written in Perl. (Takes care of everything regarding Syslog communication; provides one function for each of the following Syslog message levels: debug, info, warning, error, notice, critical, and alert.)
- **zlib1g-dev** – A shared Linux library that implements the deflate compression method found in gzip and PKZIP.
- **libpam0g:i386** – Linux Pluggable Authentication Modules for Intel x86. (Enables the local system administrator to choose how applications authenticate users.)

- **libstdc++6:i386** – GNU Standard C++ Library, Version 3 for Intel x86. (A runtime library for C++ programs built with the GNU compiler.)

To install the newest versions of all Linux packages that are currently on the system, follow these steps:

1. Open a terminal window if you do not already have one open.
2. Run the command **su - root**.
3. When the **Password** prompt is displayed, provide the password you created for the root user earlier. The command line prompt should change from $ to #.
4. Run the command **apt-get upgrade** to fetch and install the newest versions of all Linux packages that are currently on the system.
5. Run the following commands to install the prerequisite Linux packages:

```
apt-get -y install libaio1
apt-get -y install binutilsapt-get -y install zlib1g-dev
apt-get -y install liblogger-syslog-perl
apt-get -y install libpam0g:i386
apt-get -y install libstdc++6:i386
```

6. Run the command **apt-get update** to update the list of all Linux packages that are installed on the system.

Restrictions

- You cannot have more than one instance of the ***Db2 Setup*** wizard running in any user account.
- The use of XML features is restricted to a database that is defined with the code set UTF-8 and has only one database partition.
- The ***Db2 Setup*** wizard fields do not accept non-English characters.

Procedure

Use the ***Db2 Setup*** wizard to define your installation and install the Db2 database product on your system.

To install Db2 Community Edition on a Linux machine:

1. Go to the ***Db2 Download*** page and sign in with your IBM account credentials.
2. Choose your contact option (for example, email), review the Terms and Conditions, and click **Continue**.
3. Locate the *IBM® Db2 for Linux® on AMD64 and Intel® EM64T systems (x64)* option from the ***Downloads*** page and click the associated **Download now** link.
4. Choose **Save File** from the pop-up box.

   You can now extract the downloaded software and install Db2 Community Edition on your machine.
5. Open a terminal window.
6. Run the command **su - root**.
7. Provide the correct password for the root user when the `password` prompt is displayed.
8. Create a directory named `software` in the */home* directory and run the following commands to make the software directory accessible to everyone:

```
mkdir /home/software
chmod 777 /home/software
```

9. Run the following command to move the downloaded file to the directory:

```
mv /home/AdminUser/Download/*.gz /home/software
```

10. Go to the `/home/software` directory and run the following commands to extract and untar the file.

On Linux operating systems:

```
cd /home/software
gunzip *.gz
tar -xvf v11.5_linuxx64_dec.tar
```

On AIX operating systems:

```
cd /home/software
gunzip *.gz
gnutar -xvf v11.5_aix64_server.tar.gz
```

11. Delete the `.tar` file that was created in the previous step.
12. Make the sub-directory (`server_dec`) accessible to everyone, then run the following commands to rename the sub-directory to ibm-db2:

```
rm -f v*.tar
chmod 777 server_dec
mv server_dec ibm-db2
```

13. Move to the `/home/software/ibm-db2` directory and run the following commands to verify that the server has everything needed to install and run Db2:

```
cd ibm-db2
./db2prereqcheck -v 11.5.0.0s
```

When all checks are complete, the following message is displayed:

```
DBT3533I The db2prereqcheck utility has confirmed that all installation prerequisites were
met.
```

If this message is not displayed, review the output of the db2`prereqcheck` utility and resolve any problems identified.

14. Run the command **`./db2setup`** to load and start the Db2 Setup/Installation program. From the Welcome window, click **New Install**.
15. Select **DB2 Version 11.5.0.0 Server Editions** and then click **Next**.
16. Click **Click to view** to view the IBM terms . Click the **I agree to the IBM terms** check box and then click **Next**.
17. Enter a password for the Db2 instance owner (**db2inst1**). The instance owner username and password provided is the user ID and password you will use to work with Db2. Click **Next**.
18. Enter a password for the Db2 fenced user (**db2fenc1**). The fenced user is used to run user-defined functions and stored procedures outside of the address space that is used by a Db2 database. Click **Next**.
19. Verify that the **Install DB2 Server Edition on this computer and save my settings in a response file** option is selected.
20. Click **Finish** to start the Db2 installation process.
21. When the installation process is complete, click **Post-install steps**.

    Carefully read the information:

    - Verify that the Db2 software was installed correctly
    - View your Db2 license entitlements
    - Start using Db2
    - Access the online Db2 documentation
22. Click **Close**.
23. Click **Log file** to review the contents of the log file after control returns to the ***Setup Complete*** window.
24. Click **Close** to return to the ***Setup Complete*** window. Click **Finish** to close the ***Db2 Setup*** installation program.

Results

For non-root installations, Db2 database products are always installed in the *$HOME*/sqllib directory, where *$HOME* represents the non-root user's home directory.

For root installations, Db2 database products are installed, by default, in one of the following directories:

AIX
/opt/IBM/db2/V11.5

Linux
/opt/ibm/db2/V11.5

If you are installing on a system where this directory is already being used, the Db2 database product installation path has *_xx* added to it. The *_xx* are digits, starting at 01 and increasing, depending on how many Db2 copies you have installed.

You can also specify a custom Db2 database product installation path.

Db2 installation paths have the following rules:

- Can include lowercase letters (a-z), uppercase letters (A-Z), and the underscore character (_).
- Cannot exceed 128 characters.
- Cannot contain spaces.
- Cannot contain characters that are other than English characters.
- The path name cannot be a subdirectory of an existing Db2 installation.
- The installation paths cannot be symbolic links.

The installation log files are:

- The Db2 setup log file. This file captures all Db2 installation information, including errors.
  - For root installations, the Db2 setup log file name is db2setup.log.
  - For non-root installations, the Db2 setup log file name is db2setup_*username*.log, where *username* is the non-root user ID under which the installation was performed.
- The Db2 error log file. This file captures any error output that is returned by Java™ (for example, exceptions and trap information).
  - For root installations, the Db2 error log file name is db2setup.err.
  - For non-root installations, the Db2 error log file name is db2setup_*username*.err, where *username* is the non-root user ID under which the installation was performed.

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

There is no longer a db2setup.his file. Instead, the Db2 installer saves a copy of the Db2 setup log file in the DB2_DIR/install/logs/ directory, and renames it db2install.history. If the name exists, then the Db2 installer renames it db2install.history.*xxxx*, where *xxxx* is 0000-9999, depending on the number of installations you have on that machine.

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

What to do next

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

National Language Packs can also be installed by running the **./db2setup** command from the directory, where the National Language Pack resides, after you install the Db2 database product.

On Linux x86, if you want your Db2 database product to have access to Db2 documentation either on your local computer or on another computer on your network, then you must install the *Db2 Information*

Center. The *Db2 Information Center* contains documentation for the Db2 database system and Db2 related products.

Fast communications manager (Linux and UNIX)
The fast communications manager (FCM) provides communications support for partitioned database environments.

In multiple member environments, each member has a pair of FCM daemons to support communication between members that is related to agent requests. One daemon is for sending communications, and the other is for receiving. These daemons and supporting infrastructure are activated when an instance is started. FCM communication is also used for agents working within the same member; this type of communication is also known as intra-member communication.

The FCM daemon collects information about communication activities. You can obtain information about FCM communications by using the database system monitor. If communications fail between members or if they re-establish communications, the FCM daemons update monitor elements with this information. The FCM daemons also trigger the appropriate action for this event. An example of an appropriate action is the rollback of an affected transaction. You can use the database system monitor to help you set the FCM configuration parameters.

You can specify the number of FCM message buffers by using the **fcm_num_buffers** database manager configuration parameter. You can specify the number of FCM channels by using the **fcm_num_channels** database manager configuration parameter. By default, the **fcm_num_buffers** and **fcm_num_channels** database manager configuration parameters are set to AUTOMATIC. If the setting is AUTOMATIC, which is the recommended setting, the FCM monitors resource usage and adjusts resources to accommodate workload demand.

Before you install

Additional partitioned database environment preinstallation tasks (Linux and UNIX)

Updating environment settings for a partitioned Db2 installation (AIX)

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

Procedure

To update AIX environment settings:

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

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

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

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

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

```
no -a | more
```

To set a parameter, enter the follow command:

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

where:

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

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

```
no -o tcp_sendspace=221184
```

4. If you are using a high speed interconnect, you must set the `spoolsize` and `rpoolsize` for `css0` to the following values:

```
spoolsize     16777216
rpoolsize     16777216
```

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

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

To set these parameters, enter the following commands:

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

If you are not using the `/tftpboot/tuning.cst` file to tune your system, you can use the *DB2DIR*`/misc/rc.local.sample` sample script file, where *DB2DIR* is path where the Db2 database product has been installed, to update the network-related parameters after installation. To update the network-related parameters using the sample script file after installation, perform the following steps:

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

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

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

c) Add an entry to the `/etc/inittab` file so that the `/etc/rc.local` script is executed whenever the machine is rebooted.

You can use the **mkitab** command to add an entry to the `/etc/inittab` file.

To add this entry, enter the following command:

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

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

```
lsitab rcnfs
```

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

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

5. Ensure that you have enough paging space for a partitioned installation of Db2 Enterprise Server Edition to run.

If you do not have sufficient paging space, the operating system will kill the process that is using the most virtual memory (this is likely to be one of the Db2 processes).

To check for available paging space, enter the following command:

```
lsps -a
```

This command will return output similar to the following:

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

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

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

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

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

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

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

Setting up a working collective to distribute commands to multiple AIX nodes

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

Before you begin

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

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

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

You must have a file that lists the host names for all of the workstations that will participate in your partitioned database system.

Procedure

To set up the working collective to distribute commands to a list of workstations:

1. Create a file called `nodelist.txt` that will list the host names for all of the workstations that will participate in the working collective.

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

```
workstation1
workstation2
```

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

```
export DSH_NODE_LIST=path/nodelist.txt
```

   where *path* is the location where `nodelist.txt` was created, and `nodelist.txt` is the name of the file that you created that lists the workstations in the working collective.

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

```
dsh -q
```

You will receive output similar to the following:

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

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

Procedure

- To verify that NFS is running on each computer:
  - AIX operating systems:

    Type the following command on each computer:

    ```
    lssrc -g nfs
    ```

    The `Status` field for NFS processes should indicate `active`. After you have verified that NFS is running on each system, you should check for the specific NFS processes required by Db2 database products. The required processes are:

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

  - Linux operating systems:

    Type the following command on each computer:

    ```
    showmount -e hostname
    ```

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

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

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

    After you have verified that NFS is running on each system, you should check for the specific NFS processes required by Db2 database products. The required process is `rpc.statd`.

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

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

Verifying port range availability on participating computers (Linux and UNIX)
This task describes the steps required to verify port range availability on participating computers. The port range is used by the Fast Communications Manager (FCM). FCM is a feature of Db2 that handles communications between database partition servers.

Before you begin

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

When you install the instance-owning database partition server on the primary computer, Db2 reserves a port range according to the specified number of logical database partition servers participating in

partitioned database environment. The default range is four ports. For each server that participates in the partitioned database environment, you must manually configure the `/etc/services` file for the FCM ports. The range of the FCM ports depends on how many logical partitions you want to use on the participating computer. A minimum of two entries are required, DB2_*instance* and DB2_*instance*_END. Other requirements for the FCM ports specified on participating computers are:

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

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

Procedure

To verify the port range availability on participating computers:

1. Open the `services` file located in the `/etc/services` directory.
2. Locate the ports reserved for the Db2 Fast Communications Manager (FCM). The entries should appear similar to the following example:

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

   Db2 reserves the first four available ports after 60000.
3. On each participating computer, open the `services` file and verify that the ports reserved for Db2 FCM in the services file of the primary computer are not being used.
4. In the event that the required ports are in use on a participating computer, identify an available port range for all computers and update each service file, including the services file on the primary computer.

What to do next

After you install the instance-owning database partition server on the primary computer, you must install your Db2 database product on the participating database partition servers. You can use the response file generated for the partitioning servers (default name is `db2ese_addpart.rsp`), you need to manually configure the `/etc/services` files for the FCM ports. The range of the FCM ports depend on how many logical partitions you want to use on the current machine. The minimum entries are for DB2_ and DB2__END two entries with consecutive free port numbers. The FCM port numbers used on each participating machines must have the same starting port number, and subsequent ports must be sequentially numbered.

Creating a Db2 home file system for a partitioned database system (Linux)

As part of setting up your partitioned database system on Linux operating systems, you need to create a Db2 home file system. Then you must NFS export the home file system and mount it from each computer participating in the partitioned database system.

About this task

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

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

For more information on setting up NFS on Db2 products, see Setting up DB2 for UNIX and Linux on NFS mounted file systems.

For more command information, see your Linux distribution documentation.

Procedure

To create, NFS export, and NFS mount the Db2 home file system, perform the following steps:

1. On one machine, select a disk partition or create one using **fdisk**.
2. Using a utility like **mkfs**, create a file system on this partition.

   The file system should be large enough to contain the necessary Db2 program files as well as enough space for your database needs.
3. Locally mount the file system you have just created and add an entry to the `/etc/fstab` file so that this file system is mounted each time the system is rebooted.
   For example:

   ```
   /dev/hda1    /db2home    ext3   defaults   1 2
   ```
4. To automatically export an NFS file system on Linux at boot time, add an entry to the `/etc/exports` file.

   Be sure to include all of the host names participating in the cluster as well as all of the names that a machine might be known as. Also, ensure that each machine in the cluster has root authority on the exported file system by using the "`root`" option.

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

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

   To export the NFS directory, run

   ```
   /usr/sbin/exportfs -r
   ```
5. On each of the remaining machines in the cluster, add an entry to the `/etc/fstab` file to NFS mount the file system automatically at boot time.

   As in the following example, when you specify the mount point options, ensure that the file system is mounted at boot time, is read-write, is mounted hard, includes the bg (background) option, and that **setuid** programs can be run properly.

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

   where *fusion-en* represents the machine name.
6. NFS mount the exported file system on each of the remaining machines in the cluster.
   Enter the following command:

   ```
   mount /db2home
   ```

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

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

   where *fusion-en* represents the machine name.

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

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

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

Results

By performing these steps, you have completed the following tasks:

1. On a single computer in the partitioned database environment, you have created a file system to be used as the instance and home directory.
2. If you have a configuration that uses more than one machine for a single database instance, you have exported this file system using NFS.
3. You have mounted the exported file system on each participating computer.

Creating a Db2 home file system for a partitioned database system (AIX)

As part of setting up your partitioned database system, you need to create a Db2 home file system. Then you must NFS export the home file system and mount it from each computer participating in the partitioned database system.

Before you begin

It is recommended that you create a home file system that is as large as the content on the Db2 database product DVD. You can use the following command to check the size, KB:

```
du -sk DVD_mounting_point
```

A Db2 instance will require at least 200 MB of space. If you do not have enough free space, you can mount the Db2 database product DVD from each participating computer as an alternative to copying the contents to disk.

You must have:

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

Procedure

To create, NFS export, and NFS mount the Db2 home file system, perform the following steps:

1. Create the Db2 home file system.

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

   a) Enter the **`smit jfs`** command.

   b) Click on the **Add a Journaled File System** icon.

   c) Click on the **Add a Standard Journaled File System** icon.

   d) Select an existing volume group from the **Volume Group Name** list where you want this file system to physically reside.

   e) Set the SIZE of file system (**SIZE of file system (in 512-byte blocks) (Num.)** field).

   This sizing is enumerated in 512-byte blocks, so if you only need to create a file system for the instance home directory, you can use 180 000, which is about 90 MB. If you need to copy the product DVD image over to run the installation, you can create it with a value of 2 000 000, which is about 1 GB.

   f) Enter the mount point for this file system in the **MOUNT POINT** field. In this example, the mount point is `/db2home`.

   g) Set the **Mount AUTOMATICALLY at system restart** field to `yes`.

   The remaining fields can be left to the default settings.

   h) Click **OK**.

2. Export the Db2 home file system.

NFS export the /db2home file system so that it is available to all of the computers that will participate in your partitioned database system.

a) Enter the **smit nfs** command.

b) Click on the **Network File System (NFS)** icon.

c) Click on the **Add a Directory to Exports List** icon.

d) Enter the path name and directory to export (for example, /db2home) in the **PATHNAME of directory to export** field.

e) Enter the name of each workstation that will participate in your partitioned database system in the **HOSTS allowed root access** field.

Use a comma (,) as the delimiter between each name. For example, ServerA, ServerB, ServerC. If you are using a high speed interconnect, it is recommended that you specify the high speed interconnect names for each workstation in this field as well. The remaining fields can be left to the default settings.

f) Click **OK**.

3. Log out.

4. Mount the Db2 home file system from each participating computer.

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

a) Enter the **smit nfs** command.

b) Click on the **Network File System (NFS)** icon.

c) Click on the **Add a File System for Mounting** icon.

d) Enter the path name of the mount point in the **PATHNAME of the mount point (Path)** field.

The path name of the mount point is where you should create the Db2 home directory. For this example, use/db2home.

e) Enter the path name of the remote directory in the **PATHNAME of the remote directory** field.

For this example, you should enter the same value that you entered in the **PATHNAME of the mount point (Path)** field.

f) Enter the *hostname* of the machine where you exported the file system in the **HOST where the remote directory resides** field.

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

To improve performance, you may want to NFS mount the file system that you created over a high speed interconnect. If you want to mount this file system using a high speed interconnect, you must enter its name in the **HOST where remote directory resides** field.

You should be aware that if the high speed interconnect ever becomes unavailable for some reason, every workstation that participates in your partitioned database system will lose access to the Db2 home directory.

g) Set the **MOUNT now, add entry to /etc/filesystems or both?** field to both.

h) Set the **/etc/filesystems entry will mount the directory on system RESTART** field to yes.

i) Set the **MODE for this NFS file system** field to read-write.

j) Set the **Mount file system soft or hard** field to hard.

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

The remaining fields can be left to the default settings.

k) Ensure that this file system is mounted with the **Allow execution of SUID and sgid programs in this file system?** field set to `Yes`.

   This is the default setting.

l) Click **OK**.

m) Log out.

Creating required users for a Db2 server installation in a partitioned database environment (Linux)
Three users and groups are required to operate Db2 databases in partitioned database environments on Linux operating systems.

Before you begin

- You must have root user authority to create users and groups.
- If you manage users and groups with security software, additional steps might be required when defining Db2 users and groups.

About this task

The user and group names used in the following instructions are documented in the following table. You can specify your own user and group names if they adhere to your system naming rules and Db2 naming rules.

If you are planning to use the **Db2 Setup** wizard to install your Db2 database product, the **Db2 Setup** wizard will create these users for you.

Table 10. Required users and groups

Required user	User name	Group name
Instance owner	db2inst1	db2iadm1
Fenced user	db2fenc1	db2fadm1
Db2 administration server user	dasusr1	dasadm1

If the Db2 administration server user is an existing user, this user must exist on all the participating computers before the installation. If you use the **Db2 Setup** wizard to create a new user for the Db2 administration server on the instance-owning computer, then the new user is also created (if necessary) during the response file installations on the participating computers. If the user already exists on the participating computers, the user must have the same primary group.

Restrictions

The user names you create must conform to both your operating system's naming rules, and those of the Db2 database system.

Procedure

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

1. Log on to the primary computer.
2. Create a group for the instance owner (for example, `db2iadm1`), the group that will run UDFs or stored procedures (for example, `db2fadm1`), and the group that will own the Db2 administration server (for example, `dasadm1`) by entering the following commands:

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

   Ensure that the specific numbers you are using do not currently exist on any of the machines.

3. Create a user that belongs to each group that you created in the previous step using the following commands. The home directory for each user will be the Db2 home directory that you previously created and shared (db2home).

```
useradd -u 1004 -g db2iadm1 -m -d /db2home/db2inst1 db2inst1
useradd -u 1003 -g db2fadm1 -m -d /db2home/db2fenc1 db2fenc1
useradd -u 1002 -g dasadm1 -m -d /home/dasusr1 dasusr1
```

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

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

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

Creating required users for a Db2 server installation in a partitioned database environment (AIX)
Three users and groups are required to operate Db2 databases in partitioned database environments on AIX operating systems.

Before you begin

- You must have root user authority to create users and groups.
- If you manage users and groups with security software, additional steps might be required when defining Db2 users and groups.

About this task

The user and group names used in the following instructions are documented in the following table. You can specify your own user and group names if they adhere to your system naming rules and Db2 naming rules.

If you are planning to use the **Db2 Setup** wizard to install your Db2 database product, the **Db2 Setup** wizard will create these users for you.

Table 11. Required users and groups

Required user	User name	Group name
Instance owner	db2inst1	db2iadm1
Fenced user	db2fenc1	db2fadm1
Db2 administration server user	dasusr1	dasadm1

If the Db2 administration server user is an existing user, this user must exist on all the participating computers before the installation. If you use the **Db2 Setup** wizard to create a new user for the Db2 administration server on the instance-owning computer, then the new user is also created (if necessary) during the response file installations on the participating computers. If the user already exists on the participating computers, the user must have the same primary group.

Restrictions

The user names you create must conform to both your operating system's naming rules, and those of the Db2 database system.

Procedure

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

1. Log on to the primary computer.
2. Create a group for the instance owner (for example, db2iadm1), the group that will run UDFs or stored procedures (for example, db2fadm1), and the group that will own the Db2 administration server (for example, dasadm1) by entering the following commands:

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

3. Create a user that belongs to each group that you created in the previous step using the following commands. The home directory for each user will be the Db2 home directory that you previously created and shared (db2home).

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

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

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

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

Installing your DB2 server product

Setting up a partitioned database environment

This topic describes how to set up a partitioned database environment. You will use the **Db2 Setup** wizard to install your instance-owning database server and to create the response files that will in turn be used to create your participating database servers.

Before you begin

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

- Ensure that you have the Db2 Warehouse Activation CD license key that will need to be copied over to all participating computers.
- The same number of consecutive ports must be free on each computer that is to participate in the partitioned database environment. For example, if the partitioned database environment will be comprised of four computers, then each of the four computers must have the same four consecutive ports free. During instance creation, a number of ports equal to the number of logical partitions on the current server will be reserved in the /etc/services on Linux and UNIX and in the %SystemRoot%\system32\drivers\etc\services on Windows. These ports will be used by the Fast Communication Manager. The reserved ports will be in the following format:

```
DB2_InstanceName
DB2_InstanceName_1
```

```
DB2_InstanceName_2
DB2_InstanceName_END
```

The only mandatory entries are the beginning (DB2_InstanceName) and ending (DB2_InstanceName_END) ports. The other entries are reserved in the services file so that other applications do not use these ports

- To support multiple participating Db2 database servers, the computer on which you want to install Db2 must belong to an accessible domain. However, you can add local partitions to this computer even though the computer doesn't belong to a domain.
- On Linux and UNIX systems, a remote shell utility is required for partitioned database systems. Db2 database systems support the following remote shell utilities:
  - rsh
  - ssh

  By default, Db2 database systems use rsh when executing commands on remote Db2 nodes, for example, when starting a remote Db2 database partition.

  To use the Db2 default, the rsh-server package must be installed. For more information, see "Security considerations when installing and using the Db2 database manager" in *Database Security Guide*.

  If you choose to use the rsh remote shell utility, inetd (or xinetd) must be installed and running as well. If you choose to use the ssh remote shell utility, you need to set the **DB2RSHCMD** registry variable immediately after the Db2 installation is complete. If this registry variable is not set, rsh is used.
- On Linux and UNIX operating systems, ensure the `hosts` file under the `etc` directory does not contain an entry for "127.0.0.2" if that IP address maps to the fully qualified hostname of the machine.

About this task

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

Procedure

To set up a partitioned database environment:

1. Install your instance-owning database server using the **Db2 Setup** wizard. For detailed instructions, see the appropriate "Installing Db2 servers" topic for your platform.
   - On the **Select installation, response files creation, or both** window, ensure that you select the **Save my installation settings in a response files** option. After the installation has completed, two files will be copied to the directory specified in the **Db2 Setup** wizard: PROD_ESE.rsp and PROD_ESE_addpart.rsp. The PROD_ESE.rsp file is the response file for instance-owning database servers. The PROD_ESE_addpart.rsp file is the response file for participating database servers.
   - On the **Set up partitioning options for the Db2 instance** window, ensure that you select **Multiple partition instance**, and enter the maximum number of logical partitions.
2. Make the Db2 install image available to all participating computers in the partitioned database environment.
3. Distribute the participating database servers response file (PROD_ESE_addpart.rsp).
4. Install a Db2 database server on each of the participating computers using the **db2setup** command on Linux and UNIX, or the **setup** command on Windows:

   Linux and UNIX

   Go to the directory where the Db2 database product code is available and run:

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

Windows

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

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

Linux and UNIX

Go to the directory where the Db2 database product code is available and run:

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

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

Windows

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

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

5. (Linux and UNIX only) Configure the db2nodes.cfg file. The Db2 installation only reserves the maximum number of logical partitions you want to use for the current computer, but does not configure the db2nodes.cfg file. If you do not configure the db2nodes.cfg file, the instance is still a single partitioned instance.
6. Update the services file on the participating servers to define the corresponding FCM port for the Db2 instance.

   The services file is in the following location:

   - /etc/services on Linux and UNIX
   - %SystemRoot%\system32\drivers\etc\services on Windows
7. For partitioned database environments on Windows 2000 or later, start the Db2 Remote Command Service security feature to protect your data and resources.

   To be fully secure, start either the computer (if the service is running under the context of the LocalSystem account) or a user for delegation (if the service is being run under the logon context of a user).

   To start the Db2 Remote Command Service security feature:

   a) Open the **Active Directory Users and Computers** window on the domain controller, click **Start** and select **Programs** > **Administrative tools** > **Active Directory Users and Computers**

   b) In the right window panel, right-click the computer or user to start, select **Properties**

   c) Click the **General** tab and select the **Trust computer for delegation** check box. For user setting, click the **Account** tab and select the **Account is trusted for delegation** check box in the **Account option** group. Ensure that the **Account is sensitive and cannot be delegated** box has not been checked.

   d) Click **OK** to start the computer or user for delegation.

   Repeat these steps for each computer or user that needs to be started. You must restart your computer for the security change to take effect.

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

In this task you will use the response file you created using the **Db2 Setup** wizard to install database partition servers on participating computers.

Before you begin

- You have installed a Db2 copy on the primary computer using the **Db2 Setup** wizard.
- You have created a response file for installing on participating computers and copied it onto the participating computer.

- You must have administrative authority on participating computers.

Procedure

To install additional database partition servers using a response file:

1. Log to the computer that will participate in the partitioned database environment with the local Administrator account that you have defined for the Db2 installation.
2. Change to the directory containing the Db2 database product DVD.
   For example:

   ```
   cd c:\db2dvd
   ```

   where db2dvd represents the name of the directory containing the Db2 database product DVD.
3. From a command prompt, enter the **setup** command as follows:

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

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

   ```
   setup -u c:\reponsefile\Addpart.file
   ```

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

   ```
   === Logging stopped: 5/9/2007  10:41:32 ===
   MSI (c) (C0:A8) [10:41:32:984]: Product: Db2 Enterprise Server
   Edition - DB2COPY1 -- Installation
   operation completed successfully.
   ```

5. When you install the instance-owning database partition server on the primary computer, the Db2 database product reserves a port range according to the specified number of logical database partition servers participating in partitioned database environment. The default range is four ports. For each server that participates in the partitioned database environment, you must manually configure the /etc/services file for the FCM ports. The range of the FCM ports depends on how many logical partitions you want to use on the participating computer. A minimum of two entries are required, DB2_*instance* and DB2_*instance*_END. Other requirements for the FCM ports specified on participating computers are:
   - The starting port number must match the starting port number of the primary computer.
   - Subsequent ports must be sequentially numbered.
   - Specified port numbers must be free.

Results

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

What to do next

If you want your Db2 database product to have access to Db2 documentation either on your local computer or on another computer on your network, then you must install the *Db2 Information Center*. The *Db2 Information Center* contains documentation for the Db2 database system and Db2 related products.

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

In this task you will use the response file you created using the **Db2 Setup** wizard to install database partition servers on participating computers.

Before you begin

- You have installed Db2 database product on the primary computer using the **Db2 Setup** wizard and have created a response file for installing on participating computers.
- You must have root user authority on participating computers.

Procedure

To install additional database partition servers using a response file:

1. As root, log on to a computer that will participate in the partitioned database environment.
2. Change to the directory where you copied the contents of the Db2 database product DVD. For example:

   ```
   cd /db2home/db2dvd
   ```

3. Enter the **db2setup** command as follows:

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

   In this example, the response file, `AddPartitionResponse.file`, was saved to the `/db2home` directory. The command for this situation would be:

   ```
   ./db2setup -r /db2home/AddPartitionResponse.file
   ```

4. Check the messages in the log file when the installation finishes.

Results

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

What to do next

If you want your Db2 database product to have access to Db2 database documentation either on your local computer or on another computer on your network, then you must install the *Db2 Information Center*. The *Db2 Information Center* contains documentation for the Db2 database system and Db2 database related products.

After you install

Verifying the installation

Verifying a partitioned database environment installation (Windows)

To verify that your Db2 database server installation was successful, you will create a sample database and run SQL commands to retrieve sample data and to verify that the data has been distributed to all participating database partition servers.

Before you begin

You have completed all of the installation steps.

Procedure

To create the SAMPLE database:

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

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

   The SAMPLE database is automatically cataloged with the database alias SAMPLE when it is created.
3. Start the database manager by entering the **db2start** command.
4. Enter the following Db2 commands from a Db2 command window to connect to the SAMPLE database, retrieve a list of all the employees that work in department 20:

   ```
   db2 connect to sample
   db2 "select * from staff where dept = 20"
   ```
5. To verify that data has been distributed across database partition servers, enter the following commands from a Db2 command window:

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

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

What to do next

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

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

Verifying a partitioned database server installation (Linux and UNIX)
Use the **db2val** tool to verify the core functions of a Db2 copy by validating installation files, instances, database creation, connections to that database, and the state of partitioned database environments. For details, see "Validating your Db2 copy". The state of a partitioned database environment is only verified if there are at least 2 nodes. In addition, to verify that your Db2 database server installation was successful, you will create a sample database and run SQL commands to retrieve sample data and to verify that the data has been distributed to all participating database partition servers.

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

Procedure

To create the SAMPLE database:

1. Log on to the primary computer (ServerA) as the instance-owning user.
   For this example, db2inst1 is the instance-owning user.
2. Enter the **db2sampl** command to create the SAMPLE database.

   By default, the sample database will be created in the instance-owner's home directory. In our example `/db2home/db2inst1/` is the instance owner's home directory. The instance owner's home directory is the default database path.

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

   The SAMPLE database is automatically cataloged with the database alias SAMPLE when it is created.
3. Start the database manager by entering the **db2start** command.
4. Enter the following Db2 commands from a Db2 command window to connect to the SAMPLE database, retrieve a list of all the employees that work in department 20:

```
db2 connect to sample
db2 "select * from staff where dept = 20"
```

5. To verify that data has been distributed across database partition servers, enter the following commands from a Db2 command window:

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

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

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

What to do next

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

Chapter 3. Implementation and maintenance

Before creating a database

Setting up partitioned database environments

The decision to create a multi-partition database must be made before you create your database. As part of the database design decisions you make, you will have to determine if you should take advantage of the performance improvements database partitioning can offer.

About this task

In a partitioned database environment, you still use the **CREATE DATABASE** command or the sqlecrea() function to create a database. Whichever method is used, the request can be made through any of the partitions listed in the `db2nodes.cfg` file. The `db2nodes.cfg` file is the database partition server configuration file.

Except on the Windows operating system environment, any editor can be used to view and update the contents of the database partition server configuration file (`db2nodes.cfg`). On the Windows operating system environment, use **db2ncrt** and **db2nchg** commands to create and change the database partition server configuration file

Before creating a multi-partition database, you must select which database partition will be the catalog partition for the database. You can then create the database directly from that database partition, or from a remote client that is attached to that database partition. The database partition to which you attach and execute the **CREATE DATABASE** command becomes the *catalog partition* for that particular database.

The catalog partition is the database partition on which all system catalog tables are stored. All access to system tables must go through this database partition. All federated database objects (for example, wrappers, servers, and nicknames) are stored in the system catalog tables at this database partition.

If possible, you should create each database in a separate instance. If this is not possible (that is, you must create more than one database per instance), you should spread the catalog partitions among the available database partitions. Doing this reduces contention for catalog information at a single database partition.

Note: You should regularly do a backup of the catalog partition and avoid putting user data on it (whenever possible), because other data increases the time required for the backup.

When you create a database, it is automatically created across all the database partitions defined in the `db2nodes.cfg` file.

When the first database in the system is created, a system database directory is formed. It is appended with information about any other databases that you create. When working on UNIX, the system database directory is `sqldbdir` and is located in the `sqllib` directory under your home directory, or under the directory where Db2 database was installed. When working on UNIX, this directory must reside on a shared file system, (for example, NFS on UNIX platforms) because there is only one system database directory for all the database partitions that make up the partitioned database environment. When working on Windows, the system database directory is located in the instance directory.

Also resident in the `sqldbdir` directory is the system intention file. It is called `sqldbins`, and ensures that the database partitions remain synchronized. The file must also reside on a shared file system since there is only one directory across all database partitions. The file is shared by all the database partitions making up the database.

Configuration parameters have to be modified to take advantage of database partitioning. Use the **GET DATABASE CONFIGURATION** and the **GET DATABASE MANAGER CONFIGURATION** commands to find out the values of individual entries in a specific database, or in the database manager configuration file.

To modify individual entries in a specific database, or in the database manager configuration file, use the **UPDATE DATABASE CONFIGURATION** and the **UPDATE DATABASE MANAGER CONFIGURATION** commands respectively.

The database manager configuration parameters affecting a partitioned database environment include **conn_elapse**, **fcm_num_buffers**, **fcm_num_channels**, **max_connretries**, **max_coordagents**, **max_time_diff**, **num_poolagents**, and **start_stop_time**.

Creating node configuration files

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

About this task

To enable database partitioning, the db2nodes.cfg file must be located in the sqllib subdirectory of the home directory for the instance before you start the database manager. This file contains configuration information for all database partitions in an instance, and is shared by all database partitions for that instance.

Windows considerations

If you are using Db2 Enterprise Server Edition on Windows, the node configuration file is created for you when you create the instance. You should not attempt to create or modify the node configuration file manually. You can use the **db2ncrt** command to add a database partition server to an instance. You can use the **db2ndrop** command to drop a database partition server from an instance. You can use the **db2nchg** command to modify a database partition server configuration including moving the database partition server from one computer to another; changing the TCP/IP host name; or, selecting a different logical port or network name.

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

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

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

Tokens are delimited by blanks. The variables are:

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

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

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

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

```
NODE nnnn
```

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

hostname
The host name of the IP address for inter-partition communications. Use the fully-qualified name for the host name. The /etc/hosts file also should use the fully-qualified name. If the fully-qualified

name is not used in the db2nodes.cfg file and in the /etc/hosts file, you might receive error message SQL30082N RC=3.

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

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

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

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

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

The following example shows a possible node configuration file for a system on which SP2EN1 has multiple TCP/IP interfaces, two logical partitions, and uses SP2SW1 as the Db2 database interface. It also shows the database partition numbers starting at 1 (rather than at 0), and a gap in the *dbpartitionnum* sequence:

Table 12. Database partition number example table.

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

You can update the db2nodes.cfg file using an editor of your choice. (The exception is: an editor should not be used on Windows.) You must be careful, however, to protect the integrity of the information in the file, as database partitioning requires that the node configuration file is locked when you issue **START DBM** and unlocked after **STOP DBM** ends the database manager. The **START DBM** command can update the file, if necessary, when the file is locked. For example, you can issue **START DBM** with the **RESTART** option or the **ADD DBPARTITIONNUM** option.

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

Format of the Db2 node configuration file
The db2nodes.cfg file is used to define the database partition servers that participate in a Db2 instance. The db2nodes.cfg file is also used to specify the IP address or host name of a high-speed interconnect, if you want to use a high-speed interconnect for database partition server communication.

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

```
dbpartitionnum hostname logicalport netname resourcesetname
```

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

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

```
dbpartitionnum hostname computername logicalport netname resourcesetname
```

On Windows operating systems, these entries to the db2nodes.cfg are added by the **db2ncrt** or START DBM ADD DBPARTITIONNUM commands. The entries can also be modified by the **db2nchg** command. You should not add these lines directly or edit this file.

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

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

This entry is required.

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

When the system has more than one network interface card installed and the hostname that is used in the db2nodes.cfg file cannot be resolved to be the default host of the system, it might be treated as a remote host. This setup imposes a limitation that database migration cannot be done successfully because the local database directory cannot be found if the instance is not started. Therefore, HADR might require the hostname to match the name that is used by the operating system to identify the host to make migration possible. In addition to this, the operating system name of the host must be specified in db2nodes.cfg when it is running in a Tivoli SA MP, PowerHA® SystemMirror®, and other high availability environments, including the Db2 fault monitor.

Starting with Db2 Version 9.1, both TCP/IPv4 and TCP/IPv6 protocols are supported. The method to resolve host names has changed.

While the method used in pre-Version 9.1 releases resolves the string as defined in the db2nodes.cfg file, the method in Version 9.1 or later tries to resolve the Fully Qualified Domain Names (FQDN) when short names are defined in the db2nodes.cfg file. Specifying short configured for fully qualified host names, this may lead to unnecessary delays in processes that resolve host names.

To avoid any delays in Db2 commands that require host name resolution, use any of the following workarounds:

1. If short names are specified in the db2nodes.cfg files and the operating system host name file, specify the short name and the fully qualified domain name for host name in the operating system host files.
2. To use only IPv4 addresses when you know that the Db2 server listens on an IPv4 port, issue the following command:

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

3. To use only IPv6 addresses when you know that the Db2 server listens on an IPv6 port, issue the following command:

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

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

Db2 reserves a port range (for example, 60000 - 60003) in the `/etc/services` file for interpartition communications at the time of installation. This *logicalport* field in `db2nodes.cfg` specifies which port in that range you want to assign to a particular logical partition server.

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

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

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

Each physical server must have a logical node `0`.

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

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

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

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

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

This parameter is only supported on AIX.

On AIX, this concept is known as "resource sets", refer to AIX documentation for more information about resource management.

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

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

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

Example configurations

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

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

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

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

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

Two computers, three database partition server on one computer

If you want your partitioned database system to contain two physical workstations, called ServerA and ServerB, and ServerA is running 3 database partition servers, update the db2nodes.cfg file as follows:

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

Two computers, three database partition servers with high speed switches

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

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

Examples using resourcesetname

These restrictions apply to the following examples:

- This example shows the usage of *resourcesetname* when there is no high speed interconnect in the configuration.
- The *netname* is the fourth column and a *hostname* also can be specified on that column where there is no switch name and you want to use *resourcesetname*. The fifth parameter is *resourcesetname* if it is defined. The resource group specification can only show as the fifth column in the db2nodes.cfg file. This means that for you to specify a resource group, you must also enter a fourth column. The fourth column is intended for a high speed switch.
- If you do not have a high speed switch or you do not want to use it, you must then enter the *hostname* (same as the second column). In other words, the Db2 database management system does not support column gaps (or interchanging them) in the db2nodes.cfg files. This restriction already applies to the first three columns, and now it applies to all five columns.

AIX example

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

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

1. Define resource sets by using the AIX mkrset command:

```
mkrset -c 0 1 2 3 DB2/MLN1
mkrset -c 4 5 6 7 DB2/MLN2
mkrset -c 8 9 10 11 DB2/MLN3
mkrset -c 12 13 14 15 DB2/MLN4
mkrset -c 16 17 18 19 DB2/MLN5
mkrset -c 20 21 22 23 DB2/MLN6
mkrset -c 24 25 26 27 DB2/MLN7
mkrset -c 28 29 30 31 DB2/MLN8
```

2. Give instance permissions to use resource sets:

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

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

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

```
7 regatta 6 regatta DB2/MLN7
8 regatta 7 regatta DB2/MLN8
```

Linux example

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

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

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

   ```
   $ numactl --hardware
   ```

   Output similar to the following displays:

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

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

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

Specifying the list of machines in a partitioned database environment

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

About this task

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

Procedure

- To override the list of computers in db2nodes.cfg:
  - Specify a path name to the file that contains the list of computers by exporting (on Linux and UNIX operating systems) or setting (on Windows) the environment variable **RAHOSTFILE**.
  - Specify the list explicitly, as a string of names separated by spaces, by exporting (on Linux and UNIX operating systems) or setting (on Windows) the environment variable **RAHOSTLIST**.

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

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

If you are running multiple logical database partition servers on one computer, your db2nodes.cfg file contains multiple entries for that computer.

About this task

In this situation, the **rah** command needs to know whether you want the command to be executed only once on each computer or once for each logical database partition listed in the db2nodes.cfg file. Use

the **rah** command to specify computers. Use the **db2_all** command to specify logical database partitions.

Note: On Linux and UNIX operating systems, if you specify computers, **rah** normally eliminates duplicates from the computer list, with the following exception: if you specify logical database partitions, **db2_all** prepends the following assignment to your command:

```
export DB2NODE=nnn  (for Korn shell syntax)
```

where *nnn* is the database partition number taken from the corresponding line in the db2nodes.cfg file, so that the command is routed to the desired database partition server.

When specifying logical database partitions, you can restrict the list to include all logical database partitions except one, or specify only one using the <<-*nnn*< and <<+*nnn*< prefix sequences. You might want to do this if you want to run a command to catalog the database partition first, and when that has completed, run the same command at all other database partition servers, possibly in parallel. This is usually required when running the **RESTART DATABASE** command. You need to know the database partition number of the catalog partition to do this.

If you execute **RESTART DATABASE** using the **rah** command, duplicate entries are eliminated from the list of computers. However if you specify the " prefix, then duplicates are not eliminated, because it is assumed that use of the " prefix implies sending to each database partition server, rather than to each computer.

Updating the node configuration file (Linux and UNIX)
In a Db2 partitioned database environment, this task provides steps for updating the db2nodes.cfg file to include entries for participating computers.

Before you begin

- The Db2 database product must be installed on all participating computers.
- A Db2 instance must exist on the primary computer.
- You must be a user with SYSADM authority.
- Review the configuration examples and file format information provided in the *Format of the Db2 node configuration file* topic if either of the following conditions apply:
  - You plan to use a high speed switch for communication between database partition servers
  - Your partitioned configuration will have multiple logical partitions

About this task

The node configuration file (db2nodes.cfg), located in the instance owner's home directory, contains configuration information that tells the Db2 database system which servers participate in an instance of the partitioned database environment. A db2nodes.cfg file exists for each instance in a partitioned database environment.

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

For example, when you created the Db2 instance using the **Db2 Setup** wizard, on the instance-owning server ServerA, the db2nodes.cfg file is updated as follows:

```
     0     ServerA     0
```

Restrictions

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

Procedure

To update the db2nodes.cfg file:

1. Log on as the instance owner.

   For example, db2inst1 is the instance owner in these steps.
2. Ensure that the Db2 instance is stopped by entering:

   ```
   INSTHOME/sqllib/adm/db2stop
   ```

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

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

   ```
   /db2home/db2inst1/sqllib/adm/db2stop
   ```
3. Add an entry to the .rhosts file for each Db2 instance.
   Update the file by adding the following:

   ```
   hostname db2instance
   ```

   where *hostname* is the TCP/IP host name of the database server and *db2instance* is the name of the instance you use to access the database server.
4. Add an entry to the db2nodes.cfg file of each participating server.

   When you first view the db2nodes.cfg file, it should contain an entry similar to the following:

   ```
   0     ServerA     0
   ```

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

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

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

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

Setting up multiple logical partitions

There are several situations in which it is advantageous to have several database partition servers running on the same computer.

This means that the configuration can contain more database partitions than computers. In these cases, the computer is said to be running *multiple logical partitions* if they participate in the same instance. If they participate in different instances, this computer is not hosting multiple logical partitions.

With multiple logical partition support, you can choose from three types of configurations:

- A standard configuration, where each computer has only one database partition server
- A multiple logical partition configuration, where a computer has more than one database partition server
- A configuration where several logical partitions run on each of several computers

Configurations that use multiple logical partitions are useful when the system runs queries on a computer that has symmetric multiprocessor (SMP) architecture. The ability to configure multiple logical partitions on a computer is also useful if a computer fails. If a computer fails (causing the database partition server or servers on it to fail), you can restart the database partition server (or servers) on another computer using the **START DBM DBPARTITIONNUM** command. This ensures that user data remains available.

Another benefit is that multiple logical partitions can use SMP hardware configurations. In addition, because database partitions are smaller, you can obtain better performance when performing such tasks as backing up and restoring database partitions and table spaces, and creating indexes.

Configuring multiple logical partitions
There are two methods of configuring multiple logical partitions.

About this task

- Configure the logical partitions (database partitions) in the db2nodes.cfg file. You can then start all the logical and remote partitions with the **db2start** command or its associated API.

  Note: For Windows, you must use **db2ncrt** to add a database partition if there is no database in the system; or, **db2start addnode** command if there is one or more databases. Within Windows, the db2nodes.cfg file should never be manually edited.
- Restart a logical partition on another processor on which other logical partitions are already running. This allows you to override the hostname and port number specified for the logical partition in db2nodes.cfg.

To configure a logical database partition in db2nodes.cfg, you must make an entry in the file to allocate a logical port number for the database partition. Following is the syntax you should use:

```
nodenumber hostname logical-port netname
```

Note: For Windows, you must use **db2ncrt** to add a database partition if there is no database in the system; or, **db2start addnode** command if there is one or more databases. Within Windows, the db2nodes.cfg file should never be manually edited.

The format for the db2nodes.cfg file on Windows is different when compared to the same file on UNIX. On Windows, the column format is:

```
nodenumber hostname computername logical_port netname
```

Use the fully-qualified name for the hostname. The /etc/hosts file also should use the fully-qualified name. If the fully-qualified name is not used in the db2nodes.cfg file and in the/etc/hosts file, you might receive error message SQL30082N RC=3.

You must ensure that you define enough ports in the services file of the etc directory for FCM communications.

Enabling inter-partition query parallelism

Inter-partition parallelism occurs automatically based on the number of database partitions and the distribution of data across these database partitions.

About this task

You must modify configuration parameters to take advantage of parallelism within a database partition or within a non-partitioned database. For example, intra-partition parallelism can be used to take advantage of the multiple processors on a symmetric multi-processor (SMP) machine.

Procedure

- To enable parallelism when loading data:

The load utility automatically makes use of parallelism, or you can use the following parameters on the **LOAD** command:

- **CPU_PARALLELISM**
- **DISK_PARALLELISM**

In a partitioned database environment, inter-partition parallelism for data loading occurs automatically when the target table is defined on multiple database partitions. Inter-partition parallelism for data loading can be overridden by specifying **OUTPUT_DBPARTNUMS**. The load utility also intelligently enables database partitioning parallelism depending on the size of the target database partitions. **MAX_NUM_PART_AGENTS** can be used to control the maximum degree of parallelism selected by the load utility. Database partitioning parallelism can be overridden by specifying **PARTITIONING_DBPARTNUMS** when **ANYORDER** is also specified.

- To enable parallelism when creating an index:
  - The table must be large enough to benefit from parallelism
  - Multiple processors must be enabled on an SMP computer.
- To enable I/O parallelism when backing up a database or table space:
  - Use more than one target media.
  - Configure table spaces for parallel I/O by defining multiple containers, or use a single container with multiple disks, and the appropriate use of the **DB2_PARALLEL_IO** registry variable. If you want to take advantage of parallel I/O, you must consider the implications of what must be done before you define any containers. This cannot be done whenever you see a need; it must be planned for before you reach the point where you need to backup your database or table space.
  - Use the **PARALLELISM** parameter on the **BACKUP** command to specify the degree of parallelism.
  - Use the **WITH** `num-buffers` **BUFFERS** parameter on the **BACKUP** command to ensure that enough buffers are available to accommodate the degree of parallelism. The number of buffers should equal the number of target media you have plus the degree of parallelism selected plus a few extra.

    Also, use a backup buffer size that is:

    - As large as feasible. 4 MB or 8 MB (1024 or 2048 pages) is a good rule of thumb.
    - At least as large as the largest (extent size * number of containers) product of the table spaces being backed up.
- To enable I/O parallelism when restoring a database or table space:
  - Use more than one source media.
  - Configure table spaces for parallel I/O. You must decide to use this option before you define your containers. This cannot be done whenever you see a need; it must be planned for before you reach the point where you need to restore your database or table space.
  - Use the **PARALLELISM** parameter on the **RESTORE** command to specify the degree of parallelism.
  - Use the **WITH** `num-buffers` **BUFFERS** parameter on the **RESTORE** command to ensure that enough buffers are available to accommodate the degree of parallelism. The number of buffers should equal the number of target media you have plus the degree of parallelism selected plus a few extra.

    Also, use a restore buffer size that is:

    - As large as feasible. 4 MB or 8 MB (1024 or 2048 pages) is a good rule of thumb.
    - At least as large as the largest (extent size * number of containers) product of the table spaces being restored.
    - The same as, or an even multiple of, the backup buffer size.

Enabling intrapartition parallelism for queries

To enable intrapartition query parallelism, modify one or more database or database manager configuration parameters, precompile or bind options, or a special register. Alternatively, use the MAXIMUM DEGREE option on the CREATE or ALTER WORKLOAD statement, or the

ADMIN_SET_INTRA_PARALLEL procedure to enable or disable intrapartition parallelism at the transaction level.

Before you begin

Use the following controls to specify what degree of intrapartition parallelism the optimizer is to use:

- CURRENT DEGREE special register (for dynamic SQL)
- DEGREE bind option (for static SQL)
- **dft_degree** database configuration parameter (provides the default value for the previous two parameters)

Use the following controls to limit the degree of intrapartition parallelism at run time. The runtime settings override the optimizer settings.

- **max_querydegree** database manager configuration parameter
- SET RUNTIME DEGREE command
- MAXIMUM DEGREE workload option
- MAXIMUM DEGREE service class option

Use any of the following controls to enable or disable intrapartition parallelism:

- **intra_parallel** database manager configuration parameter
- ADMIN_SET_INTRA_PARALLEL stored procedure
- MAXIMUM DEGREE workload option

About this task

Use the **GET DATABASE CONFIGURATION** or the **GET DATABASE MANAGER CONFIGURATION** command to find the values of individual entries in a specific database or instance configuration file. To modify one or more of these entries, use the **UPDATE DATABASE CONFIGURATION** or the **UPDATE DATABASE MANAGER CONFIGURATION** command.

intra_parallel
Database manager configuration parameter that specifies whether or not the database manager can use intrapartition parallelism. The default is NO, which means that applications in this instance are run without intrapartition parallelism. For example:

```
update dbm cfg using intra_parallel yes;
get dbm cfg;
```

max_querydegree
Database manager configuration parameter that specifies the maximum degree of intrapartition parallelism that is used for any SQL statement running on this instance. An SQL statement does not use more than this value when running parallel operations within a database partition. The default is -1, which means that the system uses the degree of intrapartition parallelism that is determined by the optimizer, not the user-specified value. For example:

```
update dbm cfg using max_querydegree any;
get dbm cfg;
```

The **intra_parallel** database manager configuration parameter must also be set to YES for the value of **max_querydegree** to be used.

dft_degree
Database configuration parameter that specifies the default value for the **DEGREE** precompile or bind option and the CURRENT DEGREE special register. The default is 1. A value of -1 (or ANY) means that

the system uses the degree of intrapartition parallelism that is determined by the optimizer. For example:

```
connect to sample;
update db cfg using dft_degree -1;
get db cfg;
connect reset;
```

DEGREE

Precompile or bind option that specifies the degree of intrapartition parallelism for the execution of static SQL statements on a symmetric multiprocessing (SMP) system. For example:

```
connect to prod;
prep demoapp.sqc bindfile;
bind demoapp.bnd degree 2;
...
```

CURRENT DEGREE

Special register that specifies the degree of intrapartition parallelism for the execution of dynamic SQL statements. Use the SET CURRENT DEGREE statement to assign a value to the CURRENT DEGREE special register. For example:

```
connect to sample;
set current degree = '1';
connect reset;
```

The **intra_parallel** database manager configuration parameter must also be set to YES to use intrapartition parallelism. If it is set to NO, the value of this special register is ignored, and the statement will not use intrapartition parallelism. The value of the **intra_parallel** database manager configuration parameter and the CURRENT DEGREE special register can be overridden in a workload by setting the MAXIMUM DEGREE workload attribute.

MAXIMUM DEGREE

CREATE WORKLOAD or CREATE SERVICE CLASS statement (or ALTER WORKLOAD or ALTER SERVICE CLASS statement) option that specifies the maximum runtime degree of parallelism for a workload or service class.

For example, suppose that `bank_trans` is a packaged application that mainly executes short OLTP transactions, and `bank_report` is another packaged application that runs complex queries to generate a business intelligence (BI) report. Neither application can be modified, and both are bound with degree 4 to the database. While `bank_trans` is running, it is assigned to workload `trans`, which disables intrapartition parallelism. This OLTP application will run without any performance degradation associated with intrapartition parallelism overhead. While `bank_report` is running, it is assigned to workload `bi`, which enables intrapartition parallelism and specifies a maximum runtime degree of 8. Because the compilation degree for the package is 4, the static SQL statements in this application run with only a degree of 4. If this BI application contains dynamic SQL statements, and the CURRENT DEGREE special register is set to 16, these statements run with a degree of 8.

```
connect to sample;

create workload trans
  applname('bank_trans')
  maximum degree 1
  enable;

create workload bi
  applname('bank_report')
  maximum degree 8
  enable;

connect reset;
```

ADMIN_SET_INTRA_PARALLEL

Procedure that enables or disables intrapartition parallelism for a database application. Although the procedure is called in the current transaction, it takes effect starting with the next transaction. For

example, assume that the following code is part of the demoapp application, which uses the ADMIN_SET_INTRA_PARALLEL procedure with both static and dynamic SQL statements:

```
EXEC SQL CONNECT TO prod;

// Disable intrapartition parallelism:
EXEC SQL CALL SYSPROC.ADMIN_SET_INTRA_PARALLEL('NO');
// Commit so that the effect of this call
// starts in the next statement:
EXEC SQL COMMIT;

// All statements in the next two transactions run
// without intrapartition parallelism:
strcpy(stmt, "SELECT deptname FROM org");
EXEC SQL PREPARE rstmt FROM :stmt;
EXEC SQL DECLARE c1 CURSOR FOR rstmt;
EXEC SQL OPEN c1;
EXEC SQL FETCH c1 INTO :deptname;
EXEC SQL CLOSE c1;
...
// New section for this static statement:
EXEC SQL SELECT COUNT(*) INTO :numRecords FROM org;
...
EXEC SQL COMMIT;

// Enable intrapartition parallelism:
EXEC SQL CALL SYSPROC.ADMIN_SET_INTRA_PARALLEL('YES');
// Commit so that the effect of this call
// starts in the next statement:
EXEC SQL COMMIT;

strcpy(stmt, "SET CURRENT DEGREE='4'");
// Set the degree of parallelism to 4:
EXEC SQL EXECUTE IMMEDIATE :stmt;

// All dynamic statements in the next two transactions
// run with intrapartition parallelism and degree 4:
strcpy(stmt, "SELECT deptname FROM org");
EXEC SQL PREPARE rstmt FROM :stmt;
EXEC SQL DECLARE c2 CURSOR FOR rstmt;
EXEC SQL OPEN c2;
EXEC SQL FETCH c2 INTO :deptname;
EXEC SQL CLOSE c2;
...
// All static statements in the next two transactions
// run with intrapartition parallelism and degree 2:
EXEC SQL SELECT COUNT(*) INTO :numRecords FROM org;
...
EXEC SQL COMMIT;
```

The degree of intrapartition parallelism for dynamic SQL statements is specified through the CURRENT DEGREE special register, and for static SQL statements, it is specified through the DEGREE bind option. The following commands are used to prepare and bind the demoapp application:

```
connect to prod;
prep demoapp.sqc bindfile;
bind demoapp.bnd degree 2;
...
```

Management of data server capacity

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

Adding processors

If a single-partition database configuration with a single processor is used to its maximum capacity, you might either add processors or add logical partitions. The advantage of adding processors is greater processing power. In a single-partition database configuration with multiple processors (SMP), processors share memory and storage system resources. All of the processors are in one system, so

communication between systems and coordination of tasks between systems does not factor into the workload. Utilities such as load, back up, and restore can take advantage of the additional processors.

Note: Some operating systems can dynamically turn processors on- and offline.

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

- Default degree (**dft_degree**)
- Maximum degree of parallelism (**max_querydegree**)
- Enable intrapartition parallelism (**intra_parallel**)

You should also evaluate parameters that determine how applications perform parallel processing.

In an environment where TCP/IP is used for communication, review the value for the **DB2TCPCONNMGRS** registry variable.

Adding additional computers

If you have an existing partitioned database environment, you can increase processing power and data-storage capacity by adding additional computers (either single-processor or multiple-processor) and storage resource to the environment. The memory and storage resources are not shared among computers. This choice provides the advantage of balancing data and user access across storage and computers.

After adding the new computers and storage, you would use the **START DATABASE MANAGER** command to add new database partition servers to the new computers. A new database partition is created and configured for each database in the instance on each new database partition server that you add. In most situations, you do not need to restart the instance after adding the new database partition servers.

Fast communications manager

Fast communications manager (Windows)

In multiple member environments, each member has a pair of FCM daemons to support communication between members that is related to agent requests. One daemon is for sending communications, and the other is for receiving. These daemons and supporting infrastructure are activated when an instance is started. FCM communication is also used for agents working within the same member; this type of communication is also known as intra-member communication.

You can specify the number of FCM message buffers by using the **fcm_num_buffers** database manager configuration parameter. You can specify the number of FCM channels by using the **fcm_num_channels** database manager configuration parameter. By default, the **fcm_num_buffers** and **fcm_num_channels** database manager configuration parameters are set to AUTOMATIC. If the setting is AUTOMATIC, which is the recommended setting, the FCM monitors resource usage and adjusts resources to accommodate workload demand.

Fast communications manager (Linux and UNIX)

The fast communications manager (FCM) provides communications support for partitioned database environments.

In multiple member environments, each member has a pair of FCM daemons to support communication between members that is related to agent requests. One daemon is for sending communications, and the other is for receiving. These daemons and supporting infrastructure are activated when an instance is started. FCM communication is also used for agents working within the same member; this type of communication is also known as intra-member communication.

The FCM daemon collects information about communication activities. You can obtain information about FCM communications by using the database system monitor. If communications fail between members or if they re-establish communications, the FCM daemons update monitor elements with this information. The FCM daemons also trigger the appropriate action for this event. An example of an appropriate action

is the rollback of an affected transaction. You can use the database system monitor to help you set the FCM configuration parameters.

You can specify the number of FCM message buffers by using the **fcm_num_buffers** database manager configuration parameter. You can specify the number of FCM channels by using the **fcm_num_channels** database manager configuration parameter. By default, the **fcm_num_buffers** and **fcm_num_channels** database manager configuration parameters are set to AUTOMATIC. If the setting is AUTOMATIC, which is the recommended setting, the FCM monitors resource usage and adjusts resources to accommodate workload demand.

Enabling communication between database partitions using FCM communications
In a partitioned database environment, most communication between database partitions is handled by the fast communications manager (FCM).

To enable the FCM at a database partition and allow communication with other database partitions, you must create a service entry in the database partition's services file of the etc directory as shown later in this section. The FCM uses the specified port to communicate. If you have defined multiple database partitions on the same host, you must define a range of ports, as shown later in this section.

Before attempting to manually configure memory for the fast communications manager (FCM), it is recommended that you start with the automatic setting, which is also the default setting, for the number of FCM Buffers (**fcm_num_buffers**) and for the number of FCM Channels (**fcm_num_channels**). Use the system monitor data for FCM activity to determine if this setting is appropriate.

Windows Considerations
The TCP/IP port range is automatically added to the services file by:

- The install program when it creates the instance or adds a new database partition
- The **db2icrt** utility when it creates a new instance
- The **db2ncrt** utility when it adds the first database partition on the computer

The syntax of a service entry is as follows:

```
DB2_instance port/tcp #comment
```

DB2_*instance*
The value for *instance* is the name of the database manager instance. All characters in the name must be lowercase. Assuming an instance name of DB2PUSER, you specify DB2_db2puser.

***port*/tcp**
The TCP/IP port that you want to reserve for the database partition.

#comment
Any comment that you want to associate with the entry. The comment must be preceded by a pound sign (#).

If the services file of the etc directory is shared, you must ensure that the number of ports allocated in the file is either greater than or equal to the largest number of multiple database partitions in the instance. When allocating ports, also ensure that you account for any processor that can be used as a backup.

If the services file of the etc directory is not shared, the same considerations apply, with one additional consideration: you must ensure that the entries defined for the Db2 database instance are the same in all services files of the etc directory (though other entries that do not apply to your partitioned database environment do not have to be the same).

If you have multiple database partitions on the same host in an instance, you must define more than one port for the FCM to use. To do this, include two lines in the services file of the etc directory to indicate the range of ports that you are allocating. The first line specifies the first port, and the second line

indicates the end of the block of ports. In the following example, five ports are allocated for the SALES instance. This means no processor in the instance has more than five database partitions. For example:

```
DB2_sales         9000/tcp
DB2_sales_END     9004/tcp
```

Note: You must specify END in uppercase only. You must also ensure that you include both underscore (_) characters.

Enabling communications between database partition servers (Linux and UNIX)
This task describes how to enable communication between the database partition servers that participate in your partitioned database system. Communication between database partition servers is handled by the Fast Communications Manager (FCM). To enable FCM, a port or port range must be reserved in the `/etc/services` file on each computer in your partitioned database system.

Before you begin

You must have a user ID with root user authority.

You must perform this task on all computers that participate in the instance.

About this task

The number of ports to reserve for FCM is equal to the maximum number of database partitions hosted, or potentially hosted, by any computer in the instance.

In the following example, the `db2nodes.cfg` file contains these entries:

```
0 server1 0
1 server1 1
2 server2 0
3 server2 1
4 server2 2
5 server3 0
6 server3 1
7 server3 2
8 server3 3
```

Assume that the FCM ports are numbered starting at 60000. In this situation:

- server1 uses two ports (60000, 60001) for its two database partitions
- server2 uses three ports (60000, 60001, 60002) for its three database partitions
- server3 uses four ports (60000, 60001, 60002, 60003) for its four database partitions

All computers must reserve 60000, 60001, 60002, and 60003, since this is the largest port range required by any computer in the instance.

If you use a high availability solution such as Tivoli System Automation or IBM PowerHA SystemMirror for AIX to fail over database partitions from one computer to another, you must account for potential port requirements. For example, if a computer normally hosts four database partitions, but another computer's two database partitions could potentially fail over to it, six ports must be planned for that computer.

When you create an instance, a port range is reserved on the primary computer. The primary computer is also known as the instance-owning computer. However, if the port range originally added to the `/etc/services` file is not sufficient for your needs, you will need to extend the range of reserved ports by manually adding additional entries.

Procedure

To enable communications between servers in a partitioned database environment using `/etc/services`:

1. Log on to the primary computer (instance owning computer) as a user with root authority.
2. Create an instance.

3. View the default port range that has been reserved in the /etc/services file.
   In addition to the base configuration, the FCM ports should appear similar to the following:

```
db2c_db2inst1         50000/tcp
#Add FCM port information
DB2_db2inst1          60000/tcp
DB2_db2inst1_1        60001/tcp
DB2_db2inst1_2        60002/tcp
DB2_db2inst1_END      60003/tcp
```

   By default, the first port (50000) is reserved for connection requests, and the first available four ports above 60000 are reserved for FCM communication. One port is for the instance-owning database partition server and three ports are for logical database partition servers that you might choose to add to the computer after installation is complete.

   The port range must include a start and an END entry. Intermediate entries are optional. Explicitly including intermediate values can be useful for preventing other applications from using these ports, but these entries are not verified by the database manager.

   Db2 port entries use the following format:

```
DB2_instance_name_suffix  port_number/tcp  # comment
```

   where:

   - *instance_name* is the name of the partitioned instance.
   - *suffix* is not used for the first FCM port. Intermediate entries are those between the lowest and highest port. If you include the intermediate entries between the first and ending FCM port, the *suffix* consists of an integer that you increment by one for each additional port. For example, the second port is numbered 1, and third is numbered 2, and so on to ensure uniqueness. The word END must be used as the *suffix* for the last entry.
   - *port_number* is the port number that you reserve for database partition server communications.
   - *comment* is an optional comment describing an entry.
4. Ensure that there are sufficient ports reserved for FCM communication.
   If the range of reserved ports is insufficient, add new entries to the file.
5. Ensure that none of the ports that are reserved for FCM communication is the same as the port used for the **svcename - TCP/IP service name** configuration parameter.
   For more information about defining ports in the /etc/services file, see http://www.ibm.com/support/docview.wss?uid=swg21386030.
6. Log on as a root user to each computer participating in the instance and add identical entries to the /etc/services file.

Creating and managing partitioned database environments

Managing database partitions

You can start or stop partitions, drop partitions, or trace partitions.

Before you begin

To work with database partitions, you need authority to attach to an instance. Anyone with SECADM or ACCESSCTRL authority can grant you the authority to access a specific instance.

Procedure

- To start or to stop a specific database partition, use the **START DATABASE MANAGER** command or the **STOP DATABASE MANAGER** command with the **DBPARTITIONNUM** parameter.
- To drop a specific database partition from the db2nodes.cfg configuration file, use the **STOP DATABASE MANAGER** command with the **DROP DBPARTITIONNUM** parameter.

Before using the **DROP DBPARTITIONNUM** parameter, run the **DROP DBPARTITIONNUM VERIFY** command to ensure that there is no user data on this database partition.

- To trace the activity on a database partition, use the options specified by IBM Support.

  Attention: Use the trace utility only when directed to do so by IBM Support or by a technical support representative.

  The trace utility records and formats information about Db2 operations. For more details, see the "db2trc - Trace command" topic.

Adding database partitions in partitioned database environments

You can add database partitions to the partitioned database system either when it is running, or when it is stopped. Because adding a new server can be time consuming, you might want to do it when the database manager is already running.

Use the **ADD DBPARTITIONNUM** command to add a database partition to a system. This command can be invoked in the following ways:

- As an option on the **START DBM** command
- With the **ADD DBPARTITIONNUM** command
- With the `sqleaddn` API
- With the `sqlepstart` API

If your system is stopped, use the **START DBM** command. If it is running, you can use any of the other choices.

When you use the **ADD DBPARTITIONNUM** command to add a new database partition to the system, all existing databases in the instance are expanded to the new database partition. You can also specify which containers to use for temporary table spaces for the databases. The containers can be:

- The same as those defined for the catalog partition for each database. (This is the default.)
- The same as those defined for another database partition.
- Not created at all. You must use the ALTER TABLESPACE statement to add temporary table space containers to each database before the database can be used.

Note: Any uncataloged database is not recognized when adding a new database partition. The uncataloged database will not be present on the new database partition. An attempt to connect to the database on the new database partition returns the error message SQL1013N.

You cannot use a database on the new database partition to contain data until one or more database partition groups are altered to include the new database partition.

You cannot change from a single-partition database to a multi-partition database by adding a database partition to your system. This is because the redistribution of data across database partitions requires a distribution key on each affected table. The distribution keys are automatically generated when a table is created in a multi-partition database. In a single-partition database, distribution keys can be explicitly created with the CREATE TABLE or ALTER TABLE SQL statements.

Note: If no databases are defined in the system and you are running Enterprise Server Edition on a UNIX operating system, edit the `db2nodes.cfg` file to add a new database partition definition; do not use any of the procedures described, as they apply only when a database exists.

Windows Considerations: If you are using Enterprise Server Edition on a Windows operating system and have no databases in the instance, use the **db2ncrt** command to scale the database system. If, however, you already have databases, use the **START DBM ADD DBPARTITIONNUM** command to ensure that a database partition is created for each existing database when you scale the system. On Windows operating systems, do not manually edit the database partition configuration file (`db2nodes.cfg`), because this can introduce inconsistencies to the file.

Adding an online database partition

You can add new database partitions that are online to a partitioned database environment while it is running and while applications are connected to databases.

Procedure

To add an online database partition to a running database manager using the command line:

1. On any existing database partition, run the **START DBM** command.

   On all platforms, specify the new database partition values for **DBPARTITIONNUM**, **ADD DBPARTITIONNUM**, **HOSTNAME**, **PORT**, and **NETNAME** parameters. On the Windows platform, you also specify the **COMPUTER**, **USER**, and **PASSWORD** parameters.

   You can also specify the source for any temporary table space container definitions that must be created with the databases. If you do not provide table space information, temporary table space container definitions are retrieved from the catalog partition for each database.

   For example, to add three new database partitions to an existing database, issue the following commands:

   ```
   START DBM DBPARTITIONNUM 3 ADD DBPARTITIONNUM HOSTNAME HOSTNAME3
    PORT PORT3;
   ```

   ```
   START DBM DBPARTITIONNUM 4 ADD DBPARTITIONNUM HOSTNAME HOSTNAME4
    PORT PORT4;
   ```

   ```
   START DBM DBPARTITIONNUM 5 ADD DBPARTITIONNUM HOSTNAME HOSTNAME5
    PORT PORT5;
   ```

2. Optional: Alter the database partition group to incorporate the new database partition.

   This action can also be an option when redistributing the data to the new database partition.

3. Optional: Redistribute data to the new database partition.

   This action is not really optional if you want to take advantage of the new database partitions. You can also include the alter database partition group option as part of the redistribution operation. Otherwise, altering the database partition group to incorporate the new database partitions must be done as a separate action before redistributing the data to the new database partition.

4. Optional: Back up all databases on the new database partition.

   Although optional, this would be helpful to have for the new database partition and for the other database partitions particularly if you redistributed the data across both the old and the new database partitions.

Restrictions when working online to add a database partition

The status of the new database partition following its addition to the instance depends on the status of the original database partition. Applications may or may not be aware of the new database partition following its addition to the instance if the application uses WITH HOLD cursors.

When adding a new database partition to a single-partition database instance:

- If the original database partition is up when the database partition is added, then the new database partition is down when the add database partition operation completes.
- If the original database partition is down when the database partition is added, then the new database partition is up when the add database partition operation completes.

Applications using WITH HOLD cursors that are started before the add database partition operation runs are not aware of the new database partition when the add database partition operation completes. If the WITH HOLD cursors are closed before the add database partition operation runs, then applications are aware of the new database partition when the add database partition operation completes

Adding a database partition offline (Windows)

You can add new database partitions to a partitioned database system while it is stopped. The newly added database partition becomes available to all databases when the database manager is started again.

Before you begin

- You must install the new server before you can create a database partition on it.
- Set the default value of the **DB2_FORCE_OFFLINE_ADD_PARTITION** registry variable to TRUE to enforce that any added database partitions is offline.

Procedure

To add a database partition to a stopped partitioned database server using the command line:

1. Issue **STOP DBM** to stop all the database partitions.
2. Run the **ADD DBPARTITIONNUM** command on the new server.

   A database partition is created locally for every database that already exists in the system. The database parameters for the new database partitions are set to the default value, and each database partition remains empty until you move data to it. Update the database configuration parameter values to match those on the other database partitions.
3. Run the **START DBM** command to start the database system.

   Note that the database partition configuration file has already been updated by the database manager to include the new server during the installation of the new server.
4. Update the configuration file on the new database partition as follows:
   a) On any existing database partitions, run the **START DBM** command.

      Specify the new database partition values for **DBPARTITIONNUM**, **ADD DBPARTITIONNUM**, **HOSTNAME**, **PORT**, and **NETNAME** parameters as well as the **COMPUTER**, **USER**, and **PASSWORD** parameters.

      You can also specify the source for any temporary table space container definitions that need to be created with the databases. If you do not provide table space information, temporary table space container definitions are retrieved from the catalog partition for each database.

      For example, to add three new database partitions to an existing database, issue the following commands:

      ```
      START DBM DBPARTITIONNUM 3 ADD DBPARTITIONNUM HOSTNAME HOSTNAME3
       PORT PORT3;
      ```

      ```
      START DBM DBPARTITIONNUM 4 ADD DBPARTITIONNUM HOSTNAME HOSTNAME4
       PORT PORT4;
      ```

      ```
      START DBM DBPARTITIONNUM 5 ADD DBPARTITIONNUM HOSTNAME HOSTNAME5
       PORT PORT5;
      ```

      When the **START DBM** command is complete, the new server is stopped.
   b) Stop the database manager by running the **STOP DBM** command.

      When you stop all the database partitions in the system, the node configuration file is updated to include the new database partitions. The node configuration file is not updated with the new server information until **STOP DBM** is executed. This ensures that the **ADD DBPARTITIONNUM** command, which is called when you specify the **ADD DBPARTITIONNUM** parameter to the **START DBM** command, runs on the correct database partitions. When the utility ends, the new server partitions are stopped.
5. Start the database manager by running the **START DBM** command.

   The newly added database partitions are now started with the rest of the system.

When all the database partitions in the system are running, you can run system-wide activities, such as creating or dropping a database.

Note: You might have to issue the **START DBM** command twice for all database partition servers to access the new `db2nodes.cfg` file.

6. Optional: Alter the database partition group to incorporate the new database partition.

   This action could also be an option when redistributing the data to the new database partition.

7. Optional: Redistribute data to the new database partition.

   This action is not really optional if you want to take advantage of the new database partition. You can also include the alter database partition group option as part of the redistribution operation. Otherwise, altering the database partition group to incorporate the new database partition must be done as a separate action before redistributing the data to the new database partition.

8. Optional: Back up all databases on the new database partition.

   Although optional, this would be helpful to have for the new database partition and for the other database partitions particularly if you have redistributed the data across both the old and the new database partitions.

Adding a database partition offline (Linux and UNIX)

You can add new database partitions that are offline to a partitioned database system. The newly added database partition becomes available to all databases when the database manager is started again.

Before you begin

- Install the new server if it does not exist before you can create a database partition on it.
- Make the executables accessible using shared filesystem mounts or local copies.
- Synchronize operating system files with those on existing processors.
- Ensure that the `sqllib` directory is accessible as a shared file system.
- Ensure that the relevant operating system parameters (such as the maximum number of processes) are set to the appropriate values.
- Register the host name with the name server or in the `hosts` file in the `/etc` directory on all database partitions. The host name for the computer must be registered in `.rhosts` to run remote commands using **rsh** or **rah**.
- Set the default value of the **DB2_FORCE_OFFLINE_ADD_PARTITION** registry variable to TRUE to enforce that the added database partitions is offline.

Procedure

- To add a database partition to a stopped partitioned database server using the command line:
  a) Issue **STOP DBM** to stop all the database partitions.
  b) Run the **ADD DBPARTITIONNUM** command on the new server.

     A database partition is created locally for every database that exists in the system. The database parameters for the new database partitions are set to the default value, and each database partition remains empty until you move data to it. Update the database configuration parameter values to match those on the other database partitions.

  c) Run the **START DBM** command to start the database system.

     Note that the database partition configuration file (`db2nodes.cfg`) has already been updated by the database manager to include the new server during the installation of the new server.

  d) Update the configuration file on the new database partition as follows:

     a. On any existing database partition, run the **START DBM** command.

        Specify the new database partition values for **DBPARTITIONNUM**, **ADD DBPARTITIONNUM**, **HOSTNAME**, **PORT**, and **NETNAME** parameters as well as the **COMPUTER**, **USER**, and **PASSWORD** parameters.

You can also specify the source for any temporary table space container definitions that must be created with the databases. If you do not provide table space information, temporary table space container definitions are retrieved from the catalog partition for each database.

For example, to add three new database partitions to an existing database, issue the following commands:

```
START DBM DBPARTITIONNUM 3 ADD DBPARTITIONNUM HOSTNAME HOSTNAME3
 PORT PORT3;
```

```
START DBM DBPARTITIONNUM 4 ADD DBPARTITIONNUM HOSTNAME HOSTNAME4
 PORT PORT4;
```

```
START DBM DBPARTITIONNUM 5 ADD DBPARTITIONNUM HOSTNAME HOSTNAME5
 PORT PORT5;
```

When the **START DBM** command is complete, the new server is stopped.

b. Stop the entire database manager by running the **STOP DBM** command.

When you stop all the database partitions in the system, the node configuration file is updated to include the new database partitions. The node configuration file is not updated with the new server information until **STOP DBM** is executed. This ensures that the **ADD DBPARTITIONNUM** command, which is called when you specify the **ADD DBPARTITIONNUM** parameter to the **START DBM** command, runs on the correct database partition. When the utility ends, the new server partitions are stopped.

e) Start the database manager by running the **START DBM** command.

The newly added database partition is now started with the rest of the system.

When all the database partitions in the system are running, you can run system-wide activities, such as creating or dropping a database.

Note: You might have to issue the **START DBM** command twice for all database partition servers to access the new `db2nodes.cfg` file.

f) Optional: Alter the database partition group to incorporate the new database partition.

This action might also be an option when redistributing the data to the new database partition.

g) Optional: Redistribute data to the new database partition.

This action is not really optional if you want to take advantage of the new database partition. You can also include the alter database partition group option as part of the redistribution operation. Otherwise, altering the database partition group to incorporate the new database partition must be done as a separate action before redistributing the data to the new database partition.

h) Optional: Back up all databases on the new database partition.

Although optional, this would be helpful to have for the new database partition and for the other database partitions particularly if you redistributed the data across both the old and the new database partitions.

- You can also update the configuration file manually, as follows:

a) Edit the `db2nodes.cfg` file and add the new database partition to it.

b) Issue the following command to start the new database partition: `START DBM DBPARTITIONNUM` *`partitionnum`*

Specify the number you are assigning to the new database partition as the value of *partitionnum*.

c) If the new server is to be a logical partition (that is, it is not database partition 0), use **db2set** command to update the **DBPARTITIONNUM** registry variable.

Specify the number of the database partition you are adding.

d) Run the **ADD DBPARTITIONNUM** command on the new database partition.

This command creates a database partition locally for every database that exists in the system. The database parameters for the new database partitions are set to the default value, and each

database partition remains empty until you move data to it. Update the database configuration parameter values to match those on the other database partitions.

e) When the **ADD DBPARTITIONNUM** command completes, issue the **START DBM** command to start the other database partitions in the system.

Do not perform any system-wide activities, such as creating or dropping a database, until all database partitions are successfully started.

Error recovery when adding database partitions

Adding database partitions does not fail as a result of nonexistent buffer pools, because the database manager creates system buffer pools to provide default automatic support for all buffer pool page sizes.

However, if one of these system buffer pools is used, performance might be seriously affected, because these buffer pools are very small. If a system buffer pool is used, a message is written to the administration notification log. System buffer pools are used in database partition addition scenarios in the following circumstances:

- You add database partitions to a partitioned database environment that has one or more system temporary table spaces with a page size that is different from the default of 4 KB. When a database partition is created, only the IBMDEFAULTDP buffer pool exists, and this buffer pool has a page size of 4 KB.

  Consider the following examples:

  1. You use the **START DBM** command to add a database partition to the current multi-partition database:

     ```
     START DBM DBPARTITIONNUM 2 ADD DBPARTITIONNUM HOSTNAME newhost PORT 2
     ```

  2. You use the **ADD DBPARTITIONNUM** command after you manually update the `db2nodes.cfg` file with the new database partition description.

  One way to prevent these problems is to specify the WITHOUT TABLESPACES clause on the **ADD DBPARTITIONNUM** or the **START DBM** commands. After doing this, use the CREATE BUFFERPOOL statement to create the buffer pools using the appropriate SIZE and PAGESIZE values, and associate the system temporary table spaces to the buffer pool using the ALTER TABLESPACE statement.
- You add database partitions to an existing database partition group that has one or more table spaces with a page size that is different from the default page size, which is 4 KB. This occurs because the non-default page-size buffer pools created on the new database partition have not been activated for the table spaces.

  Note: In previous versions, this command used the NODEGROUP keyword instead of the DATABASE PARTITION GROUP keywords.

  Consider the following example:

  – You use the ALTER DATABASE PARTITION GROUP statement to add a database partition to a database partition group, as follows:

    ```
    START DBM
    CONNECT TO mpp1
    ALTER DATABASE PARTITION GROUP ng1 ADD DBPARTITIONNUM (2)
    ```

    One way to prevent this problem is to create buffer pools for each page size and then to reconnect to the database before issuing the following ALTER DATABASE PARTITION GROUP statement:

    ```
    START DBM
    CONNECT TO mpp1
    CREATE BUFFERPOOL bp1 SIZE 1000 PAGESIZE 8192
    CONNECT RESET
    CONNECT TO mpp1
    ALTER DATABASE PARTITION GROUP ng1 ADD DBPARTITIONNUM (2)
    ```

Note: If the database partition group has table spaces with the default page size, message SQL1759W is returned.

Dropping database partitions

You can drop a database partition that is not being used by any database and free the computer for other uses.

Before you begin

Verify that the database partition is not in use by issuing the **DROP DBPARTITIONNUM VERIFY** command or the `sqledrpn` API.

- If you receive message SQL6034W (Database partition not used in any database), you can drop the database partition.
- If you receive message SQL6035W (Database partition in use by database), use the **REDISTRIBUTE DATABASE PARTITION GROUP** command to redistribute the data from the database partition that you are dropping to other database partitions from the database alias.

Also ensure that all transactions for which this database partition was the coordinator have all committed or rolled back successfully. This might require doing crash recovery on other servers. For example, if you drop the coordinator partition, and another database partition participating in a transaction crashed before the coordinator partition was dropped, the crashed database partition will not be able to query the coordinator partition for the outcome of any indoubt transactions.

Procedure

Issue the **STOP DBM** command with the **DROP DBPARTITIONNUM** parameter to drop the database partition.

After the command completes successfully, the system is stopped. Then start the database manager with the **START DBM** command.

Listing database partition servers in an instance (Windows)

On Windows, use the **db2nlist** command to obtain a list of database partition servers that participate in an instance.

About this task

The command is used as follows:

```
db2nlist
```

When using this command as shown, the default instance is the current instance (set by the **DB2INSTANCE** environment variable). To specify a particular instance, you can specify the instance using:

```
db2nlist /i:instName
```

where *instName* is the particular instance name you want.

You can also optionally request the status of each database partition server by using:

```
  db2nlist /s
```

The status of each database partition server might be one of: starting, running, stopping, or stopped.

Adding database partition servers to an instance (Windows)

On Windows, use the **db2ncrt** command to add a database partition server to an instance.

About this task

Note: Do not use the **db2ncrt** command if the instance already contains databases. Instead, use the **START DBM ADD DBPARTITIONNUM** command. This ensures that the database is correctly added to the new database partition server. **DO NOT EDIT** the `db2nodes.cfg` file, since changing the file might cause inconsistencies in the partitioned database environment.

The command has the following required parameters:

```
db2ncrt /n:partition_number
        /u:username,password
        /p:logical_port
```

/n:partition_number
The unique database partition number to identify the database partition server. The number can be from 1 to 999 in ascending sequence.

/u:username,password
The logon account name and password of the Db2 service.

/p:logical_port
The logical port number used for the database partition server if the logical port is not zero (0). If not specified, the logical port number assigned is 0.

The logical port parameter is only optional when you create the first database partition on a computer. If you create a logical database partition, you must specify this parameter and select a logical port number that is not in use. There are several restrictions:

- On every computer there must be a database partition server with a logical port 0.
- The port number cannot exceed the port range reserved for FCM communications in the services file in %SystemRoot%\system32\drivers\etc directory. For example, if you reserve a range of four ports for the current instance, then the maximum port number would be 3 (ports 1, 2, and 3; port 0 is for the default logical database partition). The port range is defined when **db2icrt** is used with the /r:base_port, end_port parameter.

There are also several optional parameters:

/g:network_name
Specifies the network name for the database partition server. If you do not specify this parameter, Db2 uses the first IP address it detects on your system.

Use this parameter if you have multiple IP addresses on a computer and you want to specify a specific IP address for the database partition server. You can enter the network_name parameter using the network name or IP address.

/h:host_name
The TCP/IP host name that is used by FCM for internal communications if the host name is not the local host name. This parameter is required if you add the database partition server on a remote computer.

/i:instance_name
The instance name; the default is the current instance.

/m:computer_name
The computer name of the Windows workstation on which the database partition resides; the default name is the computer name of the local computer.

/o:instance_owning_computer
The computer name of the computer that is the instance-owning computer; the default is the local computer. This parameter is required when the **db2ncrt** command is invoked on any computer that is not the instance-owning computer.

For example, if you want to add a new database partition server to the instance TESTMPP (so that you are running multiple logical database partitions) on the instance-owning computer MYMACHIN, and you want this new database partition to be known as database partition 2 using logical port 1, enter:

```
db2ncrt /n:2 /p:1 /u:my_id,my_pword /i:TESTMPP
   /M:TEST /o:MYMACHIN
```

Changing database partitions (Windows)

On Windows, use the **db2nchg** command to change database partitions.

About this task

- Move the database partition from one computer to another.
- Change the TCP/IP host name of the computer.

  If you are planning to use multiple network adapters, you must use this command to specify the TCP/IP address for the "netname" field in the db2nodes.cfg file.
- Use a different logical port number.
- Use a different name for the database partition server.

The command has the following required parameter:

```
db2nchg /n:node_number
```

The parameter **/n:** is the number of the database partition server that you want to change. This parameter is required.

Optional parameters include:

/i:*instance_name*
Specifies the instance that this database partition server participates in. If you do not specify this parameter, the default is the current instance.

/u:*username*,*password*
Changes the logon account name and password for the Db2 database service. If you do not specify this parameter, the logon account and password remain the same.

/p:*logical_port*
Changes the logical port for the database partition server. This parameter must be specified if you move the database partition server to a different computer. If you do not specify this parameter, the logical port number remains unchanged.

/h:*host_name*
Changes the TCP/IP host name used by FCM for internal communications. If you do not specify this parameter, the host name is unchanged.

/m:*computer_name*
Moves the database partition server to another computer. The database partition server can be moved only if there are no existing databases in the instance.

/g:*network_name*
Changes the network name for the database partition server.

Use this parameter if you have multiple IP addresses on a computer and you want to use a specific IP address for the database partition server. You can enter the *network_name* using the network name or the IP address.

For example, to change the logical port assigned to database partition 2, which participates in the instance TESTMPP, to use the logical port 3, enter the following command:

```
db2nchg /n:2 /i:TESTMPP /p:3
```

The Db2 database manager provides the capability of accessing Db2 database system registry variables at the instance level on a remote computer.

Currently, Db2 database system registry variables are stored in three different levels: computer or global level, instance level, and database partition level. The registry variables stored at the instance level (including the database partition level) can be redirected to another computer by using **DB2REMOTEPREG**.

When **DB2REMOTEPREG** is set, the Db2 database manager accesses the Db2 database system registry variables from the computer pointed to by **DB2REMOTEPREG**. The **db2set** command would appear as:

```
db2set DB2REMOTEPREG=remote_workstation
```

where *remote_workstation* is the remote workstation name.

Note:

- Care must be taken in setting this option since all Db2 database instance profiles and instance listings will be located on the specified remote computer name.
- If your environment includes users from domains, ensure that the logon account associated with the Db2 instance service is a domain account. This ensures that the Db2 instance has the appropriate privileges to enumerate groups at the domain level.

This feature might be used in combination with setting **DBINSTPROF** to point to a remote LAN drive on the same computer that contains the registry.

Adding containers to SMS table spaces on database partitions

You can add a container to an SMS table space only on a database partition that currently has no containers.

Procedure

- To add a container to an SMS table space using the command line, enter the following:

  ```
  ALTER TABLESPACE name
     ADD ('path')
     ON DBPARTITIONNUM (database_partition_number)
  ```

  The database partition specified by number, and every partition in the range of database partitions, must exist in the database partition group on which the table space is defined. A *database_partition_number* might only appear explicitly or within a range in exactly one *db-partitions-clause* for the statement.

Example

The following example shows how to add a new container to database partition number 3 of the database partition group used by table space "plans" on a UNIX operating system:

```
ALTER TABLESPACE plans
   ADD ('/dev/rhdisk0')
   ON DBPARTITIONNUM (3)
```

Dropping a database partition from an instance (Windows)

On Windows, use the **db2ndrop** command to drop a database partition server from an instance that has no databases. If you drop a database partition server, its database partition number can be reused for a new database partition server.

About this task

Exercise caution when you drop database partition servers from an instance. If you drop the instance-owning database partition server zero (0) from the instance, the instance becomes unusable. If you want to drop the instance, use the **db2idrop** command.

Note: Do not use the **db2ndrop** command if the instance contains databases. Instead, use the **STOP DBM DROP DBPARTITIONNUM** command. This ensures that the database is correctly removed from the database partition. **DO NOT EDIT** the db2nodes.cfg file, since changing the file might cause inconsistencies in the partitioned database environment.

If you want to drop a database partition that is assigned the logical port 0 from a computer that is running multiple logical database partitions, you must drop all the other database partitions assigned to the other

logical ports before you can drop the database partition assigned to logical port 0. Each database partition server must have a database partition assigned to logical port 0.

The command has the following parameters:

```
db2ndrop /n:dbpartitionnum /i:instance_name
```

/n:*dbpartitionnum*
The unique database partition number (*dbpartitionnum*) to identify the database partition server. This is a required parameter. The number can be from zero (0) to 999 in ascending sequence. Recall that database partition zero (0) represents the instance-owning computer.

/i:*instance_name*
The instance name (*instance_name*). This is an optional parameter. If not given, the default is the current instance (set by the **DB2INSTANCE** registry variable).

Scenario: Redistributing data in new database partitions

This scenario shows how to add new database partitions to a database and redistribute data between the database partitions. The **REDISTRIBUTE DATABASE PARTITION GROUP** command is demonstrated as part of showing how to redistribute data on different table sets within a database partition group.

About this task

Scenario:
A database DBPG1 has two database partitions, specified as (0, 1) and a database partition group definition (0, 1).

The following table spaces are defined on database partition group DBPG_1:

- Table space TS1 - this table space has two tables, T1 and T2
- Table space TS2 - this table space has three tables defined, T3, T4, and T5

Starting in Version 9.7, you can add database partitions while the database is running and while applications are connected to it. However, the operation can be performed offline in this scenario by changing the default value of the **DB2_FORCE_OFFLINE_ADD_PARTITION** registry variable to TRUE.

Procedure

To redistribute data between the database partitions in DBPG1:

1. Identify objects that must be disabled or removed before the redistribution.
   a) Replicate MQTs: This type of MQT is not supported as part of the redistribution operation. They must be dropped before running the redistribution and recreated afterward.

```
SELECT tabschema, tabname
  FROM syscat.tables
  WHERE partition_mode = 'R'
```

   b) Write-to-table event monitors: Disable any automatically activated write-to-table event monitors that have a table that resides in the database partition group to be redistributed.

```
SELECT distinct evmonname
  FROM syscat.eventtables E
  JOIN syscat.tables T on T.tabname = E.tabname
    AND T.tabschema = E.tabschema
  JOIN syscat.tablespaces S on S.tbspace = T.tbspace
    AND S.ngname = 'DBPG_1'
```

   c) Explain tables: It is recommended to create the explain tables in a single partition database partition group. If they are defined in a database partition group that requires redistribution, however, and the data generated to date does not need to be maintained, consider dropping them. The explain tables can be redefined once the redistribution is complete.
   d) Table access mode and state: Ensure that all tables in the database partition groups to be redistributed are in full access mode and normal table states.

```
SELECT DISTINCT TRIM(T.OWNER) || \'.\' || TRIM(T.TABNAME)
  AS NAME, T.ACCESS_MODE, A.LOAD_STATUS
 FROM SYSCAT.TABLES T, SYSCAT.DBPARTITIONGROUPS
  N, SYSIBMADM.ADMINTABINFO A
 WHERE T.PMAP_ID = N.PMAP_ID
  AND A.TABSCHEMA = T.OWNER
  AND A.TABNAME = T.TABNAME
  AND N.DBPGNAME = 'DBPG_1'
  AND (T.ACCESS_MODE <> 'F' OR A.LOAD_STATUS IS NOT NULL)
```

e) Statistics profiles: If a statistics profile is defined for the table, table statistics can be updated as part of the redistribution process. Having the redistribution utility update the statistics for the table reduces I/O, as all the data is scanned for the redistribution and no additional scan of the data is needed for **RUNSTATS**.

```
RUNSTATS on table schema.table
  USE PROFILE runstats_profile
  SET PROFILE ONLY
```

2. Review the database configuration.

   The **util_heap_sz** is critical to the data movement processing between database partitions - allocate as much memory as possible to **util_heap_sz** for the duration of the redistribution. Sufficient **sortheap** is required, if index rebuild is done as part of the redistribution. Increase **util_heap_sz** and **sortheap** as necessary to improve redistribution performance.

3. Retrieve the database configuration settings to be used for the new database partitions.

   When adding database partitions, a default database configuration is used. As a result, it is important to update the database configuration on the new database partitions before the **REDISTRIBUTE DATABASE PARTITION GROUP** command is issued. This sequence of events ensures that the configuration is balanced.

```
SELECT name,
  CASE WHEN deferred_value_flags = 'AUTOMATIC'
    THEN deferred_value_flags
    ELSE substr(deferred_value,1,20)
    END
  AS deferred_value
  FROM sysibmadm.dbcfg
  WHERE dbpartitionnum = existing-node
    AND deferred_value != ''
    AND name NOT IN ('hadr_local_host','hadr_local_svc','hadr_peer_window',
                'hadr_remote_host','hadr_remote_inst','hadr_remote_svc',
                'hadr_syncmode','hadr_timeout','backup_pending','codepage',
                'codeset','collate_info','country','database_consistent',
                'database_level','hadr_db_role','log_retain_status',
                'loghead','logpath','multipage_alloc','numsegs','pagesize',
                'release','restore_pending','restrict_access',
                'rollfwd_pending','territory','user_exit_status',
                'number_compat','varchar2_compat','database_memory')
```

4. Back up the database (or the table spaces in the pertinent database partition group), before starting the redistribution process.

   This action ensures a recent recovery point.

5. Add three new database partitions to the database.

   Issue the following commands:

```
START DBM DBPARTITIONNUM 3 ADD DBPARTITIONNUM HOSTNAME HOSTNAME3
 PORT PORT3 WITHOUT TABLESPACES;
```

```
START DBM DBPARTITIONNUM 4 ADD DBPARTITIONNUM HOSTNAME HOSTNAME4
 PORT PORT4 WITHOUT TABLESPACES;
```

```
START DBM DBPARTITIONNUM 5 ADD DBPARTITIONNUM HOSTNAME HOSTNAME5
 PORT PORT5 WITHOUT TABLESPACES;
```

If the **DB2_FORCE_OFFLINE_ADD_PARTITION** is set to TRUE, new database partitions are not visible to the instance until it has been shut down and restarted. For example:

```
STOP DBM;
START DBM;
```

6. Define system temporary table space containers on the newly defined database partitions.

```
ALTER TABLESPACE tablespace_name
  ADD container_information
  ON dbpartitionnums (3 to 5)
```

7. Add the new database partitions to the database partition groups.

   The following command changes the DBPG_1 definition from (0, 1) to (0, 1, 3, 4, 5):

```
ALTER DATABASE PARTITION GROUP DBPG_1
  ADD dbpartitionnums (3 to 5)
  WITHOUT TABLESPACES
```

8. Define permanent data table space containers on the newly defined database partitions.

```
ALTER TABLESPACE tablespace_name
  ADD container_information
  ON dbpartitionnums (3 to 5)
```

9. Apply the database configuration settings to the new database partitions (or issue a single **UPDATE DB CFG** command against all database partitions).
10. Capture the definition of and then drop any replicated MQTs existing in the database partition groups to be redistributed.

```
db2look -d DBPG1 -e -z
  schema -t replicated_MQT_table_names
  -o repMQTs.clp
```

11. Disable any write-to-table event monitors that exist in the database partition groups to be redistributed.

```
SET EVENT MONITOR monitor_name STATE 0
```

12. Run the redistribution utility to redistribute uniformly across all database partitions.

```
REDISTRIBUTE DATABASE PARTITION GROUP DBPG_1 NOT ROLLFORWARD RECOVERABLE
 UNIFORM STOP AT 2006-03-10-07.00.00.000000;
```

    Let us presume that the command ran successfully for tables T1, T2 and T3, and then stopped due to the specification of the **STOP AT** option.

    To abort the data redistribution for the database partition group and to revert the changes made to tables T1, T2, and T3, issue the following command:

```
REDISTRIBUTE DATABASE PARTITION GROUP DBPG_1
NOT ROLLFORWARD RECOVERABLE ABORT;
```

    You might abort the data redistribution when an error or an interruption occurs and you do not want to continue the redistribute operation. For this scenario, presume that this command was run successfully and that tables T1 and T2 were reverted to their original state.

    To redistribute T5 and T4 only with 5000 4K pages as **DATA BUFFER**:

```
REDISTRIBUTE DATABASE PARTITION GROUP DBPG_1 NOT ROLLFORWARD RECOVERABLE
 UNIFORM TABLE (T5, T4) ONLY DATA BUFFER 5000;
```

    If the command ran successfully, the data in tables T4 and T5 have been redistributed successfully.

    To complete the redistribution of data on table T1, T2, and T3 in a specified order, issue:

```
REDISTRIBUTE DATABASE PARTITION GROUP DBPG_1 NOT ROLLFORWARD RECOVERABLE
 CONTINUE TABLE (T1) FIRST;
```

Specifying **TABLE (T1) FIRST** forces the database manager to process table T1 first so that it can return to being online (read-only) before other tables. All other tables are processed in an order determined by the database manager.

Note:

- The **ADD DBPARTITIONNUM** parameter can be specified in the **REDISTRIBUTE DATABASE PARTITION GROUP** command as an alternative to performing the ALTER DATABASE PARTITION GROUP and ALTER TABLESPACE statements in steps "7" on page 131 and "8" on page 131. When a database partition is added by using this command parameter, containers for table spaces are based on the containers of the corresponding table space on the lowest numbered existing partition in the database partition group.
- The **REDISTRIBUTE DATABASE PARTITION GROUP** command in this example is not roll-forward recoverable.
- After the **REDISTRIBUTE DATABASE PARTITION GROUP** command finishes, all the table spaces it accessed will be left in the BACKUP PENDING state. Such table spaces must be backed up before the tables they contain are accessible for write operations.

For more information, refer to the "REDISTRIBUTE DATABASE PARTITION GROUP command".

Consider also specifying a table list as input to the **REDISTRIBUTE DATABASE PARTITION GROUP** command to enforce the order that the tables are processed. The redistribution utility will move the data (compressed and compacted). Optionally, indexes will be rebuilt and statistics updated if statistics profiles are defined. Therefore instead of previous command, the following script can be run:

```
REDISTRIBUTE DATABASE PARTITION GROUP DBPG_1
  NOT ROLLFORWARD RECOVERABLE uniform
  TABLE (t1, t2,...) FIRST;
```

Issuing commands in partitioned database environments

In a partitioned database environment, you might want to issue commands to be run on computers in the instance, or on database partition servers. You can do so using the **rah** command or the **db2_all** command. The **rah** command allows you to issue commands that you want to run at computers in the instance.

If you want the commands to run at database partition servers in the instance, you run the **db2_all** command. This section provides an overview of these commands. The information that follows applies to partitioned database environments only.

On Windows, to run the **rah** command or the **db2_all** command, you must be logged on with a user account that is a member of the Administrators group.

On Linux and UNIX operating systems, your login shell can be a Korn shell or any other shell; however, there are differences in the way the different shells handle commands containing special characters.

Also, on Linux and UNIX operating systems, **rah** uses the remote shell program specified by the **DB2RSHCMD** registry variable. You can select between the two remote shell programs: ssh (for additional security), or rsh. If **DB2RSHCMD** is not set, rsh is used. The ssh remote shell program is used to prevent the transmission of passwords in clear text in UNIX operating system environments.

If a command runs on one database partition server and you want it to run on all of them, use **db2_all**. The exception is the **db2trc** command, which runs on all the logical database partition servers on a computer. If you want to run **db2trc** on all logical database partition servers on all computers, use **rah**.

Note: The **db2_all** command does not support commands that require interactive user input.

rah and db2_all commands overview

You can run the commands sequentially at one database partition server after another, or you can run the commands in parallel.

On Linux and UNIX operating systems, if you run the commands in parallel, you can either choose to have the output sent to a buffer and collected for display (the default behavior) or the output can be displayed at the computer where the command is issued. On Windows, if you run the commands in parallel, the output is displayed at the computer where the command is issued.

To use the **rah** command, type:

```
rah command
```

To use the **db2_all** command, type:

```
db2_all command
```

To obtain help about **rah** syntax, type:

```
rah "?"
```

The command can be almost anything that you can type at an interactive prompt, including, for example, multiple commands to be run in sequence. On Linux and UNIX operating systems, you separate multiple commands using a semicolon (;). On Windows, you separate multiple commands using an ampersand (&). Do not use the separator character following the last command.

The following example shows how to use the **db2_all** command to change the database configuration on all database partitions that are specified in the database partition configuration file. Because the ; character is placed inside double quotation marks, the request runs concurrently.

```
db2_all ";DB2 UPDATE DB CFG FOR sample USING LOGFILSIZ 100"
```

Note: The **db2_all** command does not support commands that require interactive user input.

Specifying the rah and db2_all commands

You can specify **rah** command from the command line as the parameter, or in response to the prompt if you do not specify any parameter.

Use the prompt method if the command contains the following special characters:

```
  | & ; < > ( )  { } [ ] unsubstituted $
```

If you specify the command as the parameter on the command line, you must enclose it in double quotation marks if it contains any of the special characters just listed.

Note: On Linux and UNIX operating systems, the command is added to your command history just as if you typed it at the prompt.

All special characters in the command can be entered normally (without being enclosed in quotation marks, except for \). If you require a \ in your command, you must type two backslashes (\\).

Note: On Linux and UNIX operating systems, if you are not using a Korn shell, all special characters in the command can be entered normally (without being enclosed in quotation marks, except for ", \, unsubstituted $, and the single quotation mark (')). If you require one of these characters in your command, you must precede them by three backslashes (\\\). For example, if you require a \ in your command, you must type four backslashes (\\\\).

If you require a double quotation mark (") in your command, you must precede it by three backslashes, for example, \\\".

Note:

1. On Linux and UNIX operating systems, you cannot include a single quotation mark (') in your command unless your command shell provides some way of entering a single quotation mark inside a singly quoted string.
2. On Windows, you cannot include a single quotation mark (') in your command unless your command window provides some way of entering a single quotation mark inside a singly quoted string.

When you run any korn-shell shell-script that contains logic to read from stdin in the background, explicitly redirect stdin to a source where the process can read without getting stopped on the terminal (SIGTTIN message). To redirect stdin, you can run a script with the following form:

```
shell_script </dev/null &
```

if there is no input to be supplied.

In a similar way, always specify `</dev/null` when running **db2_all** in the background. For example:

```
db2_all ";run_this_command" </dev/null &
```

By doing this you can redirect stdin and avoid getting stopped on the terminal.

An alternative to this method, when you are not concerned about output from the remote command, is to use the "daemonize" option in the **db2_all** prefix:

```
db2_all ";daemonize_this_command" &
```

Running commands in parallel (Linux, UNIX)
By default, the command is run sequentially at each computer, but you can specify to run the commands in parallel using background rshells by prefixing the command with certain prefix sequences. If the rshell is run in the background, then each command puts the output in a buffer file at its remote computer.

Note: The information in this section applies to Linux and UNIX operating systems only.

This process retrieves the output in two pieces:

1. After the remote command completes.
2. After the rshell terminates, which might be later if some processes are still running.

The name of the buffer file is `/tmp/$USER/rahout` by default, but it can be specified by the environment variables $**RAHBUFDIR** or $**RAHBUFNAME**.

When you specify that you want the commands to be run concurrently, by default, this script prefixes an additional command to the command sent to all hosts to check that $**RAHBUFDIR** and $**RAHBUFNAME** are usable for the buffer file. It creates $**RAHBUFDIR**. To suppress this, export an environment variable RAHCHECKBUF=no. You can do this to save time if you know that the directory exists and is usable.

Before using **rah** to run a command concurrently at multiple computers:

- Ensure that a directory `/tmp/$USER` exists for your user ID at each computer. To create a directory if one does not exist, run:

  ```
  rah ")mkdir /tmp/$USER"
  ```

- Add the following line to your `.kshrc` (for Korn shell syntax) or `.profile`, and also type it into your current session:

  ```
  export RAHCHECKBUF=no
  ```

- Ensure that each computer ID at which you run the remote command has an entry in its `.rhosts` file for the ID which runs **rah**; and the ID which runs **rah** has an entry in its `.rhosts` file for each computer ID at which you run the remote command.

Extension of the rah command to use tree logic (AIX)
To enhance performance, rah has been extended to use tree_logic on large systems. That is, rah will check how many database partitions the list contains, and if that number exceeds a threshold value, it constructs a subset of the list and sends a recursive invocation of itself to those database partitions.

At those database partitions, the recursively invoked rah follows the same logic until the list is small enough to follow the standard logic (now the "leaf-of-tree" logic) of sending the command to all database partitions on the list. The threshold can be specified by the **RAHTREETHRESH** environment variable, or defaults to 15.

In the case of a multiple-logical-database partitions-per-physical-database partition system, **db2_all** will favor sending the recursive invocation to distinct physical database partitions, which will then rsh to other logical database partitions on the same physical database partition, thus also reducing inter-physical-database partition traffic. (This point applies only to **db2_all**, not rah, because rah always sends only to distinct physical database partitions.)

rah and db2_all commands
This topic includes descriptions of the **rah** and **db2_all** commands.

Command
Description

rah
Runs the command on all computers.

db2_all
Runs a non-interactive command on all database partition servers that you specify. **db2_all** does not support commands that require interactive user input.

db2_kill
Abruptly stops all processes being run on multiple database partition servers and cleans up all resources on all database partition servers. This command renders your databases inconsistent. Do *not* issue this command except under direction from IBM Software Support or as directed to recover from a sustained trap.

db2_call_stack
On Linux and UNIX operating systems, causes all processes running on all database partition servers to write call traceback to the syslog.

On Linux and UNIX operating systems, these commands execute **rah** with certain implicit settings such as:

- Run in parallel at all computers
- Buffer command output in `/tmp/$USER/db2_kill`, `/tmp/$USER/db2_call_stack` respectively.

The command **db2_call_stack** is *not* available on Windows. Use the **db2pd -stack** command instead.

rah and db2_all command prefix sequences
A prefix sequence is one or more special characters.

Type one or more prefix sequences immediately preceding the characters of the command without any intervening blanks. If you want to specify more than one sequence, you can type them in any order, but characters within any multicharacter sequence must be typed in order. If you type any prefix sequences, you must enclose the entire command, including the prefix sequences in double quotation marks, as in the following examples:

- On Linux and UNIX operating systems:

```
rah "};ps -F pid,ppid,etime,args -u $USER"
db2_all "};ps -F pid,ppid,etime,args -u $USER"
```

- On Windows operating systems:

```
rah "||db2 get db cfg for sample"
db2_all "||db2 get db cfg for sample"
```

The prefix sequences are:

Sequence
Purpose

|
Runs the commands in sequence in the background.

|&
Runs the commands in sequence in the background and terminates the command after all remote commands have completed, even if some processes are still running. This might be later if, for example, child processes (on Linux and UNIX operating systems) or background processes (on Windows operating systems) are still running. In this case, the command starts a separate background process to retrieve any remote output generated after command termination and writes it back to the originating computer.

Note: On Linux and UNIX operating systems, specifying & degrades performance, because more **rsh** commands are required.

||
Runs the commands in parallel in the background.

||&
Runs the commands in parallel in the background and terminates the command after all remote commands have completed as described previously for the |& case.

Note: On Linux and UNIX operating systems, specifying & degrades performance, because more **rsh** commands are required.

;
Same as ||&. This is an alternative shorter form.

Note: On Linux and UNIX operating systems, specifying ; degrades performance relative to ||, because more **rsh** commands are required.

]
Prepends dot-execution of user's profile before executing command.

Note: Available on Linux and UNIX operating systems only.

}
Prepends dot-execution of file named in $**RAHENV** (probably `.kshrc`) before executing command.

Note: Available on Linux and UNIX operating systems only.

]}
Prepends dot-execution of user's profile followed by execution of file named in $**RAHENV** (probably `.kshrc`) before executing command.

Note: Available on Linux and UNIX operating systems only.

)
Suppresses execution of user's profile and of file named in $**RAHENV**.

Note: Available on Linux and UNIX operating systems only.

'
Echoes the command invocation to the computer.

<
Sends to all the computers except this one.

<<-*nnn*<

Sends to all-but-database partition server *nnn* (all database partition servers in db2nodes.cfg except for database partition number *nnn*, see the first paragraph following the last prefix sequence in this table).

nnn is the corresponding 1-, 2-, or 3-digit database partition number to the *nodenum* value in the db2nodes.cfg file.

<<-nnn< is only applicable to **db2_all**.

<<+*nnn*<

Sends to only database partition server *nnn* (the database partition server in db2nodes.cfg whose database partition number is *nnn*, see the first paragraph following the last prefix sequence in this table).

nnn is the corresponding 1-, 2-, or 3-digit database partition number to the *nodenum* value in the db2nodes.cfg file.

<<+nnn< is only applicable to **db2_all**.

(blank character)

Runs the remote command in the background with stdin, stdout, and stderr all closed. This option is valid only when running the command in the background, that is, only in a prefix sequence which also includes \ or ;. It allows the command to complete much sooner (as soon as the remote command has been initiated). If you specify this prefix sequence on the **rah** command line, then either enclose the command in single quotation marks, or enclose the command in double quotation marks, and precede the prefix character by \ . For example,

```
rah '; mydaemon'
```

or

```
rah ";\ mydaemon"
```

When run as a background process, the **rah** command never waits for any output to be returned.

>

Substitutes occurrences of > with the computer name.

"

Substitutes occurrences of () by the computer index, and substitutes occurrences of ## by the database partition number.

- The computer index is a number that associated with a computer in the database system. If you are not running multiple logical partitions, the computer index for a computer corresponds to the database partition number for that computer in the database partition configuration file. To obtain the computer index for a computer in a multiple logical partition database environment, do not count duplicate entries for those computers that run multiple logical partitions. For example, if MACH1 is running two logical partitions and MACH2 is also running two logical partitions, the database partition number for MACH3 is 5 in the database partition configuration file. The computer index for MACH3, however, would be 3.
  - On Windows operating systems, do not edit the database partition configuration file. To obtain the computer index, use the **db2nlist** command.
- When " is specified, duplicates are not eliminated from the list of computers.

Usage notes

- Prefix sequences are considered to be part of the command. If you specify a prefix sequence as part of a command, you must enclose the entire command, including the prefix sequences, in double quotation marks.

Controlling the rah command

This topic lists the environment variables to control the **rah** command.

*Table 13. Environment variables that control the **rah** command*

Name	Meaning	Default
$**RAHBUFDIR** **Note:** Available on Linux and UNIX operating systems only.	Directory for buffer	/tmp/$USER
$**RAHBUFNAME** **Note:** Available on Linux and UNIX operating systems only.	File name for buffer	rahout
$**RAHOSTFILE** (on Linux and UNIX operating systems); **RAHOSTFILE** (on Windows operating systems)	File containing list of hosts	db2nodes.cfg
$**RAHOSTLIST** (on Linux and UNIX operating systems); **RAHOSTLIST** (on Windows operating systems)	List of hosts as a string	extracted from $**RAHOSTFILE**
$**RAHCHECKBUF** **Note:** Available on Linux and UNIX operating systems only.	If set to "no", bypass checks	not set
$**RAHSLEEPTIME** (on Linux and UNIX operating systems); **RAHSLEEPTIME** (on Windows operating systems)	Time in seconds this script waits for initial output from commands run in parallel.	86400 seconds for **db2_kill**, 200 seconds for all others

*Table 13. Environment variables that control the **rah** command (continued)*

Name	Meaning	Default
$**RAHWAITTIME** (on Linux and UNIX operating systems); **RAHWAITTIME** (on Windows operating systems)	On Windows operating systems, interval in seconds between successive checks that remote jobs are still running. On Linux and UNIX operating systems, interval in seconds between successive checks that remote jobs are still running and `rah: waiting for pid> ...` messages. On all operating systems, specify any positive integer. Prefix value with a leading zero to suppress messages, for example, `export RAHWAITTIME=045`. It is not necessary to specify a low value as **rah** does not rely on these checks to detect job completion.	45 seconds
$**RAHENV** **Note:** Available on Linux and UNIX operating systems only.	Specifies file name to be executed if $**RAHDOTFILES**=E or K or PE or B	$**ENV**
$**RAHUSER** (on Linux and UNIX operating systems); **RAHUSER** (on Windows operating systems)	On Linux and UNIX operating systems, user ID under which the remote command is to be run. On Windows operating systems, the logon account associated with the Db2 Remote Command Service	$**USER**

Note: On Linux and UNIX operating systems, the value of $**RAHENV** where **rah** is run is used, not the value (if any) set by the remote shell.

Specifying which . files run with rah (Linux and UNIX)
This topics lists the . files that are run if no prefix sequence is specified.

Note: The information in this section applies to Linux and UNIX operating systems only.

P
`.profile`

E
File named in $**RAHENV** (probably `.kshrc`)

K
Same as E

PE
`.profile` followed by file named in $**RAHENV** (probably `.kshrc`)

B
Same as PE

N
None (or Neither)

Note: If your login shell is not a Korn shell, any dot files you specify to be executed are executed in a Korn shell process, and so must conform to Korn shell syntax. So, for example, if your login shell is a C shell, to have your `.cshrc` environment set up for commands executed by **rah**, you should either create a Korn shell *INSTHOME*`/.profile` equivalent to your `.cshrc` and specify in your *INSTHOME*`/.cshrc`:

```
setenv RAHDOTFILES P
```

or you should create a Korn shell *INSTHOME*/.kshrc equivalent to your .cshrc and specify in your *INSTHOME*/.cshrc:

```
setenv RAHDOTFILES E
setenv RAHENV INSTHOME/.kshrc
```

Also, it is your .cshrc must not write to stdout if there is no tty (as when invoked by **rsh**). You can ensure this by enclosing any lines which write to stdout by, for example,

```
if { tty -s } then echo "executed .cshrc";
endif
```

Determining problems with rah (Linux, UNIX)

This topic gives suggestions on how to handle some problems that you might encounter when you are running **rah**.

Note: The information in this section applies to Linux and UNIX operating systems only.

1. **rah** hangs (or takes a very long time)

   This problem might be caused because:

   - **rah** has determined that it needs to buffer output, and you did not export RAHCHECKBUF=no. Therefore, before running your command, **rah** sends a command to all computers to check the existence of the buffer directory, and to create it if it does not exist.
   - One or more of the computers where you are sending your command is not responding. The **rsh** command will eventually time out but the time-out interval is quite long, usually about 60 seconds.

2. You have received messages such as:

   - Login incorrect
   - Permission denied

   Either one of the computers does not have the ID running **rah** correctly defined in its /etc/hosts file, or the ID running **rah** does not have one of the computers correctly defined in its .rhosts file. If the **DB2RSHCMD** registry variable has been configured to use ssh, then the ssh clients and servers on each computer might not be configured correctly.

   Note: You might need to have greater security regarding the transmission of passwords in clear text between database partitions. This will depend on the remote shell program you are using. **rah** uses the remote shell program specified by the **DB2RSHCMD** registry variable. You can select between the two remote shell programs: ssh (for additional security), or rsh. If this registry variable is not set, rsh is used.

3. When running commands in parallel using background remote shells, although the commands run and complete within the expected elapsed time at the computers, **rah** takes a long time to detect this and put up the shell prompt.

   The ID running **rah** does not have one of the computers correctly defined in its .rhosts file, or if the **DB2RSHCMD** registry variable has been configured to use ssh, then the ssh clients and servers on each computer might not be configured correctly.

4. Although **rah** runs fine when run from the shell command line, if you run **rah** remotely using rsh, for example,

   ```
   rsh somewher -l $USER db2_kill
   ```

   rah never completes.

   This is normal. **rah** starts background monitoring processes, which continue to run after it has exited. Those processes normally persist until all processes associated with the command you ran have themselves terminated. In the case of **db2_kill**, this means termination of all database managers. You can terminate the monitoring processes by finding the process whose command is **rahwaitfor** and kill *process_id*>. Do not specify a signal number. Instead, use the default (15).

5. The output from **rah** is not displayed correctly, or **rah** incorrectly reports that $**RAHBUFNAME** does not exist, when multiple commands of **rah** were issued under the same $**RAHUSER**.

   This is because multiple concurrent executions of **rah** are trying to use the same buffer file (for example, $**RAHBUFDIR** or $**RAHBUFNAME**) for buffering the outputs. To prevent this problem, use a different $**RAHBUFNAME** for each concurrent **rah** command, for example in the following ksh:

   ```
   export RAHBUFNAME=rahout
   rah ";$command_1" &
   export RAHBUFNAME=rah2out
   rah ";$command_2" &
   ```

   or use a method that makes the shell choose a unique name automatically such as:

   ```
   RAHBUFNAME=rahout.$$ db2_all "....."
   ```

   Whatever method you use, you must ensure that you clean up the buffer files at some point if disk space is limited. **rah** does not erase a buffer file at the end of execution, although it will erase and then re-use an existing file the next time you specify the same buffer file.
6. You entered

   ```
   rah '"print from ()'
   ```

   and received the message:

   ```
   ksh: syntax error at line 1 : (' unexpected
   ```

   Prerequisites for the substitution of () and ## are:

   - Use **db2_all**, not **rah**.
   - Ensure a **RAHOSTFILE** is used either by exporting **RAHOSTFILE** or by defaulting to your `/sqllib/db2nodes.cfg` file. Without these prerequisites, **rah** leaves the () and ## as is. You receive an error because the command **print from ()** is not valid.

   For a performance tip when running commands in parallel, use | rather than |&, and use || rather than ||& or ; unless you truly need the function provided by &. Specifying & requires more remote shell commands and therefore degrades performance.

Monitoring rah processes (Linux, UNIX)

While any remote commands are still running or buffered output is still being accumulated, processes started by **rah** monitor activity to write messages to the terminal indicating which commands have not been run, and retrieve the buffered output.

About this task

Note: The information in this section applies to Linux and UNIX operating systems only.

The informative messages are written at an interval controlled by the environment variable **RAHWAITTIME**. Refer to the help information for details on how to specify this. All informative messages can be suppressed by exporting `RAHWAITTIME=0`.

The primary monitoring process is a command whose command name (as shown by the **ps** command) is **rahwaitfor**. The first informative message tells you the pid (process id) of this process. All other monitoring processes appear as **ksh** commands running the **rah** script (or the name of the symbolic link). If you want, you can stop all monitoring processes by the command:

```
kill pid
```

where *pid* is the process ID of the primary monitoring process. Do not specify a signal number. Leave the default of 15. This does not affect the remote commands at all, but prevents the automatic display of buffered output. Note that there might be two or more different sets of monitoring processes executing at different times during the life of a single execution of **rah**. However, if at any time you stop the current set, then no more are started.

If your regular login shell is not a Korn shell (for example `/bin/ksh`), you can use **rah**, but there are some slightly different rules on how to enter commands containing the following special characters:

```
"  unsubstituted $ '
```

For more information, type `rah "?"`. Also, in a Linux or UNIX operating system, if the login shell at the ID which executes the remote commands is not a Korn shell, then the login shell at the ID which executes **rah** must also not be a Korn shell. (**rah** decides whether the shell of the remote ID is a Korn shell based on the local ID). The shell must not perform any substitution or special processing on a string enclosed in single quotation marks. It must leave it exactly as is.

Setting the default environment profile for rah on Windows
To set the default environment profile for the **rah** command, use a file called `db2rah.env`, which should be created in the instance directory.

About this task

Note: The information in this section applies to Windows only.

The file should have the following format:

```
; This is a comment line
DB2INSTANCE=instancename
DB2DBDFT=database
; End of file
```

You can specify all the environment variables that you need to initialize the environment for **rah**.

Creating tables and other related table objects

Tables in partitioned database environments

There are performance advantages to creating a table across several database partitions in a partitioned database environment. The work associated with the retrieval of data can be divided among the database partitions.

Before you begin
Before creating a table that will be physically divided or distributed, you need to consider the following:

- Table spaces can span more than one database partition. The number of database partitions they span depends on the number of database partitions in a database partition group.
- Tables can be collocated by being placed in the same table space or by being placed in another table space that, together with the first table space, is associated with the same database partition group.

About this task

Creating a table that will be a part of several database partitions is specified when you are creating the table. There is an additional option when creating a table in a partitioned database environment: the *distribution key*. A distribution key is a key that is part of the definition of a table. It determines the database partition on which each row of data is stored.

If you do not specify the distribution key explicitly, a default distribution key is automatically defined.

You must be careful to select an appropriate distribution key because *it cannot be changed later*. Furthermore, any unique indexes (and therefore unique or primary keys) must be defined as a superset of the distribution key. That is, if a distribution key is defined, unique keys and primary keys must include all of the same columns as the distribution key (they might have more columns).

The size of a database partition of a table is the smaller amount of a specific limit associated with the type of table space and page size used, and the amount of disk space available. For example, assuming a large DMS table space with a 4 KB page size, the size of a table is the smaller amount of 8 TB multiplied by the

number of database partitions and the amount of available disk space. See the related links for the complete list of database manager page size limits.

To create a table in a partitioned database environment using the command line, enter:

```
CREATE TABLE name>
  (<column_name> <data_type> <null_attribute>)
  IN <tagle_space_name>
  INDEX IN <index_space_name>
  LONG IN <long_space_name>
  DISTRIBUTE BY HASH (<column_name>)
```

Following is an example:

```
CREATE TABLE MIXREC (MIX_CNTL INTEGER NOT NULL,
                     MIX_DESC CHAR(20) NOT NULL,
                     MIX_CHR  CHAR(9) NOT NULL,
                     MIX_INT INTEGER NOT NULL,
                     MIX_INTS SMALLINT NOT NULL,
                     MIX_DEC DECIMAL NOT NULL,
                     MIX_FLT FLOAT NOT NULL,
                     MIX_DATE DATE NOT NULL,
                     MIX_TIME TIME NOT NULL,
                     MIX_TMSTMP TIMESTAMP NOT NULL)
                     IN MIXTS12
                     DISTRIBUTE BY HASH (MIX_INT)
```

In the preceding example, the table space is MIXTS12 and the distribution key is MIX_INT. If the distribution key is not specified explicitly, it is MIX_CNTL. (If no primary key is specified and no distribution key is defined, the distribution key is the first non-long column in the list.)

A row of a table, and all information about that row, always resides on the same database partition.

Large object behavior in partitioned tables

A partitioned table uses a data organization scheme in which table data is divided across multiple storage objects, called data partitions or ranges, according to values in one or more table partitioning key columns of the table. Data from a given table is partitioned into multiple storage objects based on the specifications provided in the PARTITION BY clause of the CREATE TABLE statement. These storage objects can be in different table spaces, in the same table space, or a combination of both.

A large object for a partitioned table is, by default, stored in the same table space as its corresponding data object. This applies to partitioned tables that use only one table space or use multiple table spaces. When a partitioned table's data is stored in multiple table spaces, the large object data is also stored in multiple table spaces.

Use the LONG IN clause of the CREATE TABLE statement to override this default behavior. You can specify a list of table spaces for the table where long data is to be stored. If you choose to override the default behavior, the table space specified in the LONG IN clause must be a large table space. If you specify that long data be stored in a separate table space for one or more data partitions, you must do so for all the data partitions of the table. That is, you cannot have long data stored remotely for some data partitions and stored locally for others. Whether you are using the default behavior or the LONG IN clause to override the default behavior, a long object is created to correspond to each data partition. All the table spaces used to store long data objects corresponding to each data partition must have the same: pagesize, extentsize, storage mechanism (DMS or AMS), and type (regular or large). Remote large table spaces must be of type LARGE and cannot be SMS.

For example, the following CREATE TABLE statement creates objects for the CLOB data for each data partition in the same table space as the data:

```
CREATE TABLE document(id INT, contents CLOB)
PARTITION BY RANGE(id)
(STARTING FROM 1   ENDING AT 100 IN tbsp1,
 STARTING FROM 101 ENDING AT 200 IN tbsp2,
 STARTING FROM 201 ENDING AT 300 IN tbsp3,
 STARTING FROM 301 ENDING AT 400 IN tbsp4);
```

You can use LONG IN to place the CLOB data in one or more large table spaces, distinct from those the data is in.

```
CREATE TABLE document(id INT, contents CLOB)
PARTITION BY RANGE(id)
(STARTING FROM 1   ENDING AT 100 IN tbsp1 LONG IN large1,
STARTING FROM 101 ENDING AT 200 IN tbsp2 LONG IN large1,
STARTING FROM 201 ENDING AT 300 IN tbsp3 LONG IN large2,
STARTING FROM 301 ENDING AT 400 IN tbsp4 LONG IN large2);
```

Note: Only a single LONG IN clause is allowed at the table level and for each data partition.

Creating partitioned tables

Partitioned tables use a data organization scheme in which table data is divided across multiple storage objects, called data partitions or ranges, according to values in one or more table partitioning key columns of the table. Data from a given table is partitioned into multiple storage objects based on the specifications provided in the PARTITION BY clause of the CREATE TABLE statement. These storage objects can be in different table spaces, in the same table space, or a combination of both.

Before you begin

To create a table, the privileges held by the authorization ID of the statement must include at least one of the following authorities or privileges:

- CREATETAB authority on the database and USE privilege on all the table spaces used by the table, as well as one of:
  - IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the table does not exist
  - CREATEIN privilege on the schema, if the schema name of the table refers to an existing schema
- DBADM authority

About this task

You can create a partitioned table by using the CREATE TABLE statement.

Procedure

- To create a partitioned table from the command line, issue the CREATE TABLE statement:

  ```
  CREATE TABLE NAME (column_name data_type null_attribute) IN
      table_space_list PARTITION BY RANGE (column_expression)
      STARTING FROM constant ENDING constant EVERY constant
  ```

  For example, the following statement creates a table where rows with a ≥ 1 and a ≤ 20 are in PART0 (the first data partition), rows with 21 ≤ a ≤ 40 are in PART1 (the second data partition), up to 81 ≤ a ≤ 100 are in PART4 (the last data partition).

  ```
  CREATE TABLE foo(a INT)
      PARTITION BY RANGE (a) (STARTING FROM (1)
      ENDING AT (100) EVERY (20))
  ```

Defining ranges on partitioned tables

You can specify a range for each data partition when you create a partitioned table. A partitioned table uses a data organization scheme in which table data is divided across multiple data partitions according to the values of the table partitioning key columns of the table.

About this task

Data from a given table is partitioned into multiple storage objects based on the specifications provided in the PARTITION BY clause of the CREATE TABLE statement. A range is specified by the STARTING FROM and ENDING AT values of the PARTITION BY clause.

To completely define the range for each data partition, you must specify sufficient boundaries. The following is a list of guidelines to consider when defining ranges on a partitioned table:

- The STARTING clause specifies a low boundary for the data partition range. This clause is mandatory for the lowest data partition range (although you can define the boundary as MINVALUE). The lowest data partition range is the data partition with the lowest specified bound.
- The ENDING (or VALUES) clause specifies a high boundary for the data partition range. This clause is mandatory for the highest data partition range (although you can define the boundary as MAXVALUE). The highest data partition range is the data partition with the highest specified bound.
- If you do not specify an ENDING clause for a data partition, then the next greater data partition must specify a STARTING clause. Likewise, if you do not specify a STARTING clause, then the previous data partition must specify an ENDING clause.
- MINVALUE specifies a value that is smaller than any possible value for the column type being used. MINVALUE and INCLUSIVE or EXCLUSIVE cannot be specified together.
- MAXVALUE specifies a value that is larger than any possible value for the column type being used. MAXVALUE and INCLUSIVE or EXCLUSIVE cannot be specified together.
- INCLUSIVE indicates that all values equal to the specified value are to be included in the data partition containing this boundary.
- EXCLUSIVE indicates that all values equal to the specified value are NOT to be included in the data partition containing this boundary.
- The NULLS FIRST and NULLS LAST clauses of the CREATE TABLE statement specify whether null values are to be sorted high or low when considering data partition placement. By default, null values are sorted high. Null values in the table partitioning key columns are treated as positive infinity, and are placed in a range ending at MAXVALUE. If no such data partition is defined, null values are considered to be out-of-range values. Use the NOT NULL constraint if you want to exclude null values from table partitioning key columns. LAST specifies that null values are to appear last in a sorted list of values. FIRST specifies that null values are to appear first in a sorted list of values.
- When using the long form of the syntax, each data partition must have at least one bound specified.

Tip: Before you begin defining data partitions on a table it is important to understand how tables benefit from table partitioning and what factors influence the columns you choose as partitioning columns.

The ranges specified for each data partition can be generated automatically or manually.

Automatically generated

Automatic generation is a simple method of creating many data partitions quickly and easily. This method is appropriate for equal sized ranges based on dates or numbers.

Examples 1 and 2 demonstrate how to use the CREATE TABLE statement to define and generate automatically the ranges specified for each data partition.

Example 1:

Issue a create table statement with the following ranges defined:

```
CREATE TABLE lineitem (
  l_orderkey    DECIMAL(10,0) NOT NULL,
  l_quantity    DECIMAL(12,2),
  l_shipdate    DATE,
  l_year_month  INT GENERATED ALWAYS AS (YEAR(l_shipdate)*100 + MONTH(l_shipdate)))
     PARTITION BY RANGE(l_shipdate)
    (STARTING ('1/1/1992') ENDING ('12/31/1992') EVERY 1 MONTH);
```

This statement results in 12 data partitions each with 1 key value (l_shipdate) >= ('1/1/1992'), (l_shipdate) < ('3/1/1992'), (l_shipdate) < ('4/1/1992'), (l_shipdate) < ('5/1/1992'), ..., (l_shipdate) < ('12/1/1992'), (l_shipdate) <= ('12/31/1992').

The starting value of the first data partition is inclusive because the overall starting bound ('1/1/1992') is inclusive (default). Similarly, the ending bound of the last data partition is inclusive because the overall ending bound ('12/31/1992') is inclusive (default). The remaining STARTING values are inclusive and the

remaining ENDING values are all exclusive. Each data partition holds n key values where n is given by the EVERY clause. Use the formula (start + every) to find the end of the range for each data partition. The last data partition might have fewer key values if the EVERY value does not divide evenly into the START and END range.

Example 2:

Issue a create table statement with the following ranges defined:

```
CREATE TABLE t(a INT, b INT)
    PARTITION BY RANGE(b) (STARTING FROM (1)
  EXCLUSIVE ENDING AT (1000) EVERY (100))
```

This statement results in 10 data partitions each with 100 key values (1 < b <= 101, 101 < b <= 201, ..., 901 < b <= 1000).

The starting value of the first data partition (b > 1 and b <= 101) is exclusive because the overall starting bound (1) is exclusive. Similarly the ending bound of the last data partition (b > 901 b <= 1000) is inclusive because the overall ending bound (1000) is inclusive. The remaining STARTING values are all exclusive and the remaining ENDING values are all inclusive. Each data partition holds n key values where n is given by the EVERY clause. Finally, if both the starting and ending bound of the overall clause are exclusive, the starting value of the first data partition is exclusive because the overall starting bound (1) is exclusive. Similarly the ending bound of the last data partition is exclusive because the overall ending bound (1000) is exclusive. The remaining STARTING values are all exclusive and the ENDING values are all inclusive. Each data partition (except the last) holds n key values where n is given by the EVERY clause.

Example 3:

Issue a create table statement with the following ranges defined:

```
db2 "
CREATE TABLE lineitem2 (
 l_orderkey    DECIMAL(10,0) NOT NULL,
 l_quantity    DECIMAL(12,2),
 l_shipdate    TIMESTAMP,
 l_year_month  INT GENERATED ALWAYS AS (YEAR(l_shipdate)*100 + MONTH(l_shipdate)))
  PARTITION BY RANGE(l_shipdate)
  (STARTING ('1992-01-01-00.00.00.000000') ENDING ('1992-12-31-23.59.59.999999') EVERY 1 MONTH)
"
```

This statement results in 12 data partitions each with 1 key value (l_shipdate) >= ('1992-01-01-00.00.00.000000'), (l_shipdate) < ('1992-03-01-00.00.00.000000'), (l_shipdate) < ('1992-04-01-00.00.00.000000'), (l_shipdate) < ('1992-05-01-00.00.00.000000'), ..., (l_shipdate) < ('1992-12-01-00.00.00.000000), (l_shipdate) <= ('1992-12-31-23.59.59.999999').

The starting value of the first data partition is inclusive because the overall starting bound ('1992-01-01-00.00.00.000000') is inclusive (default). Similarly, the ending bound of the last data partition is inclusive because the overall ending bound ('1992-12-31-23.59.59.999999') is inclusive (default). The remaining STARTING values are inclusive and the remaining ENDING values are all exclusive. Each data partition holds n key values where n is given by the EVERY clause. Use the formula (start + every) to find the end of the range for each data partition. The last data partition might have fewer key values if the EVERY value does not divide evenly into the START and END range.

Further, during implicit conversion in case bounds specified as DATE, the ending bound of the last data partition is different:

```
db2 "
CREATE TABLE lineitem3 (
 l_orderkey    DECIMAL(10,0) NOT NULL,
 l_quantity    DECIMAL(12,2),
 l_shipdate    TIMESTAMP,
 l_year_month  INT GENERATED ALWAYS AS (YEAR(l_shipdate)*100 + MONTH(l_shipdate)))
  PARTITION BY RANGE(l_shipdate)
  (STARTING ('1/1/1992') ENDING ('12/31/1992') EVERY 1 MONTH)
"
```

This statement results in 12 data partitions each with 1 key value (l_shipdate) >= ('1992-01-01-00.00.00.000000'), (l_shipdate) < ('1992-03-01-00.00.00.000000'), (l_shipdate) < ('1992-04-01-00.00.00.000000'), (l_shipdate) < ('1992-05-01-00.00.00.000000'), ..., (l_shipdate) < ('1992-12-01-00.00.00.000000), (l_shipdate) <= ('1992-12-31-00.00.00.000000').

Manually generated

Manual generation creates a new data partition for each range listed in the PARTITION BY clause. This form of the syntax allows for greater flexibility when defining ranges thereby increasing your data and LOB placement options. Examples 4 and 5 demonstrate how to use the CREATE TABLE statement to define and generate manually the ranges specified for a data partition.

Example 4:

This statement partitions on two date columns both of which are generated. Notice the use of the automatically generated form of the CREATE TABLE syntax and that only one end of each range is specified. The other end is implied from the adjacent data partition and the use of the INCLUSIVE option:

```
CREATE TABLE sales(invoice_date date, inv_month int NOT NULL
GENERATED ALWAYS AS (month(invoice_date)), inv_year INT NOT
NULL GENERATED ALWAYS AS ( year(invoice_date)),
item_id int NOT NULL,
cust_id int NOT NULL) PARTITION BY RANGE (inv_year,
inv_month)
(PART Q1_02 STARTING (2002,1) ENDING (2002, 3) INCLUSIVE,
PART Q2_02 ENDING (2002, 6) INCLUSIVE,
PART Q3_02 ENDING (2002, 9) INCLUSIVE,
PART Q4_02 ENDING (2002,12) INCLUSIVE,
PART CURRENT ENDING (MAXVALUE, MAXVALUE));
```

Gaps in the ranges are permitted. The CREATE TABLE syntax supports gaps by allowing you to specify a STARTING value for a range that does not line up against the ENDING value of the previous data partition.

Example 5:

Creates a table with a gap between values 101 and 200.

```
CREATE TABLE foo(a INT)
    PARTITION BY RANGE(a)
   (STARTING FROM (1) ENDING AT (100),
   STARTING FROM (201) ENDING AT (300))
```

Use of the ALTER TABLE statement, which allows data partitions to be added or removed, can also cause gaps in the ranges.

When you insert a row into a partitioned table, it is automatically placed into the proper data partition based on its key value and the range it falls within. If it falls outside of any ranges defined for the table, the insert fails and the following error is returned to the application:

```
SQL0327N  The row cannot be inserted into table <tablename>
because it is outside the bounds of the defined data partition ranges.
   SQLSTATE=22525
```

Restrictions

- Table level restrictions:
  - Tables created using the automatically generated form of the syntax (containing the EVERY clause) are constrained to use a numeric or date time type in the table partitioning key.
- Statement level restrictions:
  - MINVALUE and MAXVALUE are not supported in the automatically generated form of the syntax.
  - Ranges are ascending.
  - Only one column can be specified in the automatically generated form of the syntax.
  - The increment in the EVERY clause must be greater than zero.
  - The ENDING value must be greater than or equal to the STARTING value.

Placement of the data, index and long data of a data partition
By its very nature, creating a partitioned table allows you to place the various parts of the table and the associated table objects in specific table spaces.

When creating a table you can specify in which table space the entire table data and associated table objects will be placed. Or, you can place the table's index, long or large data, or table partitions in specific table spaces. All of the table spaces must be in the same database partition group.

The CREATE TABLE statement has the following clauses which demonstrate this ability to place the table data and associated table objects within specific table spaces:

```
CREATE TABLE table_name IN table_space_name1
   INDEX IN table_space_name2
   LONG IN table_space_name3
   PARTITIONED BY ...
     PARTITION partition_name | boundary specification | IN table_space_name4
       INDEX IN table_space_name5
       LONG IN table_space_name6
```

Each of the partitions of the partitioned table can be placed in different table spaces.

You can also specify the table space for a user-created nonpartitioned index on a partitioned table using the CREATE INDEX ... IN *table_space_name1* statement, which can be different from the index table space specified in the CREATE TABLE ... INDEX IN *table_space_name2* statement. The IN clause of the CREATE INDEX statement is used for partitioned tables only. If the INDEX IN clause is not specified on the CREATE TABLE or CREATE INDEX statements, the index is placed in the same table space as the first visible or attached partition of the table.

System generated nonpartitioned indexes, such as XML column paths indexes, are placed in the table space specified in the INDEX IN clause of the CREATE TABLE statement.

On a partitioned table with XML data, the XML region index is always partitioned in the same way as the table data. The table space of the partitioned indexes is defined at the partition level

XML data resides in the table spaces used by the long data for a table. XML data placement on a partitioned table follows the long data placement rules.

The table space for long data can be specified explicitly by you or determined by the database manager implicitly. For a partitioned table, the table level LONG IN clause can be used together with the partition level LONG IN clause. If both are specified, the partition level LONG IN clause takes precedence over any table level LONG IN clauses.

Migrating existing tables and views to partitioned tables
You can migrate a nonpartitioned table or a UNION ALL view to an empty partitioned table.

Before you begin

Attaching a data partition is not allowed if SYSCAT.COLUMNS.IMPLICITVALUE for a specific column is a nonnull value for both the source column and the target column, and the values do not match. In this case, you must drop the source table and then recreate it.

A column can have a nonnull value in the SYSCAT.COLUMNS IMPLICITVALUE field if any one of the following conditions is met:

- The IMPLICITVALUE field is propagated from a source table during an attach operation.
- The IMPLICITVALUE field is inherited from a source table during a detach operation.
- The IMPLICITVALUE field is set during migration from V8 to V9, where it is determined to be an added column, or might be an added column. An added column is a column that is created as the result of an ALTER TABLE...ADD COLUMN statement.

Always create the source and target tables involved in an attach operation with the same columns defined. In particular, never use the ALTER TABLE statement to add columns to the target table of an attach operation.

For advice on avoiding a mismatch when working with partitioned tables, see “Guidelines for attaching data partitions to partitioned tables” on page 168.

About this task

When migrating regular tables, unload the source table by using the **EXPORT** command or high performance unload. Create a new, empty partitioned table, and use the **LOAD** command to populate that partitioned table. To move the data from the old table directly into the partitioned table without any intermediate steps, use the **LOAD FROM CURSOR** command (see Step 1.

You can convert nonpartitioned data in a UNION ALL view to a partitioned table (see Step 2). UNION ALL views are used to manage large tables and achieve easy roll-in and roll-out of table data while providing the performance advantages of branch elimination. Using the ALTER TABLE...ATTACH PARTITION statement, you can achieve conversion with no movement of data in the base table. Nonpartitioned indexes and dependent views or materialized query tables (MQTs) must be recreated after the conversion. The recommended strategy to convert UNION ALL views to partitioned tables is to create a partitioned table with a single dummy data partition, then attach all of the tables of the union all view. Be sure to drop the dummy data partition early in the process to avoid problems with overlapping ranges.

Procedure

1. Migrate a regular table to a partitioned table.

   Use the **LOAD FROM CURSOR** command to avoid any intermediate steps. The following example shows how to migrate table T1 to the SALES_DP table.

   a. Create and populate a regular table T1.

   ```
   CREATE TABLE t1 (c1 int, c2 int);
   INSERT INTO t1 VALUES (0,1), (4, 2), (6, 3);
   ```

   b. Create an empty partitioned table.

   ```
   CREATE TABLE sales_dp (c1 int, c2 int)
     PARTITION BY RANGE (c1)
     (STARTING FROM 0 ENDING AT 10 EVERY 2);
   ```

   c. Use the **LOAD FROM CURSOR** command to pull the data from an SQL query directly into the new partitioned table.

   ```
      SELECT * FROM t1;
      DECLARE c1 CURSOR FOR SELECT * FROM t1;
      LOAD FROM c1 of CURSOR INSERT INTO sales_dp;SELECT * FROM sales_dp;
   ```

2. Convert nonpartitioned data in a UNION ALL view to a partitioned table.

   The following example shows how to convert the UNION ALL view named ALL_SALES to the SALES_DP table.

   a. Create the UNION ALL view.

   ```
       CREATE VIEW all_sales AS
           (
           SELECT * FROM sales_0198
           WHERE sales_date BETWEEN '01-01-1998' AND '01-31-1998'
           UNION ALL
           SELECT * FROM sales_0298
           WHERE sales_date BETWEEN '02-01-1998' AND '02-28-1998'
           UNION ALL
           ...
           UNION ALL
           SELECT * FROM sales_1200
           WHERE sales_date BETWEEN '12-01-2000' AND '12-31-2000'
           );
   ```

   b. Create a partitioned table with a single dummy partition. Choose the range so that it does not overlap with the first data partition to be attached.

```
    CREATE TABLE sales_dp (
        sales_date DATE NOT NULL,
        prod_id INTEGER,
        city_id INTEGER,
        channel_id INTEGER,
        revenue DECIMAL(20,2))
        PARTITION BY RANGE (sales_date)
        (PART dummy STARTING FROM '01-01-1900' ENDING AT '01-01-1900');
```

c. Attach the first table.

```
ALTER TABLE sales_dp ATTACH PARTITION
  STARTING FROM '01-01-1998' ENDING AT '01-31-1998'
  FROM sales_0198;
```

d. Drop the dummy partition.

```
ALTER TABLE sales_dp DETACH PARTITION dummy
  INTO dummy;
DROP TABLE dummy;
```

e. Attach the remaining partitions.

```
ALTER TABLE sales_dp ATTACH PARTITION
  STARTING FROM '02-01-1998' ENDING AT '02-28-1998'
  FROM sales_0298;
...
ALTER TABLE sales_dp ATTACH PARTITION
  STARTING FROM '12-01-2000' ENDING AT '12-31-2000'
  FROM sales_1200;
```

f. Issue the SET INTEGRITY statement to make data in the newly attached partition accessible to queries.

```
SET INTEGRITY FOR sales_dp IMMEDIATE CHECKED
FOR EXCEPTION IN sales_dp USE sales_ex;
```

Tip: If data integrity checking, including range validation and other constraints checking, can be done through application logic that is independent of the data server before an attach operation, newly attached data can be made available for use much sooner. You can optimize the data roll-in process by using the SET INTEGRITY...ALL IMMEDIATE UNCHECKED statement to skip range and constraints violation checking. In this case, the table is brought out of SET INTEGRITY pending state, and the new data is available for applications to use immediately, as long as there are no nonpartitioned user indexes on the target table.

g. Create indexes, as appropriate.

Converting existing indexes to partitioned indexes

System-created and user-created indexes might need to be migrated from nonpartitioned to partitioned. User-created indexes can be converted while maintaining availability to the table and indexes for most of the migration. System-created indexes used to enforce primary key constraints or unique constraints will not be able to have the constraints maintained while the conversion is done.

Before you begin

Indexes created in an earlier release of the product might be nonpartitioned. This could include both indexes created by you, or system-created indexes created by the database manager. Examples of system-created indexes are indexes to enforce unique and primary constraints and the block indexes of an MDC table.

About this task

Indexes created by you can be converted from nonpartitioned to partitioned while having continuous availability to the data using the index. You can create a partitioned index with the same keys as the corresponding nonpartitioned index. While the partitioning index is created, you can still use the current indexes and the table where the index is being created. Once the partitioned index is created, you can drop the corresponding nonpartitioned index and rename the new partitioned index if desired.

Results
The following examples demonstrate how to convert existing nonpartitioned indexes into partitioned indexes.

Example

Here is an example of converting a nonpartitioned index created by you to one that is a partitioned index:

```
UPDATE COMMAND OPTIONS USING C OFF;
CREATE INDEX data_part ON sales(sale_date) PARTITIONED;
DROP INDEX dateidx;
RENAME INDEX data_part TO dateidx;
COMMIT;
```

Here is an example of converting a nonpartitioned index created by the database manager to one that is a partitioned index. In this case, there will be a period of time between the dropping of the original constraint, and the creation of the new constraint.

```
ALTER TABLE employees DROP CONSTRAINT emp_uniq;
ALTER TABLE employees ADD CONSTRAINT emp_uniq UNIQUE (employee_id);
```

MDC tables created using Db2 Version 9.7 and earlier releases have nonpartitioned block indexes. To take advantage of partitioned table data availability features such as data roll in and roll out and partition level reorganization of table data and indexes, the data in the multidimensional clustering (MDC) table created using Db2 V9.7 and earlier releases must be moved to a partitioned MDC table with partitioned block indexes created using Db2 V9.7 Fix Pack 1 or a later release.

Online move of a partitioned MDC table to use partitioned block indexes

You can move data from a MDC table with nonpartitioned block indexes to an MDC table with partitioned block indexes using an online table move.

In the following example, `company1.parts` table has **`region`** and **`color`** as the MDC key columns; and the corresponding block indexes are nonpartitioned.

```
CALL SYSPROC.ADMIN_MOVE_TABLE(
 'COMPANY1',  --Table schema
 'PARTS',     --Table name
 ' ',         --null; No change to columns definition
 ' ',         --null; No additional options
 'MOVE');     --Move the table in one step
```

Offline move of a partitioned MDC table to use partitioned block indexes

To minimize data movement, you can move data from a MDC table with nonpartitioned block indexes to an MDC table with partitioned block indexes when the table is offline. The process uses the following steps:

1. Create a new, single-partition MDC table with the same definition as the table to be converted. When specifying the range for the partition, use a range outside the ranges of the partitioned MDC table to be converted.

   The block indexes of new, single-partition MDC table are partitioned. The partition created when specifying the range is detached in a later step.
2. Detach each partition of the MDC table. Each partition becomes a stand-alone MDC table.

   When a partition is detached, the partition data is attached to a new, target table without moving the data in the partition.

   Note: The last partition of the MDC table cannot be detached. It is a single-partition MDC table with nonpartitioned block indexes.
3. For each stand-alone table created by detaching the MDC table partitions, and the single-partition MDC table with nonpartitioned block indexes, attach the table to the new partitioned MDC table created in Step 1.

When the table is attached, the table data is attached to the new partitioned MDC table without moving the data, and the block indexes are created as partitioned block indexes.

4. After attaching the first stand-alone MDC table, you can detach the empty partition created when you created the new MDC table.
5. Issue SET INTEGRITY statement on the new partitioned MDC table.

What to do next

Partitioned materialized query table (MQT) behavior

All types of materialized query tables (MQTs) are supported with partitioned tables. When working with partitioned MQTs, there are a number of guidelines that can help you to administer attached and detached data partitions most effectively.

The following guidelines and restrictions apply when working with partitioned MQTs or partitioned tables with detached dependent tables:

- If you issue an ALTER TABLE ... DETACH PARTITION statement, the DETACH operation creates the target table for the detached partition data. If there are any dependent tables that need to be incrementally maintained with respect to the detached data partition (these dependent tables are referred to as detached dependent tables), the SET INTEGRITY statement is required to be run on the detached dependent tables to incrementally maintain the tables. With Db2 V9.7 Fix Pack 1 or later releases, after the SET INTEGRITY statement is run on all detached dependent tables, the asynchronous partition detach task makes the data partition into a stand-alone target table. Until the asynchronous partition detach operation completes, the target table is unavailable. The target table will be marked 'L' in the TYPE column of the SYSCAT.TABLES catalog view. This is referred to as a detached table. This prevents the target table from being read, modified or dropped until the SET INTEGRITY statement is run to incrementally maintain the detached dependent tables. After the SET INTEGRITY statement is run on all detached dependent tables, the data partition is logically detached from the source table and the asynchronous partition detach operation detaches data partition from the source table into the target table. Until the asynchronous partition detach operation completes, the target table is unavailable.
- To detect that a detached table is not yet accessible, query the SYSCAT.TABDETACHEDDEP catalog view. If any inaccessible detached tables are detected, run the SET INTEGRITY statement with the IMMEDIATE CHECKED option on all the detached dependent tables to transition the detached table to a regular accessible table. If you try to access a detached table before all its detached dependents are maintained, error code SQL20285N is returned.
- The DATAPARTITIONNUM function cannot be used in an materialized query table (MQT) definition. Attempting to create an MQT using this function returns an error (SQLCODE SQL20058N, SQLSTATE 428EC).
- When creating a nonpartitioned index on a table with detached data partitions with STATUS 'D' in SYSCAT.DATAPARTITIONS, the index does not include the data in the detached data partitions unless the detached data partition has a dependent materialized query table (MQT) that needs to be incrementally refreshed with respect to it. In this case, the index includes the data for this detached data partition.
- Altering a table with attached data partitions to an MQT is not allowed.
- Partitioned staging tables are not supported.
- Attaching to an MQT is not directly supported. See Example 1 for details.

Example 1: Converting a partitioned MQT to an ordinary table

Although the ATTACH operation is not directly supported on partitioned MQTs, you can achieve the same effect by converting a partitioned MQT to an ordinary table, performing the desired roll-in and roll-out of table data, and then converting the table back into an MQT. The following CREATE TABLE and ALTER TABLE statements demonstrate the effect:

```
CREATE TABLE lineitem (
  l_orderkey    DECIMAL(10,0) NOT NULL,
```

```
  l_quantity    DECIMAL(12,2),
  l_shipdate    DATE,
  l_year_month  INT GENERATED ALWAYS AS (YEAR(l_shipdate)*100 + MONTH(l_shipdate)))
    PARTITION BY RANGE(l_shipdate)
    (STARTING ('1/1/1992') ENDING ('12/31/1993') EVERY 1 MONTH);
CREATE TABLE lineitem_ex (
  l_orderkey    DECIMAL(10,0) NOT NULL,
  l_quantity    DECIMAL(12,2),
  l_shipdate    DATE,
  l_year_month  INT,
  ts            TIMESTAMP,
  msg           CLOB(32K));

CREATE TABLE quan_by_month (
  q_year_month, q_count) AS
    (SELECT l_year_month AS q_year_month, COUNT(*) AS q_count
     FROM lineitem
     GROUP BY l_year_month)
     DATA INITIALLY DEFERRED REFRESH IMMEDIATE
     PARTITION BY RANGE(q_year_month)
     (STARTING (199201) ENDING (199212) EVERY (1),
      STARTING (199301) ENDING (199312) EVERY (1));
CREATE TABLE quan_by_month_ex(
  q_year_month  INT,
  q_count       INT NOT NULL,
  ts            TIMESTAMP,
  msg           CLOB(32K));

SET INTEGRITY FOR quan_by_month IMMEDIATE CHECKED;
CREATE INDEX qbmx ON quan_by_month(q_year_month);

ALTER TABLE quan_by_month DROP MATERIALIZED QUERY;
ALTER TABLE lineitem DETACH PARTITION part0 INTO li_reuse;
ALTER TABLE quan_by_month DETACH PARTITION part0 INTO qm_reuse;

SET INTEGRITY FOR li_reuse OFF;
ALTER TABLE li_reuse ALTER l_year_month SET GENERATED ALWAYS
AS (YEAR(l_shipdate)*100 + MONTH(l_shipdate));

LOAD FROM part_mqt_rotate.del OF DEL MODIFIED BY GENERATEDIGNORE
MESSAGES load.msg REPLACE INTO li_reuse;

DECLARE load_cursor CURSOR FOR
  SELECT l_year_month, COUNT(*)
    FROM li_reuse
    GROUP BY l_year_month;
LOAD FROM load_cursor OF CURSOR MESSAGES load.msg
  REPLACE INTO qm_reuse;

ALTER TABLE lineitem ATTACH PARTITION STARTING '1/1/1994'
  ENDING '1/31/1994' FROM li_reuse;

SET INTEGRITY FOR lineitem ALLOW WRITE ACCESS IMMEDIATE CHECKED
FOR EXCEPTION IN lineitem USE lineitem_ex;

ALTER TABLE quan_by_month ATTACH PARTITION STARTING 199401
  ENDING 199401 FROM qm_reuse;

SET INTEGRITY FOR quan_by_month IMMEDIATE CHECKED
  FOR EXCEPTION IN quan_by_month USE quan_by_month_ex;

ALTER TABLE quan_by_month ADD MATERIALIZED QUERY
  (SELECT l_year_month AS q_year_month, COUNT(*) AS q_count
     FROM lineitem
       GROUP BY l_year_month)
         DATA INITIALLY DEFERRED REFRESH IMMEDIATE;

SET INTEGRITY FOR QUAN_BY_MONTH ALL IMMEDIATE UNCHECKED;
```

Use the SET INTEGRITY statement with the IMMEDIATE CHECKED option to check the attached data partition for integrity violations. This step is required before changing the table back to an MQT. The SET INTEGRITY statement with the IMMEDIATE UNCHECKED option is used to bypass the required full refresh of the MQT. The index on the MQT is necessary to achieve optimal performance. The use of exception tables with the SET INTEGRITY statement is recommended, where appropriate.

Typically, you create a partitioned MQT on a large fact table that is also partitioned. If you do roll out or roll in table data on the large fact table, you must adjust the partitioned MQT manually, as demonstrated in Example 2.

Example 2: Adjusting a partitioned MQT manually

Alter the MQT (quan_by_month) to convert it to an ordinary partitioned table:

```
ALTER TABLE quan_by_month DROP MATERIALIZED QUERY;
```

Detach the data to be rolled out from the fact table (lineitem) and the MQT and re-load the staging table li_reuse with the new data to be rolled in:

```
ALTER TABLE lineitem DETACH PARTITION part0 INTO li_reuse;
```

```
LOAD FROM part_mqt_rotate.del OF DEL MESSAGES load.msg REPLACE INTO li_reuse;
```

```
ALTER TABLE quan_by_month DETACH PARTITION part0 INTO qm_reuse;
```

Prune qm_reuse before doing the insert. This deletes the detached data before inserting the subselect data. This is accomplished with a load replace into the MQT where the data file of the load is the content of the subselect.

```
db2 load from datafile.del of del replace into qm_reuse
```

You can refresh the table manually using INSERT INTO ... (SELECT ...) This is only necessary on the new data, so the statement should be issued before attaching:

```
INSERT INTO qm_reuse
  (SELECT COUNT(*) AS q_count, l_year_month AS q_year_month
       FROM li_reuse
         GROUP BY l_year_month);
```

Now you can roll in the new data for the fact table:

```
ALTER TABLE lineitem ATTACH PARTITION STARTING '1/1/1994'
ENDING '1/31/1994' FROM TABLE li_reuse;
SET INTEGRITY FOR lineitem ALLOW WRITE ACCESS IMMEDIATE CHECKED FOR
EXCEPTION IN li_reuse USE li_reuse_ex;
```

Next, roll in the data for the MQT:

```
ALTER TABLE quan_by_month ATTACH PARTITION STARTING 199401
ENDING 199401 FROM TABLE qm_reuse;
SET INTEGRITY FOR quan_by_month IMMEDIATE CHECKED;
```

After attaching the data partition, the new data must be verified to ensure that it is in range.

```
ALTER TABLE quan_by_month ADD MATERIALIZED QUERY
  (SELECT COUNT(*) AS q_count, l_year_month AS q_year_month
     FROM lineitem
       GROUP BY l_year_month)
         DATA INITIALLY DEFERRED REFRESH IMMEDIATE;
SET INTEGRITY FOR QUAN_BY_MONTH ALL IMMEDIATE UNCHECKED;
```

The data is not accessible until it has been validated by the SET INTEGRITY statement. Although the REFRESH TABLE operation is supported, this scenario demonstrates the manual maintenance of a partitioned MQT through the ATTACH PARTITION and DETACH PARTITION operations. The data is marked as validated by the user through the IMMEDIATE UNCHECKED clause of the SET INTEGRITY statement.

Creating range-clustered tables

Guidelines for using range-clustered tables

This topic lists some guidelines to follow when working with range-clustered tables (RCT).

- Because the process of creating a range-clustered table pre-allocates the required disk space, that space must be available.

- When defining the range of key values, the minimum value is optional; if it is not specified, the default is 1. A negative minimum value must be specified explicitly. For example:

```
ORGANIZE BY KEY SEQUENCE (f1 STARTING FROM -100 ENDING AT -10)
```

- You cannot create a regular index on the same key values that are used to define the range-clustered table.
- ALTER TABLE statement options that affect the physical structure of the table are not allowed.

Scenarios: Range-clustered tables
Range-clustered tables can have single-column or multiple-column keys, and can allow or disallow rows with key values that are outside of the defined range of values. This section contains scenarios that illustrate how such tables can be created.

Scenario 1: Creating a range-clustered table (overflow allowed)

The following example shows a range-clustered table that can be used to retrieve information about a specific student. Each student record contains the following information:

- School ID
- Program ID
- Student number
- Student ID
- Student first name
- Student last name
- Student grade point average (GPA)

```
CREATE TABLE students (
 school_id      INT NOT NULL,
 program_id     INT NOT NULL,
 student_num    INT NOT NULL,
 student_id     INT NOT NULL,
 first_name     CHAR(30),
 last_name      CHAR(30),
 gpa            FLOAT
)
 ORGANIZE BY KEY SEQUENCE
  (student_id STARTING FROM 1 ENDING AT 1000000)
  ALLOW OVERFLOW
;
```

In this example, the STUDENT_ID column, which serves as the table key, is used to add, update, or delete student records.

The size of each record is based on the sum of the column lengths. In this example, each record is 97 bytes long (10-byte header + 4 + 4 + 4 + 4 + 30 + 30 + 8 + 3 bytes for nullable columns). With a 4-KB (or 4096-byte) page size, after accounting for overhead, there are 4038 bytes (enough for 41 records) available per page. A total of 24391 such pages is needed to accommodate 1 million student records. Assuming four pages for table overhead and three pages for extent mapping, 24384 4-KB pages would be pre-allocated when this table is created. (The extent mapping assumes a single three-page container for the table.)

Scenario 2: Creating a range-clustered table (overflow not allowed)

In the following example, a school board administers 200 schools, each having 20 classrooms with a capacity of 35 students per classroom. This school board can accommodate a maximum of 140,000 students.

```
CREATE TABLE students (
 school_id      INT NOT NULL,
 class_id       INT NOT NULL,
 student_num    INT NOT NULL,
 student_id     INT NOT NULL,
```

```
  first_name     CHAR(30),
  last_name      CHAR(30),
  gpa            FLOAT
)
  ORGANIZE BY KEY SEQUENCE
   (school_id STARTING FROM 1 ENDING AT 200,
    class_id STARTING FROM 1 ENDING AT 20,
    student_num STARTING FROM 1 ENDING AT 35)
   DISALLOW OVERFLOW
;
```

In this example, the SCHOOL_ID, CLASS_ID, and STUDENT_NUM columns together serve as the table key, which is used to add, update, or delete student records.

Overflow is not allowed, because school board policy restricts the number of students in each classroom, and there is a fixed number of schools and classrooms being administered by this school board. Some smaller schools (schools with fewer classrooms than the largest school) will have pre-allocated space in the table that will likely never be used.

Considerations when creating MDC or ITC tables

There are many factors to consider when creating MDC or ITC tables. Decisions on how to create, place, and use your MDC or ITC tables can be influenced by your current database environment (for example, whether you have a partitioned database or not), and by your choice of dimensions.

Moving data from existing tables to MDC tables

To improve query performance and reduce the requirements of data maintenance operations in a data warehouse or large database environment, you can move data from regular tables into multidimensional clustering (MDC) tables. To move data from an existing table to an MDC table:

1. export your data,
2. drop the original table (optional),
3. create a multidimensional clustering (MDC) table (using the CREATE TABLE statement with the ORGANIZE BY DIMENSIONS clause),
4. load the MDC table with your data.

An ALTER TABLE procedure called SYSPROC.ALTOBJ can be used to carry out the translation of data from an existing table to an MDC table. The procedure is called from the Db2 Design Advisor. The time required to translate the data between the tables can be significant and depends on the size of the table and the amount of data that needs to be translated.

The ALTOBJ procedure runs the following steps when altering a table:

1. drop all dependent objects of the table,
2. rename the table,
3. create the table with the new definition,
4. recreate all dependent objects of the table,
5. transform existing data in the table into the data required in the new table. That is, the selecting of data from the old table and loading that data into the new one where column functions can be used to transform from an old data type to a new data type.

Moving data from existing tables to ITC tables

To reduce the requirements of data maintenance operations, you can move data from regular tables into insert time clustering (ITC) tables. To move data from an existing table to an ITC table use the online table move stored procedure.

The ExampleBank scenario shows how data from an existing table is moved into an ITC table. The scenario also shows how convenient reclaiming space is when using ITC tables. For more information, see the Related concepts links.

MDC Advisor feature on the Db2 Design Advisor

The Db2 Design Advisor (**db2advis**) has an MDC feature. This feature recommends clustering dimensions for use in an MDC table, including coarsifications on base columns in order to improve workload performance. The term *coarsification* refers to a mathematical expression to reduce the cardinality (the number of distinct values) of a clustering dimension. A common example is coarsification by date, week of the date, month of the date, or quarter of the year.

A requirement to use the MDC feature of the Db2 Design Advisor is the existence of at least several extents of data within the database. The Db2 Design Advisor uses the data to model data density and cardinality.

If the database does not have data in the tables, the Db2 Design Advisor does not recommend MDC, even if the database contains empty tables but has a mocked up set of statistics to imply a populated database.

The recommendation includes identifying potential generated columns that define coarsification of dimensions. The recommendation does not include possible block sizes. The extent size of the table space is used when making recommendations for MDC tables. The assumption is that the recommended MDC table is created in the same table space as the existing table, and therefore has the same extent size. The recommendations for MDC dimensions change depending on the extent size of the table space, because the extent size affects the number of records that can fit into a block or cell. The extent size directly affects the density of the cells.

Only single-column dimensions, and not composite-column dimensions, are considered, although single or multiple dimensions might be recommended for the table. The MDC feature recommends coarsifications for most supported data types with the goal of reducing the cardinality of cells in the resulting MDC solution. The data type exceptions include: CHAR, VARCHAR, GRAPHIC, and VARGRAPHIC data types. All supported data types are cast to INTEGER and are coarsified through a generated expression.

The goal of the MDC feature of the Db2 Design Advisor is to select MDC solutions that result in improved performance. A secondary goal is to keep the storage expansion of the database constrained to a modest level. A statistical method is used to determine the maximum storage expansion on each table.

The analysis operation within the advisor includes not only the benefits of block index access but also the effect of MDC on insert, update, and delete operations against dimensions of the table. These actions on the table have the potential to cause records to be moved between cells. The analysis operation also models the potential performance effect of any table expansion resulting from the organization of data along particular MDC dimensions.

The MDC feature is run by using the `-m <advise type>` flag on the db2advis utility. The "C" advise type is used to indicate multidimensional clustering tables. The advise types are: "I" for index, "M" for materialized query tables, "C" for MDC, and "P" for partitioned database environment. The advise types can be used in combination with each other.

Note: The Db2 Design Advisor does not explore tables that are less than 12 extents in size.

The advisor analyzes both MQTs and regular base tables when coming up with recommendations.

The output from the MDC feature includes:

- Generated column expressions for each table for coarsified dimensions that appear in the MDC solution.
- An ORGANIZE BY DIMENSIONS clause recommended for each table.

The recommendations are reported both to stdout and to the ADVISE tables that are part of the explain facility.

MDC tables and partitioned database environments

Multidimensional clustering can be used in a partitioned database environment. In fact, MDC can complement a partitioned database environment. A partitioned database environment is used to distribute data from a table across multiple physical or logical database partitions to:

- take advantage of multiple machines to increase processing requests in parallel,
- increase the physical size of the table beyond the limits of a single database partition,
- improve the scalability of the database.

The reason for distributing a table is independent of whether the table is an MDC table or a regular table. For example, the rules for the selection of columns to make up the distribution key are the same. The distribution key for an MDC table can involve any column, whether those columns make up part of a dimension of the table or not.

If the distribution key is identical to a dimension from the table, then each database partition contains a different portion of the table. For instance, if our example MDC table is distributed by color across two database partitions, then the Color column is used to divide the data. As a result, the Red and Blue slices might be found on one database partition and the Yellow slice on the other. If the distribution key is not identical to the dimensions from the table, then each database partition has a subset of data from each slice. When choosing dimensions and estimating cell occupancy, note that on average the total amount of data per cell is determined by taking all of the data and dividing by the number of database partitions.

MDC tables with multiple dimensions

If you know that certain predicates are heavily used in queries, you can cluster the table on the columns involved. You can do this by using the ORGANIZE BY DIMENSIONS clause.

Example 1:

```
CREATE TABLE T1 (c1 DATE, c2 INT, c3 INT, c4 DOUBLE)
   ORGANIZE BY DIMENSIONS (c1, c3, c4)
```

The table in Example 1 is clustered on the values within three columns forming a logical cube (that is, having three dimensions). The table can now be logically sliced up during query processing on one or more of these dimensions such that only the blocks in the appropriate slices or cells are processed by the relational operators involved. The size of a block (the number of pages) is the extent size of the table.

MDC tables with dimensions based on more than one column

Each dimension can be made up of one or more columns. As an example, you can create a table that is clustered on a dimension containing two columns.

Example 2:

```
CREATE TABLE T1 (c1 DATE, c2 INT, c3 INT, c4 DOUBLE)
   ORGANIZE BY DIMENSIONS (c1, (c3, c4))
```

In Example 2, the table is clustered on two dimensions, c1 and (c3,c4). Thus, in query processing, the table can be logically sliced up on either the c1 dimension, or on the composite (c3, c4) dimension. The table has the same number of blocks as the table in Example 1, but one less dimension block index. In Example 1, there are three dimension block indexes, one for each of the columns c1, c3, and c4. In Example 2, there are two dimension block indexes, one on the column c1 and the other on the columns c3 and c4. The main difference between the two approaches is that, in Example 1, queries involving c4 can use the dimension block index on c4 to quickly and directly access blocks of relevant data. In Example 2, c4 is a second key part in a dimension block index, so queries involving c4 involve more processing. However, in Example 2 there is one less block index to maintain and store.

The Db2 Design Advisor does not make recommendations for dimensions containing more than one column.

MDC tables with column expressions as dimensions

Column expressions can also be used for clustering dimensions. The ability to cluster on column expressions is useful for rolling up dimensions to a coarser granularity, such as rolling up an address to a geographic location or region, or rolling up a date to a week, month, or year. To implement the rolling up of dimensions in this way, you can use generated columns. This type of column definition allows the

creation of columns using expressions that can represent dimensions. In Example 3, the statement creates a table clustered on one base column and two column expressions.

Example 3:

```
CREATE TABLE T1(c1 DATE, c2 INT, c3 INT, c4 DOUBLE,
   c5 DOUBLE GENERATED ALWAYS AS (c3 + c4),
   c6 INT GENERATED ALWAYS AS (MONTH(C1)))
      ORGANIZE BY DIMENSIONS (c2, c5, c6)
```

In Example 3, column c5 is an expression based on columns c3 and c4, and column c6 rolls up column c1 to a coarser granularity in time. The statement clusters the table based on the values in columns c2, c5, and c6.

Range queries on generated column dimensions

Range queries on a generated column dimension require monotonic column functions. Expressions must be monotonic to derive range predicates for dimensions on generated columns. If you create a dimension on a generated column, queries on the base column are able to take advantage of the block index on the generated column to improve performance, with one exception. For range queries on the base column (date, for example) to use a range scan on the dimension block index, the expression used to generate the column in the CREATE TABLE statement must be monotonic. Although a column expression can include any valid expression (including user-defined functions (UDFs)), if the expression is non-monotonic, only equality or IN predicates are able to use the block index to satisfy the query when these predicates are on the base column.

As an example, assume that you create an MDC table with dimensions on the generated column month, where `month = INTEGER (date)/100`. For queries on the dimension (month), block index scans can be done. For queries on the base column (date), block index scans can also be done to narrow down which blocks to scan, and then apply the predicates on date to the rows in those blocks only.

The compiler generates additional predicates to be used in the block index scan. For example, with the query:

```
   SELECT * FROM MDCTABLE WHERE DATE > "1999-03-03" AND DATE < "2000-01-15"
```

the compiler generates the additional predicates: "month >= 199903" and "month <= 200001" which can be used as predicates for a dimension block index scan. When scanning the resulting blocks, the original predicates are applied to the rows in the blocks.

A non-monotonic expression allows equality predicates to be applied to that dimension. A good example of a non-monotonic function is MONTH( ) as seen in the definition of column c6 in Example 3. If the c1 column is a date, timestamp, or valid string representation of a date or timestamp, then the function returns an integer value in the range of 1 to 12. Even though the output of the function is deterministic, it actually produces output similar to a step function (that is, a cyclic pattern):

```
MONTH(date('01/05/1999')) = 1
MONTH(date('02/08/1999')) = 2
MONTH(date('03/24/1999')) = 3
MONTH(date('04/30/1999')) = 4
...
MONTH(date('12/09/1999')) = 12
MONTH(date('01/18/2000')) = 1
MONTH(date('02/24/2000')) = 2
...
```

Although date in this example is continually increasing, MONTH(date) is not. More specifically, it is not guaranteed that whenever date1 is larger than date2, MONTH(date1) is greater than or equal to MONTH(date2). It is this condition that is required for monotonicity. This non-monotonicity is allowed, but it limits the dimension in that a range predicate on the base column cannot generate a range predicate on the dimension. However, a range predicate on the expression is fine, for example, `where month(c1) between 4 and 6`. This can use the index on the dimension in the typical way, with a starting key of 4 and a stop key of 6.

To make this function monotonic, include the year as the high-order part of the month. There is an extension to the INTEGER built-in function to help in defining a monotonic expression on date. INTEGER(date) returns an integer representation of the date, which then can be divided to find an integer representation of the year and month. For example, `INTEGER(date('2000/05/24'))` returns `20000524`, and therefore `INTEGER(date('2000/05/24'))/100 = 200005`. The function INTEGER(date)/100 is monotonic.

Similarly, the built-in functions DECIMAL and BIGINT also have extensions so that you can derive monotonic functions. DECIMAL(timestamp) returns a decimal representation of a timestamp, and this can be used in monotonic expressions to derive increasing values for month, day, hour, minute, and so on. BIGINT(date) returns a big integer representation of the date, similar to INTEGER(date).

The database manager determines the monotonicity of an expression, where possible, when creating the generated column for the table, or when creating a dimension from an expression in the dimensions clause. Certain functions can be recognized as monotonicity-preserving, such as DAYS( ) or YEAR( ). Also, various mathematical expressions such as division, multiplication, or addition of a column and a constant are monotonicity-preserving. Where Db2 determines that an expression is not monotonicity-preserving, or if it cannot determine this, the dimension supports only the use of equality predicates on its base column.

Altering a database

Altering an instance

Changing the database configuration across multiple database partitions

When you have a database that is distributed across more than one database partition, the database configuration file should be the same on all database partitions.

About this task

Consistency is required since the SQL compiler compiles distributed SQL statements based on information in the database partition configuration file and creates an access plan to satisfy the needs of the SQL statement. Maintaining different configuration files on database partitions could lead to different access plans, depending on which database partition the statement is prepared. Use **db2_all** to maintain the configuration files across all database partitions.

Altering a database

Altering tables and other related table objects

Altering partitioned tables

All relevant clauses of the ALTER TABLE statement are supported for a partitioned table. In addition, the ALTER TABLE statement allows you to add new data partitions, roll-in (*attach*) new data partitions, and roll-out (*detach*) existing data partitions.

Before you begin

To alter a partitioned table to detach a data partition the user must have the following authorities or privileges:

- The user performing the DETACH PARTITION operation must have the authority necessary to ALTER, to SELECT from, and to DELETE from the source table.
- The user must also have the authority necessary to create the target table. Therefore, to alter a table to detach a data partition, the privilege held by the authorization ID of the statement must include at least one of the following authorities or privileges on the target table:
  - DBADM authority

– CREATETAB authority on the database and USE privilege on the table spaces used by the table as well as one of:
  - IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the table does not exist
  - CREATEIN privilege on the schema, if the schema name of the table refers to an existing schema.

To alter a partitioned table to attach a data partition, the privileges held by the authorization ID of the statement must include at least one of the following authorities or privileges on the source table:

- DATAACCESS authority or SELECT privilege on the source table and DBADM authority or DROPIN privilege on the schema of the source table
- INSERT privilege on target table
- CONTROL privilege on the source table

To alter a partitioned table to add a data partition, the privileges held by the authorization ID of the statement must have privileges to use the table space where the new partition is added, and include at least one of the following authorities or privileges on the source table:

- ALTER privilege
- CONTROL privilege
- DBADM
- ALTERIN privilege on the table schema

About this task

- Each ALTER TABLE statement issued with the PARTITION clause must be in a separate SQL statement.
- No other ALTER operations are permitted in an SQL statement containing an ALTER TABLE ... PARTITION operation. For example, you cannot attach a data partition and add a column to the table in a single SQL statement.
- Multiple ALTER statements can be executed, followed by a single SET INTEGRITY statement.

Procedure

- To alter a partitioned table from the command line, issue the ALTER TABLE statement.

Guidelines and restrictions on altering partitioned tables

This topic identifies the most common alter table actions and special considerations in the presence of attached and detached data partitions.

The STATUS column of the SYSCAT.DATAPARTITIONS catalog view contains the state information for the partitions of a table.

- If the STATUS is the empty string, the partition is visible and is in the normal state.
- If the STATUS is 'A', the partition is newly attached and the SET INTEGRITY statement must be issued to bring the attached partition into the normal state.
- If the STATUS is 'D', 'L', or 'I', the partition is being detached, but the detach operation has not completed.
  – For a partition in the 'D' state, the SET INTEGRITY statement must be issued on all detached dependent tables in order to transition the partition to the logically detached state.
  – For a partition in the 'L' state, the partition is a logically detached partition and the asynchronous partition detach task is completing the detach of the partition for Db2 Version 9.7 Fix Pack 1 and later releases.
  – For a partition in the 'I' state the asynchronous partition detach task has completed and asynchronous index cleanup is updating nonpartitioned indexes defined on the partition.

Adding or attaching a data partition - locking behavior

For information about locking behavior during an add partition or attach partition operation, see the "Scenarios: Rolling in and rolling out partitioned table data" topic.

Adding or altering a constraint

Adding a check or a foreign key constraint is supported with attached and detached data partitions. When a partitioned table has detached partitions in state 'D' or 'L', adding a primary or unique constraint will return an error if the system has to generate a new partitioned index to enforce the constraint. For a partition in the 'L' state, the operation returns SQL20285N (SQLSTATE 55057). For a partition in the 'D' state, the operation returns SQL20054 (SQLSTATE 55019).

Adding a column

When adding a column to a table with attached data partitions, the column is also added to the attached data partitions. When adding a column to a table with detached data partitions in the 'I' state, the column is not added to the detached data partitions because the detached data partitions are no longer physically associated to the table.

For a detached partition in the 'L' or 'D' state, the operation fails and an error is returned. For a partition in the 'L' state, the operation returns SQL20285N (SQLSTATE 55057). For a partition in the 'D' state, the operation returns SQL20296N (SQLSTATE 55057).

Altering a column

When altering a column in a table with attached data partitions, the column will also be altered on the attached data partitions. When altering a column in a table with detached data partitions, the column is not altered on the detached data partitions, because the detached data partitions are no longer physically associated to the table.

When dropping or renaming a column when a partition is detached in the 'L' or 'D' state the operation fails and an error is returned. For a partition in the 'L' state, the operation returns SQL20285N (SQLSTATE 55057). For a partition in the 'D' state, the operation returns SQL0270N (SQLSTATE 42997).

Adding a generated column

When adding a generated column to a partitioned table with attached or detached data partitions, it must respect the rules for adding any other types of columns.

Adding or modifying a nonpartitioned index

When creating, recreating, or reorganizing an index on a table with attached data partitions, the index does not include the data in the attached data partitions because the SET INTEGRITY statement maintains all indexes for all attached data partitions. When creating, recreating or reorganizing an index on a table with detached data partitions, the index does not include the data in the detached data partitions, unless the detached data partition has a detached dependent table or staging tables that need to be incrementally refreshed with respect to the data partition, the partition is in the 'D' state. In this case, the index includes the data for this detached data partition.

Adding or modifying a partitioned index

When creating a partitioned index in the presence of attached data partitions, an index partition for each attached data partition will be created. The index entries for index partitions on attached data partitions will not be visible until the SET INTEGRITY statement is run to bring the attached data partitions online. Note that because create index includes the attached data partitions, creation of a unique partitioned index may find rows in the attached data partition which are duplicate key values and thus fail the index creation. It is recommended that users do not attempt to create partitioned indexes in the presence of attached partitions to avoid this problem.

If the table has any detached dependent tables, creation of partitioned indexes is not supported on partitioned tables with detached dependent tables. Any attempt to create a partitioned index in this situation will result in SQLSTATE 55019. When creating a partitioned index on a table that has partitions in 'L' state, the operation returns SQL20285N (SQLSTATE 55057).

WITH EMPTY TABLE

You cannot empty a table with attached data partitions.

ADD MATERIALIZED QUERY AS
Altering a table with attached data partitions to an MQT is not allowed.

Altering additional table attributes that are stored in a data partition
The following table attributes are also stored in a data partition. Changes to these attributes are reflected on the attached data partitions, but not on the detached data partitions.

- DATA CAPTURE
- VALUE COMPRESSION
- APPEND
- COMPACT/LOGGED FOR LOB COLUMNS

Creating and accessing data partitions within the same transaction
If a table has a nonpartitioned index, you cannot access a new data partition in that table within the same transaction as the add or attach operation that created the partition, if the transaction does not have the table locked in exclusive mode (SQL0668N, reason code 11).

Special considerations for XML indexes when altering a table to ADD, ATTACH, or DETACH a partition

Similar to a nonpartitioned relational index, a nonpartitioned index over an XML column is an independent object that is shared among all data partitions of a partitioned table. XML region indexes and column path indexes are affected when you alter a table by adding, attaching, or detaching a partition. Indexes over XML column paths are always nonpartitioned, and indexes over XML data are generated as partitioned by default.

XML regions index

ADD PARTITION will create a new regions index partition for the new empty data partition being added. A new entry for the regions index partition will be added to the SYSINDEXPARTITIONS table. The table space for the partitioned index object on the new partition will be determined by the INDEX IN <table space> in the ADD PARTITION clause. If no INDEX IN <table space> is specified for the ADD PARTITION clause, the table space for the partitioned index object will be the same as the table space used by the corresponding data partition by default.

The system-generated XML regions index on a partitioned table is always partitioned. A partitioned index uses an index organization scheme in which index data is divided across multiple storage objects, called index partitions, according to the table partitioning scheme of the table. Each index partition only refers to table rows in the corresponding data partition.

For ATTACH, since the regions index on a partitioned table with XML column is always partitioned, the region index on the source table can be kept as the new regions index partition for the new table partition after completing the ATTACH operation. Data and index objects do not move, therefore the catalog table entries need to be updated. The catalog table entry for the regions index on the source table will be removed on ATTACH and one regions index partition will be added in the SYSINDEXPARTITIONS table. The pool ID and object ID will remain the same as they were on the source table. The index ID (IID) will be modified to match that of the regions index on the target.

After completing the DETACH operation, the regions index will be kept on the detached table. The index partition entry associated to the partition being detached will be removed from the SYSINDEXPARTITIONS table. One new regions index entry will be added in the SYSINDEXES catalog table for the detached table, which will have the same pool ID and object ID as the region index partition before the DETACH.

Index over XML data

Starting in Db2 Version 9.7 Fix Pack 1, you can create an index over XML data on a partitioned table as either partitioned or nonpartitioned. The default is a partitioned index.

Partitioned and nonpartitioned indexes over XML data are treated like any other relational indexes during ATTACH and DETACH operations.

Indexes on the source table will be dropped during the ATTACH operation. This applies to both the logical and physical XML indexes. Their entries in the system catalogs will be removed during the ATTACH operation.

Set integrity must be run after ATTACH, to maintain the nonpartitioned indexes over XML data on the target table.

For DETACH, nonpartitioned indexes over XML columns on the source table are not inherited by the target table.

XML column path indexes

Indexes over XML column paths are always nonpartitioned indexes. The XML column path indexes on the source and target tables are maintained during roll-in and rollout operations.

For ATTACH, the Db2 database manager will maintain the nonpartitioned XML column path indexes on the target table (this is unlike other nonpartitioned indexes, which are maintained during SET INTEGRITY after completing the ATTACH operation). Afterwards, the XML column path indexes on the source table will be dropped and their catalog entries will be removed because the column path indexes on the target table are nonpartitioned.

For rollout, recall that the XML column path indexes are nonpartitioned, and nonpartitioned indexes are not carried along to the standalone target table. However, XML column path indexes (one for each column) must exist on a table with XML columns before the table can be accessible to external user, therefore XML column path indexes must be created on the target table before it can be used. The time at which the column path indexes will be created depends on whether there are any detached dependent tables during the DETACH operation. If there are no detached dependent tables, then the paths indexes will be created during the DETACH operation, otherwise they will be created by SET INTEGRITY or MQT refresh to maintain the detach dependent objects.

After DETACH, the XML column path indexes created on the target table will reside in the same index object along with all other indexes on that table.

Attaching data partitions

Table partitioning allows for the efficient roll-in and roll-out of table data. The ALTER TABLE statement with the ATTACH PARTITION clause makes data roll-in easier.

Before you begin

If data integrity checking, including range validation and other constraints checking, can be done through application logic that is independent of the data server before an attach operation, newly attached data can be made available for use much sooner. You can optimize the data roll-in process by using the SET INTEGRITY...ALL IMMEDIATE UNCHECKED statement to skip range and constraints violation checking. In this case, the table is brought out of SET INTEGRITY pending state, and the new data is available for applications to use immediately, as long as all user indexes on the target table are partitioned indexes.

If there are nonpartitioned indexes (except XML column path indexes) on the table to maintain after an attach operation, the SET INTEGRITY...ALL IMMEDIATE UNCHECKED statement behaves as though it were a SET INTEGRITY...IMMEDIATE CHECKED statement. All integrity processing, nonpartitioned index maintenance, and table state transitions are performed as though a SET INTEGRITY...IMMEDIATE CHECKED statement was issued. This behavior ensures that a roll-in script that uses SET INTEGRITY...ALL IMMEDIATE UNCHECKED does not stop working if a nonpartitioned index is created for the target table some time after the roll-in script is put into service.

To alter a table to attach a data partition, the privileges held by the authorization ID of the statement must include at least one of the following authorities or privileges on the source table:

- SELECT privilege on the table and DROPIN privilege on the schema of the table
- CONTROL privilege on the table
- DATAACCESS authority

About this task

Attaching data partitions takes an existing table (source table) and attaches it to the target table as a new data partition. When attaching a data partition to a partitioned table by using the ALTER TABLE statement with the ATTACH PARTITION clause, the target partitioned table remains online, and dynamic queries against the table, running under the RS, CS, or UR isolation level, continue to run.

Restrictions and usage guidelines

The following conditions must be met before you can attach a data partition:

- The target table to which you want to attach the new data partition must be an existing partitioned table.
- The source table must be an existing nonpartitioned table or a partitioned table with a single data partition and no attached data partitions or detached data partitions. To attach multiple data partitions, you must issue multiple ATTACH statements.
- The source table cannot be a typed table.
- The source table cannot be a range-clustered table.
- The source and target table definitions must match.
- The number, type, and ordering of source and target columns must match.
- Source and target table columns must match in terms of whether they contain default values.
- Source and target table columns must match in terms of whether they allow null values.
- Source and target table compression specifications, including the VALUE COMPRESSION and COMPRESS SYSTEM DEFAULT clauses, must match.
- Source and target table specifications for the DATA CAPTURE, ACTIVATE NOT LOGGED INITIALLY, and APPEND options must match.
- Attaching a data partition is allowed even when a target column is a generated column and the corresponding source column is not a generated column. The following statement generates the values for the generated column of the attached rows:

  ```
  SET INTEGRITY FOR table-name
    ALLOW WRITE ACCESS
    IMMEDIATE CHECKED FORCE GENERATED
  ```

  The source table column that matches a generated column must match in type and nullability; however, a default value is not required. The recommended approach is to guarantee that the source table for the attach operation has the correct generated value in the generated column. If you follow the recommended approach, you are not required to use the FORCE GENERATED option, and the following statements can be used.

  ```
  SET INTEGRITY FOR table-name
    GENERATED COLUMN
    IMMEDIATE UNCHECKED
  ```

  This statement indicates that checking of the generated column is to be bypassed.

  ```
  SET INTEGRITY FOR table-name
    ALLOW WRITE ACCESS
    IMMEDIATE CHECKED
    FOR EXCEPTION IN table-name USE table-name
  ```

  This statement performs integrity checking of the attached data partition but does not check the generated column.
- Attaching a data partition is allowed even when the target column is an identity column and the source column is not an identity column. The statement SET INTEGRITY IMMEDIATE CHECKED does not generate identity values for the attached rows. The statement SET INTEGRITY FOR T GENERATE IDENTITY ALLOW WRITE ACCESS IMMEDIATE CHECKED fills in the identity values for the attached rows. The column that matches an identity column must match in type and nullability. There is no requirement on the default values of this column. The recommended approach is for you to fill in the

correct identity values at the staging table. Then after the ATTACH, there is no requirement to use the GENERATE IDENTITY option because the identity values are already guaranteed in the source table.

- For tables whose data is distributed across database partitions, the source table must also be distributed, in the same database partition group using the same distribution key and the same distribution map.
- The source table must be dropable (that is, it cannot have RESTRICT DROP set).
- If a data partition name is specified, it must not exist in the target table.
- If the target table is a multidimensional clustering (MDC) table, the source table must also be an MDC table.
- When using a nonpartitioned table, the data table space for the source table must match the data table spaces for the target table in type (that is, DMS or SMS), page size, extent size, and database partition group. A warning is returned if the prefetch size does not match. The index table space for the source table must match the index table spaces used by the partitioned indexes for the target table in type, database partition group, page size, and extent size. The large table space for the source table must match the large table spaces for the target table in type, database partition group, and page size. When using a partitioned table, the data table space for the source table must match the data table spaces for the target table in type, page size, extent size, and database partition group.
- When you issue the ALTER TABLE ATTACH statement to a partitioned table with any structured, XML, or LOB columns, the INLINE LENGTH of any structured, XML, or LOB columns on the source table must match with the INLINE LENGTH of the corresponding structured, XML, or LOB columns on the target table.
- When you use the REQUIRE MATCHING INDEXES clause with the ATTACH PARTITION clause, if there are any partitioned indexes on the target table that do not have a match on the source table, SQL20307N is returned.
- Attaching a source table that does not have a matching index for each partitioned unique index on the target table causes the attach operation to fail with error SQL20307N, reason code 17.
- When a table has a deferred index cleanup operation in progress as the result of an MDC rollout, since MDC rollout using the deferred index cleanup mechanism is not supported for partitioned indexes, the attach operation is not allowed if there are any RID indexes on the source table that are kept during the attach operation, not rebuilt, and are pending asynchronous index cleanup of the rolled-out blocks.
- Attaching a source table with an XML data format that is different from the XML data format of the target table is not supported.
- If a table contains XML columns that use the Version 9.5 or earlier XML record format, attaching the table to a partitioned table that contains XML columns that use the Version 9.7 or later record format is not supported.

  Before attaching the table, you must update the XML record format of the table to match the record format of the target partitioned table. Either of the following two methods updates the XML record format of a table:

  – Perform an online table move on the table by using the ADMIN_MOVE_TABLE procedure.
  – Perform the following steps:

    1. Use the EXPORT command to create a copy of the table data.
    2. Use the TRUNCATE statement to delete all the rows from the table and release the storage allocated to the table.
    3. Use the LOAD command to add the data into the table.

  After the XML record format of the table is updated, attach the table to the target partitioned table.
- If a table has a nonpartitioned index, you cannot access a new data partition in that table within the same transaction as the add or attach operation that created the partition, if the transaction does not have the table locked in exclusive mode (SQL0668N, reason code 11).

Before running the attach operation, create indexes on the source table that match each of the partitioned indexes in the target table. Matching the partitioned indexes makes the roll-in operation more

efficient and less active log space is needed. If the indexes on the source table are not properly prepared, the database manager is required to maintain them for you. To ensure that your roll-in does not incur any additional cost to maintain the partitioned indexes, you can specify REQUIRE MATCHING INDEXES on the attach partition operation. Specifying REQUIRE MATCHING INDEXES ensures that the attach operation fails if a source table does not have indexes to match the partitioned indexes on the target. You can then take the corrective action and reissue the attach operation.

In addition, drop any extra indexes on the source table before running the attach operation. Extra indexes are those indexes on the source table that either do not have a match on the target table, or that match nonpartitioned indexes on the target table. Dropping extra indexes before running the attach operation makes it run faster.

For example, assume that a partitioned table called ORDERS has 12 data partitions (one for each month of the year). At the end of each month, a separate table called NEWORDERS is attached to the partitioned ORDERS table.

1. Create partitioned indexes on the ORDERS table.

```
CREATE INDEX idx_delivery_date ON orders(delivery) PARTITIONED
CREATE INDEX idx_order_price ON orders(price) PARTITIONED
```

2. Prepare for the attach operation by creating the corresponding indexes on the NEWORDERS table.

```
CREATE INDEX idx_delivery_date_for_attach ON neworders(delivery)
CREATE INDEX idx_order_price_for_attach ON neworders(price)
```

3. There are two steps to the attach operation:
   a. ATTACH. The indexes on the NEWORDERS table that match the partitioned indexes on the ORDERS table are kept.

```
ALTER TABLE orders ATTACH PARTITION part_jan2009
   STARTING FROM ('01/01/2009')
   ENDING AT ('01/31/2009') FROM TABLE neworders
```

   The ORDERS table is automatically placed into the Set Integrity Pending state. Both the idx_delivery_date_for_attach index and the idx_order_price_for_attach index become part of the ORDERS table after the completion of the attach operation. No data movement occurs during this operation.

   b. SET INTEGRITY. A range check is done on the newly attached partition. Any constraints that exist are enforced. Upon completion, the newly attached data becomes visible within the database.

```
SET INTEGRITY FOR orders IMMEDIATE CHECKED
```

When nonpartitioned indexes exist on the target table, the SET INTEGRITY statement has to maintain the index along with other tasks, such as range validation and constraints checking on the data from the newly attached partition. Nonpartitioned index maintenance requires a large amount of active log space that is proportional to the data volumes in the newly attached partition, the key size of each nonpartitioned index, and the number of nonpartitioned indexes.

Each partitioned index on the new data partition is given an entry in the SYSINDEXPARTITIONS catalog table using the table space identifier and object identifier from the source table. The identifier information is taken from either the SYSINDEXES table (if the table is nonpartitioned) or the SYSINDEXPARTITIONS table (if the table is partitioned). The index identifier is taken from the partitioned index of the matching target table.

When the source table is partitioned, those partitioned indexes on the source table that match the partitioned indexes on the target table are kept as part of the attach operation. Index partition entries in the SYSINDEXPARTITIONS table are updated to show that they are index partitions on the new target table with new index identifiers.

When attaching data partitions, some statistics for indexes as well as data are carried over from the source table to the target table for the new partition. Specifically, all fields in the SYSDATAPARTITIONS and SYSINDEXPARTITIONS tables for the new partition on the target are populated from the source.

When the source table is nonpartitioned, these statistics come from the SYSTABLES and SYSINDEXES tables. When the source table is a single-partition partitioned table, these statistics come from the SYSDATAPARTITIONS and SYSINDEXPARTITIONS tables of the single source partition.

Note: Execute a runstats operation after the completion of an attach operation, because the statistics that are carried over will not affect the aggregated statistics in the SYSINDEXES and SYSTABLES tables.

Nonpartitioned index maintenance during SET INTEGRITY...ALL IMMEDIATE UNCHECKED. When SET INTEGRITY...ALL IMMEDIATE UNCHECKED is issued on a partitioned table to skip range checking for a newly attached partition, if there are any nonpartitioned indexes (except the XML column path index) on the table, SET INTEGRITY...ALL IMMEDIATE UNCHECKED performs as follows:

- If the SET INTEGRITY...ALL IMMEDIATE UNCHECKED statement references one target table, the behavior is as though a SET INTEGRITY...ALLOW WRITE ACCESS...IMMEDIATE CHECKED statement was issued instead. The SET INTEGRITY...ALL IMMEDIATE UNCHECKED statement maintains all nonpartitioned indexes (except XML column path indexes), performs all other integrity processing, updates the constraints checking flag values in the CONST_CHECKED column in the SYSCAT.TABLES catalog view, and returns errors and stops immediately when constraints violations are detected.
- If the SET INTEGRITY...ALL IMMEDIATE UNCHECKED statement references more than one target table, an error is returned (SQL20209N with reason code 13).

Rebuild of invalid partitioned indexes during SET INTEGRITY. The SET INTEGRITY statement can detect whether the partitioned index object for a newly attached partition is invalid and performs a partitioned index rebuild if necessary.

Guidelines for attaching data partitions to partitioned tables

This topic provides guidelines for correcting various types of mismatches that can occur when attempting to attach a data partition to a partitioned table when issuing the ALTER TABLE ...ATTACH PARTITION statement. You can achieve agreement between tables by modifying the source table to match the characteristics of the target table, or by modifying the target table to match the characteristics of the source table.

The source table is the existing table you want to attach to a target table. The target table is the table to which you want to attach the new data partition.

One suggested approach to performing a successful attach is to use the exact CREATE TABLE statement for the source table as you did for the target table, but without the PARTITION BY clause. In cases where it is difficult to modify the characteristics of either the source or target tables for compatibility, you can create a new source table that is compatible with the target table. For details on creating a new source, see "Creating tables like existing tables".

To help you prevent a mismatch from occurring, see the restrictions and usage guidelines section of "Attaching data partitions". The section outlines conditions that must be met before you can successfully attach a data partition. Failure to meet the listed conditions returns error SQL20408N or SQL20307N.

The following sections describe the various types of mismatches that can occur and provides the suggested steps to achieve agreement between tables:

The (value) compression clause (the COMPRESSION column of SYSCAT.TABLES) does not match. (SQL20307N reason code 2)

To achieve value compression agreement, use one of the following statements:

```
ALTER TABLE... ACTIVATE VALUE COMPRESSION
or
ALTER TABLE... DEACTIVATE VALUE COMPRESSION
```

To achieve row compression agreement use one of the following statements:

```
ALTER TABLE... COMPRESS YES
or
ALTER TABLE... COMPRESS NO
```

The APPEND mode of the tables does not match. (SQL20307N reason code 3)

To achieve append mode agreement use one of the following statements:

```
ALTER TABLE ... APPEND ON
or
ALTER TABLE ... APPEND OFF
```

The code pages of the source and target table do not match. (SQL20307N reason code 4)

Create a new source

The source table is a partitioned table with more than one data partition or with attached or detached data partitions. (SQL20307N reason code 5)

Detach data partitions from the source table until there is a single visible data partition using the statement:

```
ALTER TABLE ... DETACH PARTITION
```

Detached partitions remain detached until each of the following steps has been completed:

1. Execute any necessary SET INTEGRITY statements to incrementally refresh detached dependents.
2. In Version 9.7.1 and later, wait for the detach to complete asynchronously. To expedite this process, ensure that all access to the table that started prior to the detach operation either completes or is terminated.
3. If the source table has nonpartitioned indexes, wait for the asynchronous index cleanup to complete. To expedite this process, one option might be to drop the nonpartitioned indexes on the source table.

If you want to perform an attach operation immediately, one option might be to create a new source table.

The source table is a system table, a view, a typed table, a table ORGANIZED BY KEY SEQUENCE, a created temporary table, or a declared temporary table. (SQL20307N reason code 6)

Create a new source.

The target and source table are the same. (SQL20307N reason code 7)

You cannot attach a table to itself. Determine the correct table to use as the source or target table.

The NOT LOGGED INITIALLY clause was specified for either the source table or the target table, but not for both. (SQL20307N reason code 8)

Either make the table that is not logged initially be logged by issuing the COMMIT statement, or make the table that is logged be not logged initially by entering the statement:

```
ALTER TABLE ... ACTIVATE NOT LOGGED INITIALLY
```

The DATA CAPTURE CHANGES clause was specified for either the source table or the target table, but not both. (SQL20307N reason code 9)

To enable data capture changes on the table that does not have data capture changes turned on, run the following statement:

```
ALTER TABLE ... DATA CAPTURE CHANGES
```

To disable data capture changes on the table that does have data capture changes turned on, run the statement:

```
ALTER TABLE ... DATA CAPTURE NONE
```

The distribution clauses of the tables do not match. The distribution key must be the same for the source table and the target table. (SQL20307N reason code 10)

It is recommended that you create a new source table. You cannot change the distribution key of a table spanning multiple database partitions. To change a distribution key on tables in single-partition database, run the following statements:

```
ALTER TABLE ... DROP DISTRIBUTION;
    ALTER TABLE ... ADD DISTRIBUTION(key-specification)
```

An error is returned when there are missing indexes during an attach operation (SQL20307N reason code 18)

The attach operation implicitly builds missing indexes on the source table corresponding to the partitioned indexes on the target table. The implicit creation of the missing indexes does take time to complete. You have an option to create and error condition if the attach operation encounters any missing indexes. The option is called ERROR ON MISSING INDEXES and is one of the attach operation options. The error returned when this happens is SQL20307N, SQLSTATE 428GE, reason code 18. Information on the nonmatching indexes is placed in the administration log.

The attach operation drops indexes on the source table that do not match the partitioned indexes on the target table. The identification and dropping of these nonmatching indexes takes time to complete. You should drop these indexes before attempting the attach operation.

An error is returned when the nonmatching indexes on the target table are unique indexes, or the XML indexes are defined with the REJECT INVALID VALUES clause, during an attach operation (SQL20307N reason code 17)

When there are partitioned indexes on the target table with no matching indexes on the source table and the ERROR ON MISSING INDEXES is not used, then you could expect the following results:

1. If the nonmatching indexes on the target table are unique indexes, or the XML indexes are defined with the REJECT INVALID VALUES clause, then the attach operation will fail and return the error message SQL20307N, SQLSTATE 428GE, reason code 17.
2. If the nonmatching indexes on the target table do not meet the conditions in the previous point, the index object on the source table is marked invalid during the attach operation. The attach operation completes successfully, but the index object on the new data partition is marked invalid. The SET INTEGRITY operation is used to rebuild the index objects on the newly attached partition. Typically this is the next operation you would perform following the attaching of a data partition. The recreation of the indexes takes time.

The administration log will have details about any mismatches between the indexes on the source and target tables.

Only one of the tables has an ORGANIZE BY DIMENSIONS clause specified or the organizing dimensions are different. (SQL20307N reason code 11)

Create a new source.

The data type of the columns (TYPENAME) does not match. (SQL20408N reason code 1)

To correct a mismatch in data type, issue the statement:

```
ALTER TABLE ... ALTER COLUMN ... SET DATA TYPE...
```

The nullability of the columns (NULLS) does not match. (SQL20408N reason code 2)

To alter the nullability of the column that does not match for one of the tables issue one of the following statements:

```
ALTER TABLE... ALTER COLUMN... DROP NOT NULL
or
ALTER TABLE... ALTER COLUMN... SET NOT NULL
```

The implicit default value (SYSCAT.COLUMNS IMPLICITVALUE) of the columns are incompatible. (SQL20408N reason code 3)

Create a new source table. Implicit defaults must match exactly if both the target table column and source table column have implicit defaults (if IMPLICITVALUE is not NULL).

If IMPLICITVALUE is not NULL for a column in the target table and IMPLICITVALUE is not NULL for the corresponding column in the source table, each column was added after the original CREATE TABLE statement for the table. In this case, the value stored in IMPLICITVALUE must match for this column.

There is a situation, where through migration from a pre-V9.1 table or through attach of a data partition from a pre-V9.1 table, that IMPLICITVALUE is not NULL because the system did not know whether or not the column was added after the original CREATE TABLE statement. If the database is not certain whether the column is added or not, it is treated as added. An added column is a column created as the result of an ALTER TABLE ...ADD COLUMN statement. In this case, the statement is not allowed because the value of the column could become corrupted if the attach were allowed to proceed. You must copy the data from the source table to a new table (with IMPLICITVALUE for this column NULL) and use the new table as the source table for the attach operation.

The code page (COMPOSITE_CODEPAGE) of the columns does not match. (SQL20408N reason code 4)

Create a new source table.

The system compression default clause (COMPRESS) does not match. (SQL20408N reason code 5)

To alter the system compression of the column issue one of the following statements to correct the mismatch:

```
ALTER TABLE ... ALTER COLUMN ...  COMPRESS SYSTEM DEFAULT
or
ALTER TABLE ... ALTER COLUMN ... COMPRESS OFF
```

Conditions for matching a source table index with a target table partitioned index

If you want to reuse the indexes on the source table as index partitions on a target table when attaching data partitions, all index key columns or expressions for the indexes on the source table must match the index key columns or expressions for the partitioned indexes on the target table.

If the source table has an expression-based index, the system-generated statistical view and package that are associated with the index are dropped as part of the process to attach the partition. If the target table has a partitioned expression-based index, the expression is used when determining whether the source table has a matching index. If all other properties of the two indexes are the same, the index on the source table is considered to match the partitioned index on the target table. That is, the index on the source table can be used as an index on the target table.

The following table applies only when the target index is partitioned. The target index property is assumed by the source index in all cases where they are considered to be a match.

Table 14. Determining whether the source index matches the target index when the target index property is different from the source index property

Rule number	Target index property	Source index property	Does the source index match?
1.	Non-unique	Unique	Yes, if the index is not an XML index.
2.	Unique	Non-unique	No.
3.	Column X is descending	Column X is ascending	No.
4.	Column X is ascending	Column X is descending	No.
5.	Column X is random	Column X is ascending	No.

Table 14. Determining whether the source index matches the target index when the target index property is different from the source index property (continued)

Rule number	Target index property	Source index property	Does the source index match?
6.	Column X is ascending	Column X is random	No.
7.	Column X is random	Column X is descending	No.
8.	Column X is descending	Column X is random	No.
9.	Partitioned	Nonpartitioned	No. It is assumed that the source table is partitioned.
10.	pctfree n1	pctfree n2	Yes
11.	level2pctfree n1	level2pctfree n2	Yes.
12.	minpctused n1	minpctused n2	Yes.
13.	Disallow reverse scans	Allow reverse scans	Yes, the physical index structure is the same irrespective of whether reverse scans are allowed.
14.	Allow reverse scans	Disallow reverse scans	Yes, the same reason as 9.
15.	pagesplit [L|H|S]	pagesplit [L|H|S]	Yes.
16.	Sampled statistics	Detailed statistics	Yes.
17.	Detailed statistics	Sampled statistics	Yes.
18.	Not clustered	Clustered	Yes.
19.	Clustered	Not clustered	Yes. The index becomes a clustering index, but the data is not clustered according to this index until the data is reorganized. You can use a partition-level reorganization after attaching the data partition to cluster the data according to this index partition.
20.	Ignore invalid values	Reject invalid values	Yes.
21.	Reject invalid values	Ignore invalid values	No. The target index property of rejecting invalid values must be respected, and the source table might have rows that violate this index constraint.
22.	Index compression enabled	Index compression not enabled	Yes. Compression of the underlying index data does not occur until the index is rebuilt.
23.	Index compression not enabled	Index compression enabled	Yes. Decompression of the index data does not occur until the index is rebuilt.

Note: With rule number 9, an ALTER TABLE ... ATTACH PARTITION statement fails and returns error message SQL20307N, SQLSTATE 428GE if both of the following conditions are true:

- You attempt to attach a multidimensional clustering (MDC) table (with nonpartitioned block indexes) that you created by using Db2 Version 9.7 or earlier to a new MDC partitioned table (with partitioned block indexes) that you created by using Db2 Version 9.7 Fix Pack 1 or later.

- You specify the ERROR ON MISSING INDEXES clause.

Removing the ERROR ON MISSING INDEXES clause allows the attachment to be completed because the database manager maintains the indexes during the attach operation. If you receive error message SQL20307N, SQLSTATE 428GE, consider removing the ERROR ON MISSING INDEXES clause. An alternative is to use the online table move procedure to convert an MDC partitioned table that has nonpartitioned block indexes to a table that has partitioned block indexes.

Detaching data partitions

Table partitioning allows for the efficient roll-in and roll-out of table data. This efficiency is achieved by using the ATTACH PARTITION and DETACH PARTITION clauses of the ALTER TABLE statement.

Before you begin

To detach a data partition from a partitioned table you must have the following authorities or privileges:

- The user performing the DETACH PARTITION operation must have the authority necessary to ALTER, to SELECT from and to DELETE from the source table.
- The user must also have the authority necessary to create the target table. Therefore, to alter a table to detach a data partition, the privilege held by the authorization ID of the statement must include at least one of the following authorities or privileges on the target table:
  - DBADM authority
  - CREATETAB authority on the database and USE privilege on the table spaces used by the table as well as one of:
    - IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the table does not exist
    - CREATEIN privilege on the schema, if the schema name of the table refers to an existing schema.

Note: When detaching a data partition, the authorization ID of the statement is going to effectively perform a CREATE TABLE statement and therefore must have the necessary privileges to perform that operation. The authorization ID of the ALTER TABLE statement becomes the definer of the new table with CONTROL authority, as if the user had issued the CREATE TABLE statement. No privileges from the table being altered are transferred to the new table. Only the authorization ID of the ALTER TABLE statement and users with DBADM or DATAACCESS authority have access to the data immediately after the ALTER TABLE...DETACH PARTITION statement.

About this task

Rolling-out partitioned table data allows you to easily separate ranges of data from a partitioned table. Once a data partition is detached into a separate table, the table can be handled in several ways. You can drop the separate table (whereby, the data from the data partition is destroyed); archive it or otherwise use it as a separate table; attach it to another partitioned table such as a history table; or you can manipulate, cleanse, transform, and reattach to the original or some other partitioned table.

With Db2 Version 9.7 Fix Pack 1 and later releases, when detaching a data partition from a partitioned table by using the ALTER TABLE statement with the DETACH PARTITION clause, the source partitioned table remains online. Queries running against the table continue to run. The data partition being detached is converted into a stand-alone table in the following two-phase process:

1. The ALTER TABLE...DETACH PARTITION operation logically detaches the data partition from the partitioned table.
2. An asynchronous partition detach task converts the logically detached partition into a stand-alone table.

If there are any dependent tables that need to be incrementally maintained with respect to the detached data partition (these dependent tables are referred to as detached dependent tables), the asynchronous partition detach task starts only after the SET INTEGRITY statement is run on all detached dependent tables.

In absence of detached dependent tables, the asynchronous partition detach task starts after the transaction issuing the ALTER TABLE...DETACH PARTITION statement commits.

Restrictions

If the source table is an MDC table created by Db2 Version 9.7 or earlier releases, block indexes are not partitioned. Access to the newly detached table is not allowed in the same unit of work as the ALTER TABLE...DETACH PARTITION operation. MDC tables do not support partitioned block indexes. In that case, block indexes are created upon first access to the table after the ALTER TABLE...DETACH PARTITION operation is committed. If the source table had any other partitioned indexes before detach time then the index object for the target table is marked invalid to allow for creation of the block indexes. As a result access time is increased while the block indexes are created and any partitioned indexes are recreated.

When the source table is an MDC created by Db2 V9.7 Fix Pack 1 or later releases, the block indexes are partitioned, and partitioned indexes become indexes on the target table of detach without the need to be recreated.

You must meet the following conditions before you can perform a DETACH PARTITION operation:

- The table to be detached from (source table) must exist and be a partitioned table.
- The data partition to be detached must exist in the source table.
- The source table must have more than one data partition. A partitioned table must have at least one data partition. Only visible and attached data partitions pertain in this context. An attached data partition is a data partition that is attached but not yet validated by the SET INTEGRITY statement.
- The name of the table to be created by the DETACH PARTITION operation (target table) must not exist.
- DETACH PARTITION is not allowed on a table that is the parent of an enforced referential integrity (RI) relationship. If you have tables with an enforced RI relationship and want to detach a data partition from the parent table, a workaround is available. In the following example, all statements are run within the same unit of work (UOW) to lock out concurrent updates:

```
// Change the RI constraint to informational:
ALTER TABLE child ALTER FOREIGN KEY fk NOT ENFORCED;

ALTER TABLE parent DETACH PARTITION p0 INTO TABLE pdet;

SET INTEGRITY FOR child OFF;

// Change the RI constraint back to enforced:
ALTER TABLE child ALTER FOREIGN KEY fk ENFORCED;

SET INTEGRITY FOR child ALL IMMEDIATE UNCHECKED;
// Assuming that the CHILD table does not have any dependencies on partition P0,
//   and that no updates on the CHILD table are permitted
//   until this UOW is complete,
//   no RI violation is possible during this UOW.

COMMIT WORK;
```

- If there are any dependent tables that need to be incrementally maintained with respect to the detached data partition (these dependent tables are referred to as detached dependent tables), the SET INTEGRITY statement is required to be run on the detached dependent tables to incrementally maintain the tables. With Db2 V9.7 Fix Pack 1 or later releases, after the SET INTEGRITY statement is run on all detached dependent tables, the asynchronous partition detach task makes the data partition into a stand-alone target table. Until the asynchronous partition detach operation completes, the target table is unavailable.

Procedure

1. To alter a partitioned table and to detach a data partition from the table, issue the ALTER TABLE statement with the DETACH PARTITION clause.
2. Optional: If you wish to have the same constraints on the newly detached stand-alone table, run the ALTER TABLE... ADD CONSTRAINT on the target table after completing the detach operation.

If the index was partitioned on the source table, any indexes necessary to satisfy the constraint already exist on the target table.

Results

The detached partition is renamed with a system-generated name (using the form SQL*yymmddhhmmssxxx*) so that a subsequent attach can reuse the detached partition name immediately.

Each of the index partitions defined on the source table for the data partition being detached becomes an index on the target table. The index object is not physically moved during the detach partition operation. However, the metadata for the index partitions of the table partition being detached are removed from the catalog table SYSINDEXPARTITIONS. New index entries are added in SYSINDEXES for the new table as a result of the detach partition operation. The original index identifier (IID) is kept and stays unique just as it was on the source table.

The index names for the surviving indexes on the target table are system-generated (using the form SQL*yymmddhhmmssxxx*). The schema for these indexes is the same as the schema of the target table except for any path indexes, regions indexes, and MDC or ITC block indexes, which are in the SYSIBM schema. Other system-generated indexes like those to enforce unique and primary key constraints will have a schema of the target table because the indexes are carried over to the detached table but the constraints are not. You can use the RENAME statement to rename the indexes that are not in the SYSIBM schema.

The table level INDEX IN option specified when creating the source table is not inherited by the target table. Rather, the partition level INDEX IN (if specified) or the default index table space for the detach partition continues to be the index table space for the target table.

When detaching data partitions, some statistics are carried over from the partition being detached into the target table. Specifically, statistics from SYSINDEXPARTITIONS for partitioned indexes will be carried over to the entries SYSINDEXES for the newly detached table. Statistics from SYSDATAPARTITIONS will be copied over to SYSTABLES for the newly detached table.

What to do next

Run **RUNSTATS** after the completion of the DETACH PARTITION operation on both the new detached table and the source table, because many of the statistics will not be carried over following the completion of the detach partition operation.

Attributes of detached data partitions

When you detach a data partition from a partitioned table by using the DETACH PARTITION clause of the ALTER TABLE statement, the detached data partition becomes a stand-alone, nonpartitioned table.

Many attributes of the target table are inherited from the source table. Any attributes that are not inherited from the source table are set by using default values on the CREATE TABLE statement. If there is a partitioned index on the source table, that index is carried over to the target table. If the source table has a partitioned index with expression-based key parts, the system-generated statistical view and package are created and associated with the index in the target table.

The target table inherits all the partitioned indexes that are defined on the source table. These indexes include both system-generated indexes and user-defined indexes. The index object is not physically moved during the detach operation. The index partition metadata of the detached data partition is removed from the SYSINDEXPARTITIONS catalog table. Entries for the new table are added to the SYSINDEXES catalog table. The index identifier (IID) for a particular partitioned index from the source table is used as the IID for the index on the target table. The IID remains unique with respect to the table.

The index names for the surviving indexes on the new table are system generated in the form SQL*yymmddhhmmssxxx*. Path indexes, region indexes, and MDC or ITC indexes are made part of the SYSIBM schema. All other indexes are made part of the schema of the new table. System-generated indexes such as those to enforce unique and primary key constraints are made part of the schema of the

new table because the indexes are carried over to the new table. You can use the RENAME statement to rename the indexes that are not in the SYSIBM schema.

Constraints on the source table are not inherited by the target table. After the detach operation, you can use the ALTER TABLE ... ADD CONSTRAINT statement on the new table to enforce the same constraints that are on the source table.

The table space location that is specified by the table-level INDEX IN clause on the source table is not inherited by the target table. Rather, the table space location that is specified by the partition-level INDEX IN clause or the default index table space for the new table is used as the index table space location for the new table.

Attributes that are inherited by the target table

Attributes that are inherited by the target table include:

- The following column definitions:
  - Column name
  - Data type (includes length and precision for types that have length and precision, such as CHAR and DECIMAL)
  - Nullability
  - Column default values
  - INLINE LENGTH
  - Code page (CODEPAGE column of the SYSCAT.COLUMNS catalog view)
  - Logging for LOBs (LOGGED column of the SYSCAT.COLUMNS catalog view)
  - Compaction for LOBs (COMPACT column of the SYSCAT.COLUMNS catalog view)
  - Compression (COMPRESS column of the SYSCAT.COLUMNS catalog view)
  - Type of hidden column (HIDDEN column of the SYSCAT.COLUMNS catalog view)
  - Column order
- If the source table is a multidimensional clustering (MDC) or insert time clustering (ITC) table, the target table is also an MDC or ITC table that is defined with the same dimension columns.
- Block index definitions. The indexes are rebuilt on first access to the target table after the detach operation is committed.
- The table space ID and table object ID are inherited from the data partition, not from the source table, because no table data is moved during a detach operation. The TBSPACEID column value in the SYSCAT.DATAPARTITIONS catalog view becomes the TBSPACEID column value in the SYSCAT.TABLES catalog view. The PARTITIONOBJECTID column value in the SYSCAT.DATAPARTITIONS catalog view becomes the TABLEID column value in the SYSCAT.TABLES catalog view.
- The LONG_TBSPACEID column value in the SYSCAT.DATAPARTITIONS catalog view is translated into a table space name and becomes the LONG_TBSPACE column value in the SYSCAT.TABLES catalog view.
- The INDEX_TBSPACEID column value in the SYSCAT.DATAPARTITIONS catalog view (the partition-level index table space) is translated into a table space name and becomes the INDEX_TBSPACE value in the SYSCAT.TABLES catalog view. The target table does not inherit the index table space that is specified by the table-level INDEX IN *table space* clause in the CREATE TABLE statement.
- Table space location
- ID of the distribution map for a multipartition database (PMAP_ID column of the SYSCAT.TABLES catalog view)
- Percent free (PCTFREE column of the SYSCAT.TABLES catalog view)
- Append mode (APPEND_MODE column of the SYSCAT.TABLES catalog view)
- Preferred lock granularity (LOCKSIZE column of the SYSCAT.TABLES catalog view)
- Data capture (DATA_CAPTURE column of the SYSCAT.TABLES catalog view)
- VOLATILE (VOLATILE column of the SYSCAT.TABLES catalog view)

- DROPRULE (DROPRULE column of the SYSCAT.TABLES catalog view)
- Compression (COMPRESSION column of the SYSCAT.TABLES catalog view)
- Maximum free space search (MAXFREESPACESEARCH column of the SYSCAT.TABLES catalog view)

Partitioned hierarchical or temporary tables, range-clustered tables, and partitioned views are not supported.

Attributes that are not inherited from the source table

Attributes that are not inherited from the source table include:

- Target table type. The target table is always a regular table.
- Privileges and authorities.
- Schema.
- Generated columns, identity columns, check constraints, and referential constraints. If a source column is a generated column or an identity column, the corresponding target column has no explicit default value, meaning that it has a default value of NULL.
- Table-level index table space (INDEX_TBSPACE column of the SYSCAT.TABLES catalog view). Indexes for the table resulting from the detach operation are in the same table space as the table.
- Triggers.
- Primary key constraints and unique key constraints.
- Statistics for nonpartitioned indexes.
- All other attributes that are not mentioned in the list of attributes that are explicitly inherited from the source table.

Data partition detach phases

With Db2 Version 9.7 Fix Pack 1 and later releases, detaching a data partition from a data partitioned table consists of two phases. The first phase logically detaches the partition from the table, the second phase converts the data partition into a stand-alone table.

The detach process is initiated when an ALTER TABLE...DETACH PARTITION statement is issued:

1. The ALTER TABLE...DETACH PARTITION operation logically detaches the data partition from the partitioned table.
2. An asynchronous partition detach task converts the logically detached partition into the stand-alone table.

If there are any dependent tables that need to be incrementally maintained with respect to the detached data partition (these dependent tables are referred to as detached dependent tables), the asynchronous partition detach task starts only after the SET INTEGRITY statement is run on all detached dependent tables.

In absence of detached dependent tables, the asynchronous partition detach task starts after the transaction issuing the ALTER TABLE...DETACH PARTITION statement commits.

DETACH operation

The ALTER TABLE...DETACH PARTITION operation performs in the following manner:

- The DETACH operation does not wait for dynamic uncommitted read (UR) isolation level queries before it proceeds, nor does it interrupt any currently running dynamic UR queries. This behavior occurs even when the UR query is accessing the partition being detached.
- If dynamic non-UR queries (read or write queries) did not lock the partition to be detached, the DETACH operation can complete while dynamic non-UR queries are running against the table.
- If dynamic non-UR queries locked the partition to be detached, the DETACH operation waits for the lock to be released.

- Hard invalidation must occur on all static packages that are dependent on the table before the DETACH operation can proceed.
- The following restrictions that apply to data definition language (DDL) statements also apply to a DETACH operation because DETACH requires catalogs to be updated:
  - New queries cannot be compiled against the table.
  - A bind or rebind cannot be performed on queries that run against the table.

  To minimize the impact of these restrictions, issue a COMMIT immediately after a DETACH operation.

During the DETACH operation, the data partition name is changed to a system-generated name of the form SQL*yymmddhhmmssxxx*, and in SYSCAT.DATAPARTITIONS, the status of the partition is set to 'L' if there are no detached dependent tables, or 'D' if there are detached dependent tables.

During the DETACH operation, an entry is created in SYSCAT.TABLES for the target table. If there are detached dependent tables, the table TYPE is set to 'L'. After SET INTEGRITY is run on all detached dependent tables, the TYPE is set to 'T', however, the target table continues to be unavailable. The asynchronous partition detach task completes the detach and makes the target table available.

Soft invalidation of dynamic SQL during the DETACH operation allows dynamic SQL queries that started before the ALTER TABLE...DETACH PARTITION statement to continue running concurrently with the DETACH operation. The ALTER TABLE...DETACH PARTITION statement acquires an IX lock on the partitioned table and an X lock on the data partition being detached.

Asynchronous partition detach task

After the DETACH operation commits and any detached dependent tables are refreshed, the asynchronous partition detach task converts the logically detached partition into the stand-alone table.

The asynchronous partition detach task waits for the completion of all access on the partitioned table that started before phase 1 of the detach operation. If the partitioned table has nonpartitioned indexes, the asynchronous partition detach task creates the asynchronous index cleanup task for deferred indexed cleanup. After the access completes, the asynchronous partition detach task completes phase 2 of the detached operation, by converting the logically detached partition into a stand-alone table.

The **LIST UTILITIES** command can be used to monitor the process of the asynchronous partition detach task. The **LIST UTILITIES** command indicates whether the asynchronous partition detach task is in one of the following states:

- Waiting for old access to the partitioned table to complete
- Finalizing the detach operation and making the target table available

Asynchronous partition detach for data partitioned tables

For Db2 Version 9.7 Fix Pack 1 and later releases, the asynchronous partition detach task completes the detach of a data partition from a partitioned table that was initiated by an ALTER TABLE...DETACH operation. The task is an asynchronous background process (ABP) that is initiated after the partition becomes a logically detached partition.

The asynchronous partition detach task accelerates the process of detaching a data partition from a partitioned table. If the partitioned table has dependent materialized query tables (MQTs), the task is not initiated until after a SET INTEGRITY statement is executed on the MQTs.

By completing the detach of the data partition asynchronously, queries accessing the partitioned table that started prior to issuing ALTER TABLE...DETACH PARTITION statement continue while the partition is immediately detached.

If there are any dependent tables that need to be incrementally maintained with respect to the detached data partition (these dependent tables are referred to as detached dependent tables), the asynchronous partition detach task starts only after the SET INTEGRITY statement is run on all detached dependent tables.

In the absence of detached dependents, the asynchronous partition detach task starts after the transaction issuing the ALTER TABLE...DETACH PARTITION statement commits.

The asynchronous partition detach task performs the following operations:

- Performs hard invalidation on cached statements on which the ALTER TABLE...DETACH operation previously performed soft invalidation.
- Updates catalog entries for source partitioned table and target stand-alone table and makes the target table available.
- For multidimensional clustering (MDC) tables with nonpartitioned block indexes and no other partitioned indexes, creates an index object for target table. The block indexes are created upon first access to the target table after the asynchronous partition detach task commits.
- Creates the system path index on the target table for table containing XML columns.
- Updates the minimum recovery time (MRT) of the table space containing the detached partition.
- Creates asynchronous index cleanup AIC tasks for nonpartitioned indexes. The AIC task performs index cleanup after asynchronous partition detach completes.
- Releases the data partition ID if nonpartitioned indexes do not exist on the table.

Asynchronous partition detach task impact on performance

An asynchronous partition detach task incurs minimal performance impact. The task waits for all access to the detached partition to complete by performing a hard invalidation on cached statements on which the ALTER TABLE...DETACH operation previously performed soft invalidation. Then the task acquires the required locks on the table and on the partition and continues the process to make the detached partition a stand-alone table.

Monitoring the asynchronous partition detach task

The distribution daemon and asynchronous partition detach task agents are internal system applications that appear in **LIST APPLICATIONS** command output with the application names **db2taskd** and **db2apd**, respectively. To prevent accidental disruption, system applications cannot be forced. The distribution daemon remains online as long as the database is active. The tasks remain active until detach completes. If the database is deactivated while detach is in progress, the asynchronous partition detach task resumes when the database is reactivated.

The **LIST UTILITIES** command indicates whether the asynchronous partition detach task is in one of the following states:

- Waiting for old access to the partitioned table to complete
- Finalizing the detach operation and making the target table available

The following sample output for the **LIST UTILITIES SHOW DETAIL** command shows asynchronous partition detach task activity in the WSDB database:

```
ID                   = 1
Type                 = ASYNCHRONOUS PARTITION DETACH
Database Name        = WSDB
Partition Number     = 0
Description          = Finalize the detach for partition '4' of table 'USER1.ORDERS'.
Start Time           = 07/15/2009 14:52:14.476131
State                = Executing
Invocation Type      = Automatic
Progress Monitoring:
    Description      = Waiting for old access to the partitioned table to complete.
    Start Time       = 07/15/2009 14:52:51.268119
```

In the output of the LIST UTILITIES command, the main description for the asynchronous partition detach task identifies the data partition being detached and the target table created by the detach operation. The progress monitoring description provides information about the current state of the asynchronous partition detach task.

Note: The asynchronous partition detach task is an asynchronous process. To know when the target table of a detach operation is available, a stored procedure can be created that queries the STATUS column of the SYSCAT.DATAPARTITIONS catalog view and returns when the detach operation completes. If non-partitioned indexes exist on the partitioned table, the target table of the detach operation becomes available when the STATUS column is changed to 'I', which indicates that asynchronous index cleanup is being performed on the non-partitioned index(es). Rows with a STATUS value of 'I' are removed when all index records referring to the detached partition have been deleted. If there are no non-partitioned indexes, then the asynchronous index cleanup phase is skipped and the row specified by the original DATAPARTITIONID is removed from SYSCAT.DATAPARTITIONS. This data partition name is changed to a system-generated name using the following form: 'SQLyymmddhhmmssxxx', which is done during the DETACH operation.

Asynchronous partition detach processing in a partitioned database environment

One asynchronous partition detach task is created for each DETACH operation independent of the number of database partitions in a partitioned database environment. The task is created on the catalog database partition and distributes work to the remaining database partitions, as needed.

Error handling for the asynchronous partition detach task

The asynchronous partition detach task is transaction based. All the changes made by a task will be rolled back internally if it fails. Any errors during asynchronous partition detach processing are logged in a **db2diag** log file. A failed task is retried later by the system.

Adding data partitions to partitioned tables

You can use the ALTER TABLE statement to modify a partitioned table after the table is created. Specifically, you can use the ADD PARTITION clause to add a new data partition to an existing partitioned table.

About this task

Adding a data partition to a partitioned table is more appropriate than attaching a data partition when data is added to the data partition over time, when data is trickling in rather than rolling in from an external source, or when you are inserting or loading data directly into a partitioned table. Specific examples include daily loads of data into a data partition for January data or ongoing inserts of individual rows.

To add the new data partition to a specific table space location, the IN clause is added as an option on the ALTER TABLE ADD PARTITION statement.

To add the partitioned index of a new data partition to a specific table space location separate from the table space location of the data partition, the partition level INDEX IN clause is added as an option on the ALTER TABLE ADD PARTITION statement. If the INDEX IN option is not specified, by default any partitioned indexes on the new data partition reside in the same table space as the data partition. If any partitioned indexes exist on the partitioned table, the ADD PARTITION clause creates the corresponding empty index partitions for the new partition. A new entry is inserted into the SYSCAT.INDEXPARTITIONS catalog view for each partitioned index.

To add the LONG, LOB, or XML data of a new data partition to a specific table space location that is separate from the table space location of the data partition, the partition-level LONG IN clause is added as an option on the ALTER TABLE ADD PARTITION statement.

When adding a data partition to a partitioned table by using the ALTER TABLE statement with the ADD PARTITION clause, the target partitioned table remains online, and dynamic queries against the table, running under the RS, CS, or UR isolation level, continue to run.

Restrictions and usage guidelines

- You cannot add a data partition to a nonpartitioned table. For details on migrating an existing table to a partitioned table, see “Migrating existing tables and views to partitioned tables” on page 148.
- The range of values for each new data partition are determined by the STARTING and ENDING clauses.

- One or both of the STARTING and ENDING clauses must be supplied.
- The new range must not overlap with the range of an existing data partition.
- When adding a new data partition before the first existing data partition, the STARTING clause must be specified. Use MINVALUE to make this range open ended.
- Likewise, the ENDING clause must be specified if you want to add a new data partition after the last existing data partition. Use MAXVALUE to make this range open ended.
- If the STARTING clause is omitted, then the database manufactures a starting bound just after the ending bound of the previous data partition. Likewise, if the ENDING clause is omitted, the database creates an ending bound just before the starting bound of the next data partition.
- The start-clause and end-clause syntax is the same as specified in the CREATE TABLE statement.
- If the IN, INDEX IN, or LONG IN clauses are not specified for ADD PARTITION, the table space in which to place the data partition is chosen by using the same method as is used by the CREATE TABLE statement.
- Packages are invalidated during the ALTER TABLE...ADD PARTITION operation.
- The newly added data partition is available once the ALTER TABLE statement is committed.
- If a table has a nonpartitioned index, you cannot access a new data partition in that table within the same transaction as the add or attach operation that created the partition, if the transaction does not have the table locked in exclusive mode (SQL0668N, reason code 11).

Omitting the STARTING or ENDING bound for an ADD operation is also used to fill a gap in range values. Here is an example of filling in a gap by using the ADD operation where only the starting bound is specified:

```
CREATE TABLE hole (c1 int) PARTITION BY RANGE (c1)
(STARTING FROM 1 ENDING AT 10, STARTING FROM 20 ENDING AT 30);
DB20000I The SQL command completed successfully.
```

```
ALTER TABLE hole ADD PARTITION STARTING 15;
DB20000I The SQL command completed successfully.
```

```
SELECT SUBSTR(tabname, 1,12) tabname,
SUBSTR(datapartitionname, 1, 12) datapartitionname,
seqno, SUBSTR(lowvalue, 1, 4) lowvalue, SUBSTR(highvalue, 1, 4) highvalue
FROM SYSCAT.DATAPARTITIONS WHERE TABNAME='HOLE' ORDER BY seqno;
```

```
TABNAME DATAPARTITIONNAME SEQNO LOWVALUE HIGHVALUE
------------ ----------------- ----------- -------- ---------
HOLE PART0 0 1 10
HOLE PART2 1 15 20
HOLE PART1 2 20 30

3 record(s) selected.
```

Example 1: Add a data partition to an existing partitioned table that holds a range of values 901 - 1000 inclusive. Assume that the SALES table holds nine ranges: 0 - 100, 101 - 200, and so on, up to the value of 900. The example adds a range at the end of the table, indicated by the exclusion of the STARTING clause:

```
ALTER TABLE sales ADD PARTITION dp10
ENDING AT 1000 INCLUSIVE
```

To add the partitioned index of a new data partition to a specific table space location separate from the table space location of the data partition, the partition level INDEX IN clause is added as an option on the ALTER TABLE ADD PARTITION statement. If no INDEX IN option is specified, by default any partitioned indexes on the new data partition reside in the same table space as the data partition. If any partitioned indexes exist on the partitioned table, ADD PARTITION creates the corresponding empty index partitions for the new partition. A new entry is inserted into the SYSCAT.INDEXPARTITIONS catalog view for each partitioned index.

Example 2: Add a data partition to an existing partitioned table by separating out long data and indexes from the rest of the data partition.

```
ALTER TABLE newbusiness ADD PARTITION IN tsnewdata
INDEX IN tsnewindex LONG IN tsnewlong
```

Dropping data partitions

To drop a data partition, you detach the partition, and drop the table created by the detach operation. Use the ALTER TABLE statement with the DETACH PARTITION clause to detach the partition and create a stand-alone table, and use the DROP TABLE statement to drop the table.

Before you begin

To detach a data partition from a partitioned table the user must have the following authorities or privileges:

- The user performing the DETACH operation must have the authority to ALTER, to SELECT from and to DELETE from the source table.
- The user must also have the authority to CREATE the target table. Therefore, in order to alter a table to detach a data partition, the privilege held by the authorization ID of the statement must include at least one of the following on the target able:
  - DBADM authority
  - CREATETAB authority on the database and USE privilege on the table spaces used by the table as well as one of:
    - IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the table does not exist
    - CREATEIN privilege on the schema, if the schema name of the table refers to an existing schema.

To drop a table the user must have the following authorities or privileges:

- You must either be the definer as recorded in the DEFINER column of SYSCAT.TABLES, or have at least one of the following privileges:
  - DBADM authority
  - DROPIN privilege on the schema for the table
  - CONTROL privilege on the table

Note: The implication of the detach data partition case is that the authorization ID of the statement is going to effectively issue a CREATE TABLE statement and therefore must have the necessary privileges to perform that operation. The table space is the one where the data partition that is being detached already resides. The authorization ID of the ALTER TABLE statement becomes the definer of the new table with CONTROL authority, as if the user issued the CREATE TABLE statement. No privileges from the table being altered are transferred to the new table. Only the authorization ID of the ALTER TABLE statement and DBADM or SYSADM have access to the data immediately after the ALTER TABLE...DETACH PARTITION operation.

Procedure

- To detach a data partition of a partitioned table, issue the ALTER TABLE statement with the DETACH PARTITION clause.

Example

In the following example, the dec01 data partition is detached from table STOCK and placed in table JUNK. After ensuring that the asynchronous partition detach task made the target table JUNK available, you can drop the table JUNK, effectively dropping the associated data partition.

```
ALTER TABLE stock DETACH PART dec01 INTO junk;
 -- After the target table becomes available, issue the DROP TABLE statement
DROP TABLE junk;
```

What to do next

To make the ALTER TABLE...DETACH as fast as possible with Db2 Version 9.7 Fix Pack 1 and later releases, the asynchronous partition detach task completes the detach operation asynchronously. If there are detached dependent tables, the asynchronous partition detach task does not start and the detached data partition does not become a stand-alone table. In this case, the SET INTEGRITY statement must be issued on all detached dependent tables. After SET INTEGRITY completes, the asynchronous partition detach task starts and makes the target table accessible. When the target table is accessible it can be dropped.

Scenario: Rotating data in a partitioned table

Rotating data in Db2 databases refers to a method of reusing space in a data partition by removing obsolete data from a table (a detach partition operation) and then adding new data (an attach partition operation).

Before you begin

Alternatively, you can archive the detached partition and load the new data into a different source table before an attach operation is performed. In the following scenario, a detach operation precedes the other steps; it could as easily be the last step, depending on your specific requirements.

To alter a table to detach a data partition, the authorization ID of the statement must hold the following privileges and authorities:

- At least one of the following authorities on the target table of the detached partition:
  - CREATETAB authority on the database, and USE privilege on the table spaces used by the table, as well as one of the following authorities or privileges:
    - IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the new table does not exist
    - CREATEIN privilege on the schema, if the schema name of the new table refers to an existing schema
  - DBADM authority
- At least one of the following privileges and authorities on the source table:
  - SELECT, ALTER, and DELETE privileges on the table
  - CONTROL privilege on the table
  - DATAACCESS authority

To alter a table to attach a data partition, the authorization ID of the statement must include the following privileges and authorities:

- At least one of the following authorities or privileges on the source table:
  - SELECT privilege on the table and DROPIN privilege on the schema of the table
  - CONTROL privilege on the table
  - DATAACCESS authority
- A least one of the following authorities or privileges on the target table:
  - ALTER and INSERT privileges on the table
  - CONTROL privilege on the table
  - DATAACCESS authority

Procedure

To rotate data in a partitioned table, issue the ALTER TABLE statement.

The following example shows how to update the STOCK table by removing the data from December 2008 and replacing it with the latest data from December 2010.

a. Remove the old data from the STOCK table.

```
ALTER TABLE stock DETACH PARTITION dec08 INTO newtable;
```

b. Load the new data. Using the **LOAD** command with the REPLACE option overwrites existing data.

```
LOAD FROM data_file OF DEL REPLACE INTO newtable
```

Note: If there are detached dependents, issue the SET INTEGRITY statement on the detached dependents before loading the detached table. If SQL20285N is returned, wait until the asynchronous partition detach task is complete before issuing the SET INTEGRITY statement again.

c. If necessary, perform data cleansing activities, which can include the following actions:

- Filling in missing values
- Deleting inconsistent and incomplete data
- Removing redundant data arriving from multiple sources
- Transforming data
  - *Normalization*. Data from different sources that represents the same value in different ways must be reconciled as part of rolling the data into the warehouse.
  - *Aggregation*. Raw data that is too detailed to store in the warehouse must be aggregated before being rolled in.

d. Attach the data as a new range.

```
ALTER TABLE stock
  ATTACH PARTITION dec10
    STARTING '12/01/2008' ENDING '12/31/2010'
  FROM newtable;
```

e. Use the SET INTEGRITY statement to update indexes and other dependent objects. Read and write access is permitted during execution of the SET INTEGRITY statement.

```
SET INTEGRITY FOR stock
  ALLOW WRITE ACCESS
  IMMEDIATE CHECKED
  FOR EXCEPTION IN stock USE stock_ex;
```

Scenarios: Rolling in and rolling out partitioned table data

A common administrative operation in data warehouses is to periodically roll in new data and roll out obsolete data. The following scenarios illustrate these tasks.

Scenario 1: Rolling out obsolete data by detaching a data partition

The following example shows how to detach an unneeded data partition (DEC01) from a partitioned table named STOCK. The detached data partition is used to create a table named STOCK_DROP without any data movement.

```
ALTER TABLE stock DETACH PART dec01 INTO stock_drop;
COMMIT WORK;
```

To expedite the detach operation, index cleanup on the source table is done automatically and in the background through an asynchronous index cleanup process. If there are no detached dependent tables defined on the source table, there is no need to issue a SET INTEGRITY statement to complete the detach operation.

The new table can be dropped or attached to another table, or it can be truncated and loaded with new data before being reattached to the source table. You can perform these operations immediately, even before asynchronous index cleanup completes, unless the source table detached dependent tables.

To determine whether a detached table is accessible, query the SYSCAT.TABDETACHEDDEP catalog view. If a detached table is found to be inaccessible, issue the SET INTEGRITY statement with the IMMEDIATE CHECKED option against all of the detached dependent tables. If you try to access a detached table before all of its detached dependent tables are maintained, an error (SQL20285N) is returned.

Scenario 2: Creating a new, empty range

The following example shows how to add an empty data partition (DEC02) to a partitioned table named STOCK. The STARTING FROM and ENDING AT clauses specify the range of values for the new data partition.

```
ALTER TABLE stock ADD PARTITION dec02
 STARTING FROM '12/01/2002' ENDING AT '12/31/2002';
```

This ALTER TABLE...ADD PARTITION statement drains existing static or repeatable-read queries that are running against the STOCK table and invalidates packages on the table; that is, the statement allows such queries to complete normally before it exclusively locks the table (by using a Z lock) and performs the add operation. Existing dynamic non-repeatable-read queries against the STOCK table continue, and can run concurrently with the add operation. Any new queries attempting to access the STOCK table after the add operation starts must wait until the transaction in which the statement is issued commits.

Tip: Issue a COMMIT statement immediately after the add operation to make the table available for use sooner.

Load data into the table:

```
LOAD FROM data_file OF DEL
 INSERT INTO stock
 ALLOW READ ACCESS;
```

Issue a SET INTEGRITY statement to validate constraints and refresh dependent materialized query tables (MQTs). Any rows that violate defined constraints are moved to the exception table STOCK_EX.

```
SET INTEGRITY FOR stock
 ALLOW READ ACCESS
 IMMEDIATE CHECKED
 FOR EXCEPTION IN stock USE stock_ex;

COMMIT WORK;
```

Scenario 3: Rolling in new data by attaching a loaded data partition

The following example shows how an attach operation can be used to facilitate loading a new range of data into an existing partitioned table (the target table named STOCK). Data is loaded into a new, empty table (DEC03), where it can be checked and cleansed, if necessary, without impacting the target table. Data cleansing activities include:

- Filling in missing values
- Deleting inconsistent and incomplete data
- Removing redundant data that arrived from multiple sources
- Transforming the data through normalization or aggregation:
  - *Normalization*. Data from different sources that represents the same values in different ways must be reconciled as part of the roll-in process.
  - *Aggregation*. Raw data that is too detailed to store in a warehouse must be aggregated during roll-in.

After the data is prepared in this way, the newly loaded data partition can be attached to the target table.

```
CREATE TABLE dec03(…);
LOAD FROM data_file OF DEL REPLACE INTO dec03;
(data cleansing, if necessary)
ALTER TABLE stock ATTACH PARTITION dec03
 STARTING FROM '12/01/2003' ENDING AT '12/31/2003'
 FROM dec03;
```

During an attach operation, one or both of the STARTING FROM and ENDING AT clauses must be specified, and the lower bound (STARTING FROM clause) must be less than or equal to the upper bound (ENDING AT clause). The newly attached data partition must not overlap an existing data partition range in the target table. If the high end of the highest existing range is defined as MAXVALUE, any attempt to attach a new high range fails, because that new range would overlap the existing high range. A similar restriction applies to low ranges that end at MINVALUE. Moreover, you cannot add or attach a new data partition in the middle, unless its new range falls within a gap in the existing ranges. If boundaries are not specified by the user, they are determined when the table is created.

This ALTER TABLE...ATTACH PARTITION statement drains existing static or repeatable-read queries that are running against the STOCK table and invalidates packages on the table; that is, the statement allows such queries to complete normally before it exclusively locks the table (by using a Z lock) and performs the attach operation. Existing dynamic non-repeatable-read queries against the STOCK table continue, and can run concurrently with the attach operation. Any new queries attempting to access the STOCK table after the attach operation starts must wait until the transaction in which the statement is issued commits.

Tip:

- Issue a COMMIT statement immediately after the attach operation to make the table available for use.
- Issue a SET INTEGRITY statement immediately after the attach operation commits to make the data from the new data partition available sooner.

The data in the attached data partition is not yet visible because it is not yet validated by the SET INTEGRITY statement. The SET INTEGRITY statement is necessary to verify that the newly attached data is within the defined range. It also performs any necessary maintenance activities on indexes and other dependent objects, such as MQTs. New data is not visible until the SET INTEGRITY statement commits; however, if the SET INTEGRITY statement is running online, existing data in the STOCK table is fully accessible for both read and write operations.

Tip: If data integrity checking, including range validation and other constraints checking, can be done through application logic that is independent of the data server before an attach operation, newly attached data can be made available for use much sooner. You can optimize the data roll-in process by using the SET INTEGRITY...ALL IMMEDIATE UNCHECKED statement to skip range and constraints violation checking. In this case, the table is brought out of SET INTEGRITY pending state, and the new data is available for applications to use immediately, as long as there are no nonpartitioned user indexes on the target table.

Note: You cannot execute data definition language (DDL) statements or utility operations against the table while the SET INTEGRITY statement is running. These operations include, but are not restricted to, the following statements and commands:

- LOAD command
- REDISTRIBUTE DATABASE PARTITION GROUP command
- REORG INDEXES/TABLE command
- ALTER TABLE statement
  - ADD COLUMN
  - ADD PARTITION
  - ATTACH PARTITION
  - DETACH PARTITION
- CREATE INDEX statement

The SET INTEGRITY statement validates the data in the newly attached data partition:

```
SET INTEGRITY FOR stock
 ALLOW WRITE ACCESS
 IMMEDIATE CHECKED
 FOR EXCEPTION IN stock USE stock_ex;
```

Committing the transaction makes the table available for use:

```
COMMIT WORK;
```

Any rows that are out of range, or that violate other constraints, are moved to the exception table STOCK_EX. You can query this table, fix the rows, and insert them into the STOCK table.

Load

Parallelism and loading

The load utility takes advantage of a hardware configuration in which multiple processors or multiple storage devices are used, such as in a symmetric multiprocessor (SMP) environment.

There are several ways in which parallel processing of large amounts of data can take place using the load utility. One way is through the use of multiple storage devices, which allows for I/O parallelism during the load operation (see Figure 37 on page 187). Another way involves the use of multiple processors in an SMP environment, which allows for intra-partition parallelism (see Figure 38 on page 187). Both can be used together to provide even faster loading of data.

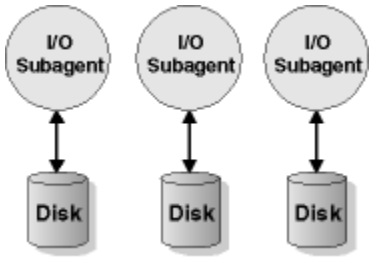


Figure 37. Taking Advantage of I/O Parallelism When Loading Data

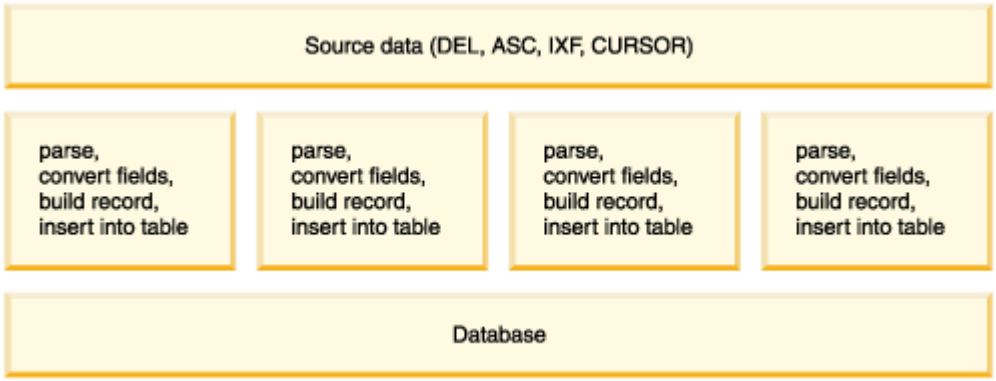


Figure 38. Taking Advantage of Intra-partition Parallelism When Loading Data

MDC and ITC considerations

The following restrictions apply when loading data into multidimensional clustering (MDC) and insert time clustering (ITC) tables:

- The SAVECOUNT option of the **LOAD** command is not supported.

- The `totalfreespace` file type modifier is not supported since these tables manage their own free space.
- The `anyorder` file type modifier is required for MDC or ITC tables. If a load is executed into an MDC or ITC table without the `anyorder` modifier, it will be explicitly enabled by the utility.

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

- If the table included a unique key before the load operation and duplicate records are loaded into the table, the original record remains and the new records are deleted during the delete phase.
- If the table did not include a unique key before the load operation and both a unique key and duplicate records are loaded into the table, only one of the records with the unique key is loaded and the others are deleted during the delete phase.

  Note: There is no explicit technique for determining which record is loaded and which is deleted.

Performance Considerations

To improve the performance of the load utility when loading MDC tables with more than one dimension, the *util_heap_sz* database configuration parameter value should be increased. The mdc-load algorithm performs significantly better when more memory is available to the utility. This reduces disk I/O during the clustering of data that is performed during the load phase. Beginning in version 9.5, the value of the DATA BUFFER option of the **LOAD** command can temporarily exceed *util_heap_sz* if more memory is available in the system. .

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

During the load phase, extra logging for the maintenance of the block map is performed. There are approximately two extra log records per extent allocated. To ensure good performance, the *logbufsz* database configuration parameter should be set to a value that takes this into account.

A system temporary table with an index is used to load data into MDC and ITC tables. The size of the table is proportional to the number of distinct cells loaded. The size of each row in the table is proportional to the size of the MDC dimension key. ITC tables only have one cell and use a 2-byte dimension key. To minimize disk I/O caused by the manipulation of this table during a load operation, ensure that the buffer pool for the temporary table space is large enough.

Load considerations for partitioned tables

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

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

General load behavior

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

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

Handling of invalid rows

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

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

History file

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

Terminating a load operation

Terminating a load replace completely truncates all visible data partitions, terminating a load insert truncates all visible data partitions to their lengths before the load. Indexes are invalidated during a termination of an ALLOW READ ACCESS load operation that failed in the load copy phase. Indexes are also invalidated when terminating an ALLOW NO ACCESS load operation that touched the index (It is invalidated because the indexing mode is rebuild, or a key was inserted during incremental maintenance leaving the index in an inconsistent state). Loading data into multiple targets does not have any effect on load recovery operations except for the inability to restart the load operation from a consistency point taken during the load phase In this case, the SAVECOUNT load option is ignored if the target table is partitioned. This behavior is consistent with loading data into a MDC target table.

Generated columns

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

Data availability

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

Important: The ALLOW READ ACCESS parameter is deprecated and might be removed in a future release. For more details, see ALLOW READ ACCESS parameter in the LOAD command is deprecated.

The ingest utility also supports partitioned tables and is better suited to allow data concurrency and availability than the LOAD command with the ALLOW READ ACCESS parameter. It can move large amounts of data from files and pipes without locking the target table. In addition, data becomes accessible as soon as it is committed based on elapsed time or number of rows.

Data partition states

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

Error isolation

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

Other considerations

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

Loading data in a partitioned database environment

Load overview-partitioned database environments

In a multi-partition database, large amounts of data are located across many database partitions. Distribution keys are used to determine on which database partition each portion of the data resides. The data must be *distributed* before it can be loaded at the correct database partition.

When loading tables in a multi-partition database, the load utility can:

- Distribute input data in parallel
- Load data simultaneously on corresponding database partitions
- Transfer data from one system to another system

Loading data into a multi-partition database takes place in two phases: the *setup phase*, during which database partition resources such as table locks are acquired, and the *load phase*, during which the data

is loaded into the database partitions. You can use the ISOLATE_PART_ERRS option of the **LOAD** command to select how errors are handled during either of these phases, and how errors on one or more of the database partitions affect the load operation on the database partitions that are not experiencing errors.

When loading data into a multi-partition database you can use one of the following modes:

PARTITION_AND_LOAD
Data is distributed (perhaps in parallel) and loaded simultaneously on the corresponding database partitions. When loading into a random distribution table that uses the random by generation method, this is the only supported mode.

PARTITION_ONLY
Data is distributed (perhaps in parallel) and the output is written to files in a specified location on each loading database partition. Each file includes a partition header that specifies how the data was distributed across the database partitions, and that the file can be loaded into the database using the LOAD_ONLY mode.

LOAD_ONLY
Data is assumed to be already distributed across the database partitions; the distribution process is skipped, and the data is loaded simultaneously on the corresponding database partitions.

LOAD_ONLY_VERIFY_PART
Data is assumed to be already distributed across the database partitions, but the data file does not contain a partition header. The distribution process is skipped, and the data is loaded simultaneously on the corresponding database partitions. During the load operation, each row is checked to verify that it is on the correct database partition. Rows containing database partition violations are placed in a dump file if the `dumpfile` file type modifier is specified. Otherwise, the rows are discarded. If database partition violations exist on a particular loading database partition, a single warning is written to the load message file for that database partition.

ANALYZE
An optimal distribution map with even distribution across all database partitions is generated.

Concepts and terminology

The following terminology is used when discussing the behavior and operation of the load utility in a partitioned database environment with multiple database partitions:

- The *coordinator partition* is the database partition to which the user connects in order to perform the load operation. In the PARTITION_AND_LOAD, PARTITION_ONLY, and ANALYZE modes, it is assumed that the data file resides on this database partition unless the CLIENT option of the **LOAD** command is specified. Specifying CLIENT indicates that the data to be loaded resides on a remotely connected client.
- In the PARTITION_AND_LOAD, PARTITION_ONLY, and ANALYZE modes, the *pre-partitioning agent* reads the user data and distributes it to the next agent in the pipeline. The actual agent depends on the distribution method.
  - For random distribution tables using random by generation method, the data is distributed in a round-robin fashion directly to the loading agents.
  - Otherwise, data is distributed in a round-robin fashion to the partitioning agents which then distribute the data. This process is always performed on the coordinator partition. A maximum of one partitioning agent is allowed per database partition for any load operation.
- In the PARTITION_AND_LOAD, LOAD_ONLY, and LOAD_ONLY_VERIFY_PART modes, *load agents* run on each output database partition and coordinate the loading of data to that database partition.
- *Load to file agents* run on each output database partition during a PARTITION_ONLY load operation. They receive data from partitioning agents and write it to a file on their database partition.
- The SOURCEUSEREXIT option provides a facility through which the load utility can execute a customized script or executable, referred to herein as the *user exit*.

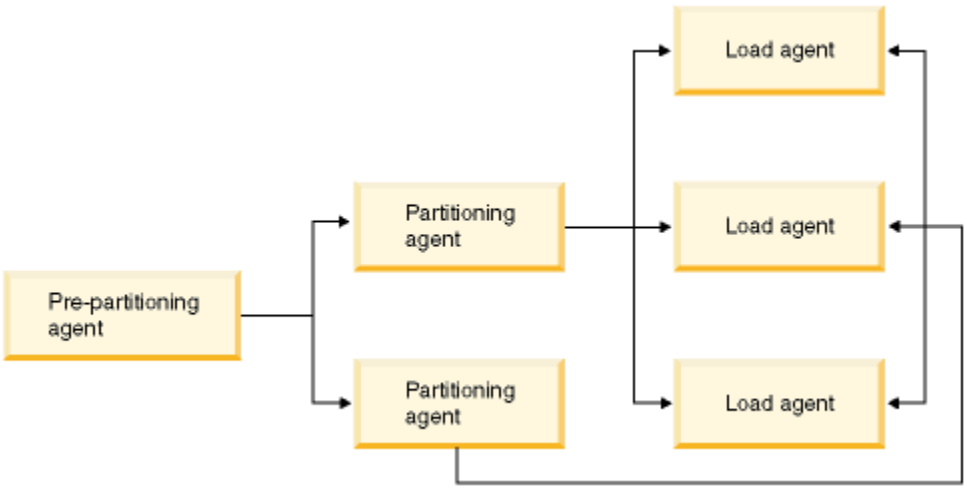


Figure 39. Partitioned Database Load Overview

Loading data in a partitioned database environment-hints and tips

The following is some information to consider before loading a table in a multi-partition database:

- Familiarize yourself with the load configuration options by using the utility with small amounts of data.
- If the input data is already sorted, or in some chosen order, and you want to maintain that order during the loading process, only one database partition should be used for distributing. Parallel distribution cannot guarantee that the data is loaded in the same order it was received. The load utility chooses a single partitioning agent by default if the `anyorder` modifier is not specified on the **LOAD** command.
- If large objects (LOBs) are being loaded from separate files (that is, if you are using the `lobsinfile` modifier through the load utility), all directories containing the LOB files must be read-accessible to all the database partitions where loading is taking place. The LOAD *lob-path* parameter must be fully qualified when working with LOBs.
- You can force a job running in a multi-partition database to continue even if the load operation detects (at startup time) that some loading database partitions or associated table spaces or tables are offline, by setting the `ISOLATE_PART_ERRS` option to SETUP_ERRS_ONLY or SETUP_AND_LOAD_ERRS.
- Use the `STATUS_INTERVAL` load configuration option to monitor the progress of a job running in a multi-partition database. The load operation produces messages at specified intervals indicating how many megabytes of data have been read by the pre-partitioning agent. These messages are dumped to the pre-partitioning agent message file. To view the contents of this file during the load operation, connect to the coordinator partition and issue a **LOAD QUERY** command against the target table.
- Better performance can be expected if the database partitions participating in the distribution process (as defined by the `PARTITIONING_DBPARTNUMS` option) are different from the loading database partitions (as defined by the `OUTPUT_DBPARTNUMS` option), since there is less contention for CPU cycles. When loading data into a multi-partition database, invoke the load utility on a database partition that is not participating in either the distributing or the loading operation.
- Specifying the `MESSAGES` parameter in the **LOAD** command saves the messages files from the pre-partitioning, partitioning, and load agents for reference at the end of the load operation. To view the contents of these files during a load operation, connect to the appropriate database partition and issue a **LOAD QUERY** command against the target table.
- The load utility chooses only one output database partition on which to collect statistics. The `RUN_STAT_DBPARTNUM` database configuration option can be used to specify the database partition.
- Before loading data in a multi-partition database, run the Design Advisor to determine the best partition for each table. For more information, see "The Design Advisor" in *Troubleshooting and Tuning Database Performance*.

Troubleshooting

If the load utility is hanging, you can:

- Use the STATUS_INTERVAL parameter to monitor the progress of a multi-partition database load operation. The status interval information is dumped to the pre-partitioning agent message file on the coordinator partition.
- Check the partitioning agent messages file to see the status of the partitioning agent processes on each database partition. If the load is proceeding with no errors, and the TRACE option has been set, there should be trace messages for a number of records in these message files.
- Check the load messages file to see if there are any load error messages.

  Note: You must specify the MESSAGES option of the **LOAD** command in order for these files to exist.
- Interrupt the current load operation if you find errors suggesting that one of the load processes encountered errors.

Loading data in a partitioned database environment

Using the load utility to load data into a partitioned database environment.

Before you begin

Before loading a table in a multi-partition database:

- Ensure that the **svcename** database manager configuration parameter and the **DB2COMM** profile registry variable are set correctly. This step is important because the load utility uses TCP/IP to transfer data from the pre-partitioning agent to the partitioning agents, and from the partitioning agents to the loading database partitions.
- Before invoking the load utility, you must be connected to (or be able to implicitly connect to) the database into which you want to load the data.
- Since the load utility issues a COMMIT statement, complete all transactions and release any locks by issuing either a COMMIT or a ROLLBACK statement before beginning the load operation. If the PARTITION_AND_LOAD, PARTITION_ONLY, or ANALYZE mode is being used, the data file that is being loaded must reside on this database partition unless:

  1. The **CLIENT** parameter has been specified, in which case the data must reside on the client machine;
  2. The input source type is CURSOR, in which case there is no input file.
- Run the **Design Advisor** to determine the best database partition for each table. For more information, see "The Design Advisor" in *Troubleshooting and Tuning Database Performance*.

Restrictions

The following restrictions apply when using the load utility to load data in a multi-partition database:

- The location of the input files to the load operation cannot be a tape device.
- The **ROWCOUNT** parameter is not supported unless the ANALYZE mode is being used.
- If the target table has an identity column that is needed for distributing and the **identityoverride** file type modifier is not specified, or if you are using multiple database partitions to distribute and then load the data, the use of a **SAVECOUNT** greater than 0 on the **LOAD** command is not supported.
- If an identity column forms part of the distribution key or it is a random distribution table using the random by generation method, only the PARTITION_AND_LOAD mode is supported.
- The LOAD_ONLY and LOAD_ONLY_VERIFY_PART modes cannot be used with the **CLIENT** parameter of the **LOAD** command.
- The LOAD_ONLY_VERIFY_PART mode cannot be used with the CURSOR input source type.
- The distribution error isolation modes LOAD_ERRS_ONLY and SETUP_AND_LOAD_ERRS cannot be used with the **ALLOW READ ACCESS** and **COPY YES** parameters of the **LOAD** command.

- Multiple load operations can load data into the same table concurrently if the database partitions specified by the**OUTPUT_DBPARTNUMS** and **PARTITIONING_DBPARTNUMS** options do not overlap. For example, if a table is defined on database partitions 0 through 3, one load operation can load data into database partitions 0 and 1 while a second load operation can load data into database partitions 2 and 3. If the database partitions specified by the **PARTITIONING_DBPARTNUMS** options do overlap, then load will automatically choose a **PARTITIONING_DBPARTNUMS** parameter where no load partitioning subagent is already executing on the table, or fail if none are available.

  Starting with Version 9.7 Fix Pack 6, if the database partitions specified by the **PARTITIONING_DBPARTNUMS** options do overlap, the load utility automatically tries to pick up a **PARTITIONING_DBPARTNUMS** parameter from the database partitions indicated by **OUTPUT_DBPARTNUMS** where no load partitioning subagent is already executing on the table, or fail if none are available.

  It is strongly recommended that if you are going to explicitly specify partitions with the **PARTITIONING_DBPARTNUMS** option, you should use that option with all concurrent **LOAD** commands, with each command specifying different partitions. If you only specify **PARTITIONING_DBPARTNUMS** on some of the concurrent load commands or if you specify overlapping partitions, the **LOAD** command will need to pick alternate partitioning nodes for at least some of the concurrent loads, and in rare cases the command might fail (SQL2038N).
- Only non-delimited ASCII (ASC) and Delimited ASCII (DEL) files can be distributed across tables spanning multiple database partitions. PC/IXF files cannot be distributed, however, you can load a PC/IXF file into a table that is distributed over multiple database partitions by using the load operation in the LOAD_ONLY_VERIFY_PART mode.

Examples

The following examples illustrate how to use the **LOAD** command to initiate various types of load operations. The database used in the following examples has five database partitions: 0, 1, 2, 3 and 4. Each database partition has a local directory /db2/data/. Two tables, TABLE1 and TABLE2, are defined on database partitions 0, 1, 3 and 4. When loading from a client, the user has access to a remote client that is not one of the database partitions.

Distribute and load example

In this scenario, you are connected to a database partition that might or might not be a database partition where TABLE1 is defined. The data file `load.del` resides in the current working directory of this database partition. To load the data from `load.del` into all of the database partitions where TABLE1 is defined, issue the following command:

```
LOAD FROM LOAD.DEL of DEL REPLACE INTO TABLE1
```

Note: In this example, default values are used for all of the configuration parameters for partitioned database environments: The **MODE** parameter defaults to PARTITION_AND_LOAD. The **OUTPUT_DBPARTNUMS** parameter defaults to all database partitions on which TABLE1 is defined. The **PARTITIONING_DBPARTNUMS** defaults to the set of database partitions selected according to the **LOAD** command rules for choosing database partitions when none are specified.

To perform a load operation where data is distributed over database partitions 3 and 4, issue the following command:

```
LOAD FROM LOAD.DEL of DEL REPLACE INTO TABLE1
    PARTITIONED DB CONFIG PARTITIONING_DBPARTNUMS (3,4)
```

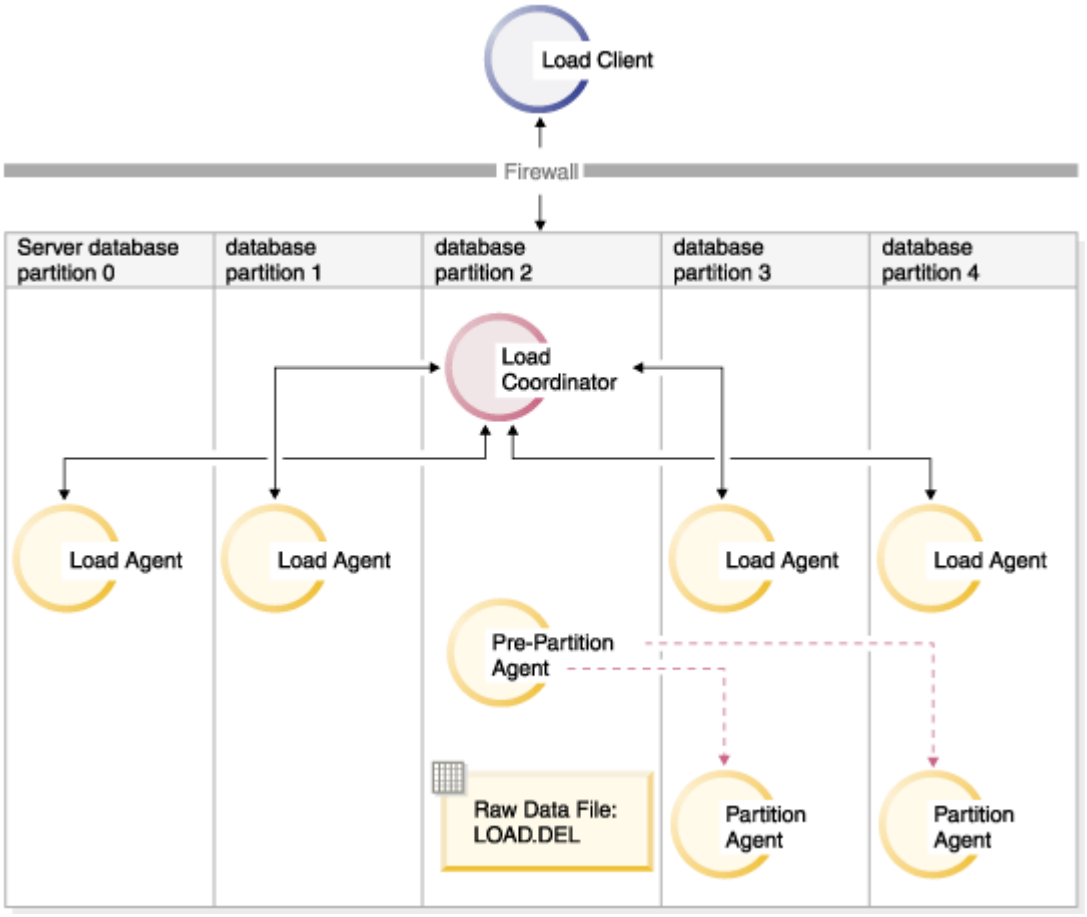


Figure 40. Loading data into database partitions 3 and 4.

Distribute only example

In this scenario, you are connected to a database partition that might or might not be a database partition where TABLE1 is defined. The data file `load.del` resides in the current working directory of this database partition. To distribute (but not load) `load.del` to all the database partitions on which TABLE1 is defined, using database partitions 3 and 4 issue the following command:

```
LOAD FROM LOAD.DEL of DEL REPLACE INTO TABLE1
    PARTITIONED DB CONFIG MODE PARTITION_ONLY
    PART_FILE_LOCATION /db2/data
    PARTITIONING_DBPARTNUMS (3,4)
```

This results in a file `load.del.`*xxx* being stored in the `/db2/data` directory on each database partition, where *xxx* is a three-digit representation of the database partition number.

To distribute the `load.del` file to database partitions 1 and 3, using only one partitioning agent running on database partition 0 (which is the default for **PARTITIONING_DBPARTNUMS**), issue the following command:

```
LOAD FROM LOAD.DEL OF DEL REPLACE INTO TABLE1
    PARTITIONED DB CONFIG MODE PARTITION_ONLY
    PART_FILE_LOCATION /db2/data
    OUTPUT_DBPARTNUMS (1,3)
```

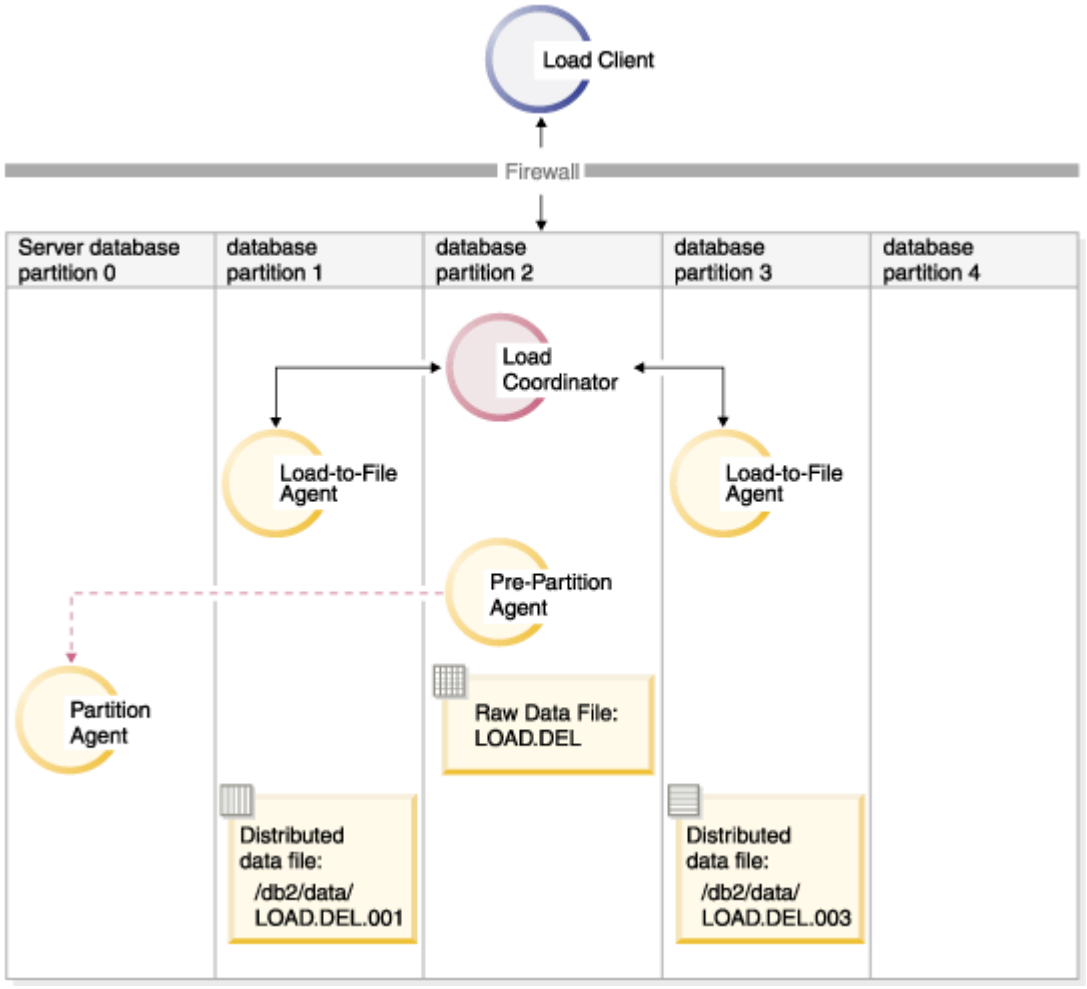


Figure 41. Loading data into database partitions 1 and 3 using one partitioning agent.

Load only example

If you have already performed a load operation in the PARTITION_ONLY mode and want to load the partitioned files in the /db2/data directory of each loading database partition to all the database partitions on which TABLE1 is defined, issue the following command:

```
LOAD FROM LOAD.DEL OF DEL REPLACE INTO TABLE1
    PARTITIONED DB CONFIG MODE LOAD_ONLY
    PART_FILE_LOCATION /db2/data
```

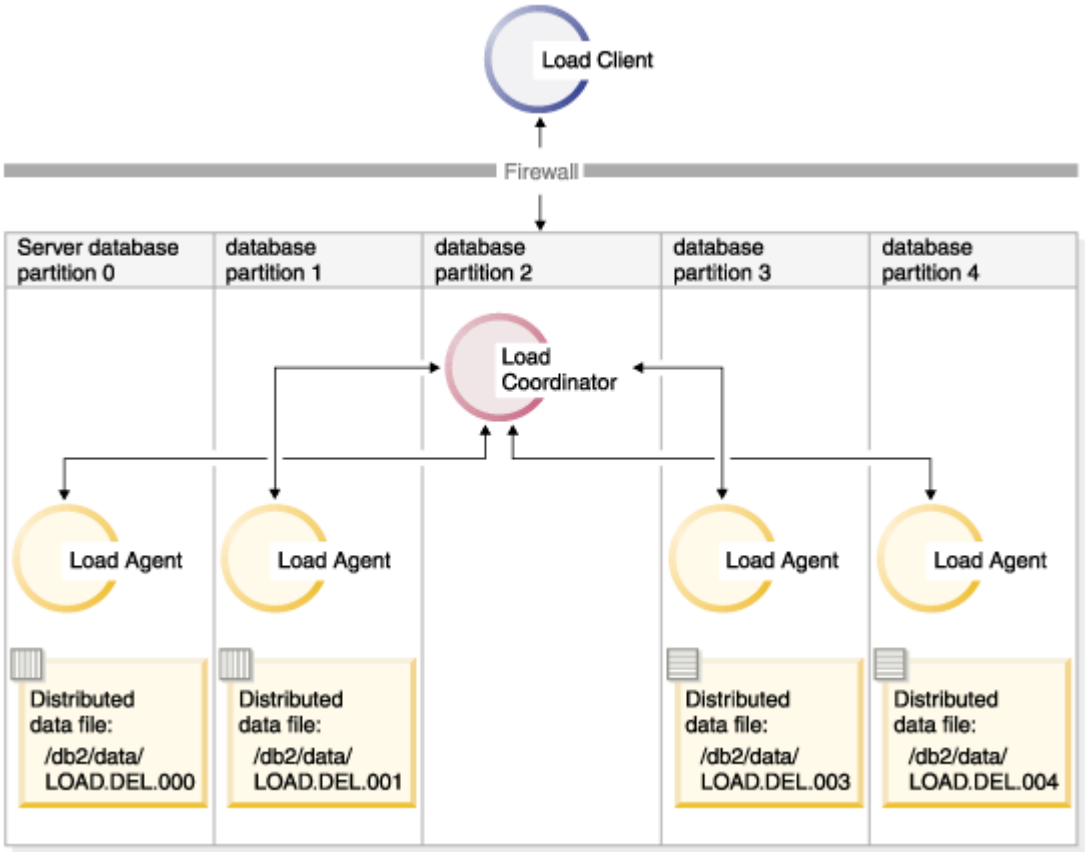


Figure 42. Loading data into all database partitions where a specific table is defined.

To load into database partition 4 only, issue the following command:

```
LOAD FROM LOAD.DEL OF DEL REPLACE INTO TABLE1
    PARTITIONED DB CONFIG MODE LOAD_ONLY
    PART_FILE_LOCATION /db2/data
    OUTPUT_DBPARTNUMS (4)
```

Loading pre-distributed files without distribution map headers

The **LOAD** command can be used to load data files without distribution headers directly into several database partitions. If the data files exist in the `/db2/data` directory on each database partition where TABLE1 is defined and have the name `load.del.`*xxx*, where *xxx* is the database partition number, the files can be loaded by issuing the following command:

```
LOAD FROM LOAD.DEL OF DEL modified by dumpfile=rejected.rows
    REPLACE INTO TABLE1
    PARTITIONED DB CONFIG MODE LOAD_ONLY_VERIFY_PART
    PART_FILE_LOCATION /db2/data
```

To load the data into database partition 1 only, issue the following command:

```
LOAD FROM LOAD.DEL OF DEL modified by dumpfile=rejected.rows
    REPLACE INTO TABLE1
    PARTITIONED DB CONFIG MODE LOAD_ONLY_VERIFY_PART
    PART_FILE_LOCATION /db2/data
    OUTPUT_DBPARTNUMS (1)
```

Note: Rows that do not belong on the database partition from which they were loaded are rejected and put into the dump file, if one has been specified.

Loading from a remote client to a multi-partition database

To load data into a multi-partition database from a file that is on a remote client, you must specify the **CLIENT** parameter of the **LOAD** command. This parameter indicates that the data file is not on a server partition. For example:

```
LOAD CLIENT FROM LOAD.DEL OF DEL REPLACE INTO TABLE1
```

Note: You cannot use the LOAD_ONLY or LOAD_ONLY_VERIFY_PART modes with the **CLIENT** parameter.

Loading from a cursor

As in a single-partition database, you can load from a cursor into a multi-partition database. In this example, for the PARTITION_ONLY and LOAD_ONLY modes, the **PART_FILE_LOCATION** parameter must specify a fully qualified file name. This name is the fully qualified base file name of the distributed files that are created or loaded on each output database partition. Multiple files can be created with the specified base name if there are LOB columns in the target table.

To distribute all the rows in the answer set of the statement SELECT * FROM TABLE1 to a file on each database partition named /db2/data/select.out.*xxx* (where *xxx* is the database partition number), for future loading into TABLE2, issue the following commands:

```
DECLARE C1 CURSOR FOR SELECT * FROM TABLE1

LOAD FROM C1 OF CURSOR REPLACE INTO TABLE2
    PARTITIONED DB CONFIG MODE PARTITION_ONLY
    PART_FILE_LOCATION /db2/data/select.out
```

The data files produced by the previous operation can then be loaded by issuing the following **LOAD** command:

```
LOAD FROM C1 OF CURSOR REPLACE INTO TABLE2
    PARTITIONED CB CONFIG MODE LOAD_ONLY
    PART_FILE_LOCATION /db2/data/select.out
```

Monitoring a load operation in a partitioned database environment using the LOAD QUERY command

During a load operation in a partitioned database environment, message files are created by some of the load processes on the database partitions where they are being executed.

The message files store all information, warning, and error messages produced during the execution of the load operation. The load processes that produce message files that can be viewed by the user are the load agent, pre-partitioning agent, and partitioning agent. The content of the message file is only available after the load operation is finished.

You can connect to individual database partitions during a load operation and issue the **LOAD QUERY** command against the target table. When issued from the CLP, this command displays the contents of the load message files that currently reside on that database partition for the table that is specified in the **LOAD QUERY** command.

For example, table TABLE1 is defined on database partitions 0 through 3 in database WSDB. You are connected to database partition 0 and issue the following **LOAD** command:

```
load from load.del of del replace into table1 partitioned db config
    partitioning_dbpartnums (1)
```

This command initiates a load operation that includes load agents running on database partitions 0, 1, 2, and 3; a partitioning agent running on database partition 1; and a pre-partitioning agent running on database partition 0.

Database partition 0 contains one message file for the pre-partitioning agent and one for the load agent on that database partition. To view the contents of these files at the same time, start a new session and issue the following commands from the CLP:

```
set client connect_node 0
connect to wsdb
load query table table1
```

Database partition 1 contains one file for the load agent and one for the partitioning agent. To view the contents of these files, start a new session and issue the following commands from the CLP:

```
set client connect_node 1
connect to wsdb
load query table table1
```

Note: The messages generated by the STATUS_INTERVAL load configuration option appear in the pre-partitioning agent message file. To view these message during a load operation, you must connect to the coordinator partition and issue the **LOAD QUERY** command.

Saving the contents of message files

If a load operation is initiated through the **db2Load** API, the messages option (piLocalMsgFileName) must be specified and the message files are brought from the server to the client and stored for you to view.

For multi-partition database load operations initiated from the CLP, the message files are not displayed to the console or retained. To save or view the contents of these files after a multi-partition database load is complete, the MESSAGES option of the **LOAD** command must be specified. If this option is used, once the load operation is complete the message files on each database partition are transferred to the client machine and stored in files using the base name indicated by the MESSAGES option. For multi-partition database load operations, the name of the file corresponding to the load process that produced it is listed in the following table:

Process type	File name
Load Agent	*<message-file-name>*.LOAD.*<dbpartition-number>*
Partitioning Agent	*<message-file-name>*.PART.*<dbpartition-number>*
Pre-partitioning Agent	*<message-file-name>*.PREP.*<dbpartition-number>*

For example, if the MESSAGES option specifies `/wsdb/messages/load`, the load agent message file for database partition 2 is `/wsdb/messages/load.LOAD.002`.

Note: It is strongly recommended that the MESSAGES option be used for multi-partition database load operations initiated from the CLP.

Resuming, restarting, or terminating load operations in a partitioned database environment

The steps you need to take following failed load operations in a partitioned database environment depend on when the failure occurred.

The load process in a multi-partition database consists of two stages:

1. The *setup stage*, during which database partition-level resources such as table locks on output database partitions are acquired

   In general, if a failure occurs during the setup stage, restart and terminate operations are not necessary. What you need to do depends on the error isolation mode that was specified for the failed load operation.

   If the load operation specified that setup stage errors were not to be isolated, the entire load operation is canceled and the state of the table on each database partition is rolled back to the state it was in before the load operation.

   If the load operation specified that setup stage errors were to be isolated, the load operation continues on the database partitions where the setup stage was successful, but the table on each of

the failing database partitions is rolled back to the state it was in before the load operation. This means that a single load operation can fail at different stages if some partitions fail during the setup stage and others fail during the load stage

2. The *load stage*, during which data is formatted and loaded into tables on the database partitions

   If a load operation fails on at least one database partition during the load stage of a multi-partition database load operation, a **LOAD RESTART** or **LOAD TERMINATE** command must be issued. This is necessary because loading data in a multi-partition database is done through a single transaction.

   If you can fix the problems that caused the failed load to occur, choose a **LOAD RESTART**. This saves time because if a load restart operation is initiated, the load operation continues from where it left off on all database partitions.

   If you want the table returned to the state it was in before the initial load operation, choose a **LOAD TERMINATE**.

Determining when a load failed

The first thing you need to do if your load operation in a partitioned environment fails is to determine on which partitions it failed and at what stage each of them failed. This is done by looking at the partition summary. If the **LOAD** command was issued from the CLP, the partition summary is displayed at the end of the load (see following example). If the **LOAD** command was issued from the db2Load API, the partition summary is contained in the **poAgentInfoList** field of the db2PartLoadOut structure.

If there is an entry of "LOAD" for "Agent Type", for a given partition, then that partition reached the load stage, otherwise a failure occurred during the setup stage. A negative SQL Code indicates that it failed. In the following example, the load failed on partition 1 during the load stage.

```
   Agent Type     Node     SQL Code     Result
   _____________________________________________________________
   LOAD           000      +00000000    Success.
   _____________________________________________________________
   LOAD           001      -00000289    Error. May require RESTART.
   _____________________________________________________________
   LOAD           002      +00000000    Success.
   _____________________________________________________________
   LOAD           003      +00000000    Success.
.
.
.
```

Resuming, restarting, or terminating a failed load

Only loads with the ISOLATE_PART_ERRS option specifying SETUP_ERRS_ONLY or SETUP_AND_LOAD_ERRS should fail during the setup stage. For loads that fail on at least one output database partition fail during this stage, you can issue a **LOAD REPLACE** or **LOAD INSERT** command. Use the OUTPUT_DBPARTNUMS option to specify only those database partitions on which it failed.

For loads that fail on at least one output database partition during the load stage, issue a **LOAD RESTART** or **LOAD TERMINATE** command.

For loads that fail on at least one output database partition during the setup stage and at least one output database partition during the load stage, you need to perform two load operations to resume the failed load-one for the setup stage failures and one for the load stage failures, as previously described. To effectively undo this type of failed load operation, issue a **LOAD TERMINATE** command. However, after issuing the command, you must account for all partitions because no changes were made to the table on the partitions that failed during the setup stage, and all the changes are undone for the partitions that failed during the load stage.

For example, TABLE1 is defined on database partitions 0 through 3 in database WSDB. The following command is issued:

```
load from load.del of del insert into table1 partitioned db config
isolate_part_errs setup_and_load_errs
```

There is a failure on output database partition 1 during the setup stage. Since setup stage errors are isolated, the load operation continues, but there is a failure on partition 3 during the load stage. To resume the load operation, you would issue the following commands:

```
load from load.del of del replace into table1 partitioned db config
output_dbpartnums (1)
```

```
load from load.del of del restart into table1 partitioned db config
isolate_part_errs setup_and_load_errs
```

Note: For load restart operations, the options specified in the **LOAD RESTART** command are honored, so it is important that they are identical to the ones specified in the original **LOAD** command.

Load configuration options for partitioned database environments

There are a number of configuration options that you can use to modify a load operation in a partitioned database environment.

MODE *X*

Specifies the mode in which the load operation occurs when loading a multi-partition database. PARTITION_AND_LOAD is the default. Valid values are:

- PARTITION_AND_LOAD. Data is distributed (perhaps in parallel) and loaded simultaneously on the corresponding database partitions.
- PARTITION_ONLY. Data is distributed (perhaps in parallel) and the output is written to files in a specified location on each loading database partition. For file types other than CURSOR, the format of the output file name on each database partition is *filename.xxx*, where *filename* is the input file name specified in the **LOAD** command and *xxx* is the 3-digit database partition number. For the CURSOR file type, the name of the output file on each database partition is determined by the PART_FILE_LOCATION option. See the PART_FILE_LOCATION option for details on how to specify the location of the distribution file for each database partition.

  Note:

  1. This mode cannot be used for a CLI load operation.
  2. If the table contains an identity column that is needed for distribution, then this mode is not supported, unless the **identityoverride** file type modifier is specified.
  3. This mode cannot be used for random distribution tables that use the random by generation method.
  4. Distribution files generated for file type CURSOR are not compatible between Db2 releases. This means that distribution files of file type CURSOR that were generated in a previous release cannot be loaded using the LOAD_ONLY mode. Similarly, distribution files of file type CURSOR that were generated in the current release cannot be loaded in a future release using the LOAD_ONLY mode.
- LOAD_ONLY. Data is assumed to be already distributed; the distribution process is skipped, and the data is loaded simultaneously on the corresponding database partitions. For file types other than CURSOR, the format of the input file name for each database partition should be *filename.xxx*, where *filename* is the name of the file specified in the **LOAD** command and *xxx* is the 3-digit database partition number. For the CURSOR file type, the name of the input file on each database partition is determined by the PART_FILE_LOCATION option. See the PART_FILE_LOCATION option for details on how to specify the location of the distribution file for each database partition.

  Note:

  1. This mode cannot be used for a CLI load operation, or when the **CLIENT** parameter of **LOAD** command is specified.
  2. If the table contains an identity column that is needed for distribution, then this mode is not supported, unless the **identityoverride** file type modifier is specified.
  3. This mode cannot be used for random distribution tables that use the random by generation method.

- LOAD_ONLY_VERIFY_PART. Data is assumed to be already distributed, but the data file does not contain a partition header. The distributing process is skipped, and the data is loaded simultaneously on the corresponding database partitions. During the load operation, each row is checked to verify that it is on the correct database partition. Rows containing database partition violations are placed in a dump file if the **dumpfile** file type modifier is specified. Otherwise, the rows are discarded. If database partition violations exist on a particular loading database partition, a single warning is written to the load message file for that database partition. The format of the input file name for each database partition should be *filename.xxx*, where *filename* is the name of the file specified in the **LOAD** command and *xxx* is the 3-digit database partition number. See the PART_FILE_LOCATION option for details on how to specify the location of the distribution file for each database partition.

  Note:

  1. This mode cannot be used for a CLI load operation, or when the **CLIENT** parameter of **LOAD** command is specified.
  2. If the table contains an identity column that is needed for distribution, then this mode is not supported, unless the **identityoverride** file type modifier is specified.
  3. This mode cannot be used for random distribution tables that use the random by generation method.
- ANALYZE. An optimal distribution map with even distribution across all database partitions is generated.

PART_FILE_LOCATION *X*
In the PARTITION_ONLY, LOAD_ONLY, and LOAD_ONLY_VERIFY_PART modes, this parameter can be used to specify the location of the distributed files. This location must exist on each database partition specified by the OUTPUT_DBPARTNUMS option. If the location specified is a relative path name, the path is appended to the current directory to create the location for the distributed files.

For the CURSOR file type, this option must be specified, and the location must refer to a fully qualified file name. This name is the fully qualified base file name of the distributed files that are created on each output database partition in the PARTITION_ONLY mode, or the location of the files to be read from for each database partition in the LOAD_ONLY mode. When using the PARTITION_ONLY mode, multiple files can be created with the specified base name if the target table contains LOB columns.

For file types other than CURSOR, if this option is not specified, the current directory is used for the distributed files.

OUTPUT_DBPARTNUMS *X*
X represents a list of database partition numbers. The database partition numbers represent the database partitions on which the load operation is to be performed. Any data that does not partition to any of the database partitions listed will not be loaded. Unless we are loading a random distribution table that uses random by generation method. In that case all data will be loaded into the set of database partitions listed.

The database partition numbers must be a subset of the database partitions on which the table is defined, except for column-organized tables, in which case all database partitions must be specified (SQL27906N). All database partitions are selected by default. The list must be enclosed in parentheses and the items in the list must be separated by commas. Ranges are permitted (for example, (0, 2 to 10, 15)).

PARTITIONING_DBPARTNUMS *X*
X represents a list of database partition numbers that are used in the distribution process. The list must be enclosed in parentheses and the items in the list must be separated by commas. Ranges are permitted (for example, (0, 2 to 10, 15)). The database partitions specified for the distribution process can be different from the database partitions being loaded. If PARTITIONING_DBPARTNUMS is not specified, the load utility determines how many database partitions are needed and which database partitions to use in order to achieve optimal performance.

If the **anyorder** file type modifier is not specified in the **LOAD** command, only one partitioning agent is used in the load session. Furthermore, if there is only one database partition specified for the

OUTPUT_DBPARTNUMS option, or the coordinator partition of the load operation is not an element of OUTPUT_DBPARTNUMS, the coordinator partition of the load operation is used in the distribution process. Otherwise, the first database partition (not the coordinator partition) in OUTPUT_DBPARTNUMS is used in the distribution process.

If the **anyorder** file type modifier is specified, the number of database partitions used in the distribution process is determined as follows: (number of partitions in OUTPUT_DBPARTNUMS/4 + 1).

This option is ignored when loading random distribution tables using the random by generation method. That distribution method does not use partitioning agents.

MAX_NUM_PART_AGENTS *X*
Specifies the maximum numbers of partitioning agents to be used in a load session. The default is 25. This option has no affect when loading into a random distribution table using random by generation method. That distribution method does not use partitioning agents.

ISOLATE_PART_ERRS *X*
Indicates how the load operation reacts to errors that occur on individual database partitions. The default is LOAD_ERRS_ONLY, unless both the **ALLOW READ ACCESS** and **COPY YES** parameters of the **LOAD** command are specified, in which case the default is NO_ISOLATION. Valid values are:

- SETUP_ERRS_ONLY. Errors that occur on a database partition during setup, such as problems accessing a database partition, or problems accessing a table space or table on a database partition, cause the load operation to stop on the failing database partitions but to continue on the remaining database partitions. Errors that occur on a database partition while data is being loaded cause the entire operation to fail.
- LOAD_ERRS_ONLY. Errors that occur on a database partition during setup cause the entire load operation to fail. If an error occurs while data is being loaded, the load operation will stop on the database partition where the error occurred. The load operation continues on the remaining database partitions until a failure occurs or until all the data is loaded. The newly loaded data will not be visible until a load restart operation is performed and completes successfully.

  Note: This mode cannot be used when both the **ALLOW READ ACCESS** and the **COPY YES** parameters of the **LOAD** command are specified.
- SETUP_AND_LOAD_ERRS. In this mode, database partition-level errors during setup or loading data cause processing to stop only on the affected database partitions. As with the LOAD_ERRS_ONLY mode, when partition errors do occur while data is loaded, newly loaded data will not be visible until a load restart operation is performed and completes successfully.

  Note: This mode cannot be used when both the **ALLOW READ ACCESS** and the **COPY YES** options of the **LOAD** command are specified.
- NO_ISOLATION. Any error during the load operation causes the load operation to fail.

STATUS_INTERVAL *X*
X represents how often you are notified of the volume of data that has been read. The unit of measurement is megabytes (MB). The default is 100 MB. Valid values are whole numbers from 1 to 4000.

PORT_RANGE *X*
X represents the range of TCP ports used to create sockets for internal communications. The default range is from 49152 to 65535. If defined at the time of invocation, the value of the **DB2ATLD_PORTS** registry variable replaces the value of the PORT_RANGE load configuration option. For the **DB2ATLD_PORTS** registry variable, the range should be provided in the following format:

```
<lower-port-number:higher-port-number>
```

From the CLP, the format is:

```
( lower-port-number, higher-port-number )
```

CHECK_TRUNCATION
Specifies that the program should check for truncation of data records at input/output. The default behavior is that data is not checked for truncation at input/output.

MAP_FILE_INPUT *X*
X specifies the input file name for the distribution map. This parameter must be specified if the distribution map is customized, as it points to the file containing the customized distribution map. A customized distribution map can be created by using the **db2gpmap** program to extract the map from the database system catalog table, or by using the ANALYZE mode of the **LOAD** command to generate an optimal map. The map generated by using the ANALYZE mode must be moved to each database partition in your database before the load operation can proceed.

MAP_FILE_OUTPUT *X*
X represents the output filename for the distribution map. The output file is created on the database partition issuing the **LOAD** command assuming that database partition is participating in the database partition group where partitioning is performed. If the **LOAD** command is invoked on a database partition that is not participating in partitioning (as defined by **PARTITIONING_DBPARTNUMS**), the output file is created at the first database partition defined with the **PARTITIONING_DBPARTNUMS** parameter. Consider the following partitioned database environment setup:

```
1 serv1 0
2 serv1 1
3 serv2 0
4 serv2 1
5 serv3 0
```

Running the following **LOAD** command on serv3, creates the distribution map on serv1.

```
LOAD FROM file OF ASC METHOD L ( ...) INSERT INTO table CONFIG
MODE ANALYZE PARTITIONING_DBPARTNUMS(1,2,3,4)
MAP_FILE_OUTPUT '/home/db2user/distribution.map'
```

This parameter should be used when the ANALYZE mode is specified. An optimal distribution map with even distribution across all database partitions is generated. If this parameter is not specified and the ANALYZE mode is specified, the program exits with an error.

TRACE *X*
Specifies the number of records to trace when you require a review of a dump of the data conversion process and the output of the hashing values. The default is `0`.

NEWLINE
Used when the input data file is an ASC file with each record delimited by a new line character and the **reclen** file type modifier is specified in the **LOAD** command. When this option is specified, each record is checked for a new line character. The record length, as specified in the **reclen** file type modifier, is also checked.

DISTFILE *X*
If this option is specified, the load utility generates a database partition distribution file with the given name. The database partition distribution file contains 32 768 integers: one for each entry in the distribution map for the target table. Each integer in the file represents the number of rows in the input files being loaded that hashed to the corresponding distribution map entry. This information can help you identify skew in your data and also help you decide whether a new distribution map should be generated for the table using the ANALYZE mode of the utility. If this option is not specified, the default behavior of the load utility is to not generate the distribution file.

Note: When this option is specified, a maximum of one partitioning agent is used for the load operation. Even if you explicitly request multiple partitioning agents, only one is used.

OMIT_HEADER
Specifies that a distribution map header should not be included in the distribution file. If not specified, a header is generated.

RUN_STAT_DBPARTNUM *X*
If the **STATISTICS USE PROFILE** parameter is specified in the **LOAD** command, statistics are collected only on one database partition. This parameter specifies on which database partition to

collect statistics. If the value is -1 or not specified at all, statistics are collected on the first database partition in the output database partition list.

Load sessions in a partitioned database environment - CLP examples

The following examples demonstrate loading data in a multi-partition database.

The database has four database partitions numbered 0 through 3. Database WSDB is defined on all of the database partitions, and table TABLE1 resides in the default database partition group which is also defined on all of the database partitions.

Example 1

To load data into TABLE1 from the user data file `load.del` which resides on database partition 0, connect to database partition 0 and then issue the following command:

```
load from load.del of del replace into table1
```

If the load operation is successful, the output will be as follows:

```
Agent Type     Node     SQL Code     Result
_____________________________________________________
LOAD           000      +00000000    Success.
_____________________________________________________
LOAD           001      +00000000    Success.
_____________________________________________________
LOAD           002      +00000000    Success.
_____________________________________________________
LOAD           003      +00000000    Success.
_____________________________________________________
PARTITION      001      +00000000    Success.
_____________________________________________________
PRE_PARTITION  000      +00000000    Success.
_____________________________________________________
RESULTS:       4 of 4 LOADs completed successfully.
_____________________________________________________

Summary of Partitioning Agents:
Rows Read                   = 100000
Rows Rejected               = 0
Rows Partitioned            = 100000

Summary of LOAD Agents:
Number of rows read         = 100000
Number of rows skipped      = 0
Number of rows loaded       = 100000
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 100000
```

The output indicates that there was one load agent on each database partition and each ran successfully. It also shows that there was one pre-partitioning agent running on the coordinator partition and one partitioning agent running on database partition 1. These processes completed successfully with a normal SQL return code of 0. The statistical summary shows that the pre-partitioning agent read 100,000 rows, the partitioning agent distributed 100,000 rows, and the sum of all rows loaded by the load agents is 100,000.

Example 2

In the following example, data is loaded into TABLE1 in the `PARTITION_ONLY` mode. The distributed output files is stored on each of the output database partitions in the directory `/db/data`:

```
load from load.del of del replace into table1 partitioned db config mode
   partition_only part_file_location /db/data
```

The output from the load command is as follows:

```
Agent Type     Node     SQL Code     Result
_____________________________________________________
```

```
LOAD_TO_FILE   000      +00000000    Success.
_____________________________________________________
LOAD_TO_FILE   001      +00000000    Success.
_____________________________________________________
LOAD_TO_FILE   002      +00000000    Success.
_____________________________________________________
LOAD_TO_FILE   003      +00000000    Success.
_____________________________________________________
PARTITION      001      +00000000    Success.
_____________________________________________________
PRE_PARTITION  000      +00000000    Success.
_____________________________________________________

Summary of Partitioning Agents:
Rows Read                  = 100000
Rows Rejected              = 0
Rows Partitioned           = 100000
```

The output indicates that there was a load-to-file agent running on each output database partition, and these agents ran successfully. There was a pre-partitioning agent on the coordinator partition, and a partitioning agent running on database partition 1. The statistical summary indicates that 100,000 rows were successfully read by the pre-partitioning agent and 100,000 rows were successfully distributed by the partitioning agent. Since no rows were loaded into the table, no summary of the number of rows loaded appears.

Example 3

To load the files that were generated during the PARTITION_ONLY load operation shown previously, issue the following command:

```
load from load.del of del replace into table1 partitioned db config mode
    load_only part_file_location /db/data
```

The output from the load command will be as follows:

```
Agent Type     Node     SQL Code     Result
_____________________________________________________
LOAD           000      +00000000    Success.
_____________________________________________________
LOAD           001      +00000000    Success.
_____________________________________________________
LOAD           002      +00000000    Success.
_____________________________________________________
LOAD           003      +00000000    Success.
_____________________________________________________
RESULTS:       4 of 4 LOADs completed successfully.
_____________________________________________________

Summary of LOAD Agents:
Number of rows read         = 100000
Number of rows skipped      = 0
Number of rows loaded       = 100000
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 100000
```

The output indicates that the load agents on each output database partition ran successfully and that the sum of the number of rows loaded by all load agents is 100,000. No summary of rows distributed is indicated since distribution was not performed.

Example 4

If the following LOAD command is issued:

```
load from load.del of del replace into table1
```

and one of the loading database partitions runs out of space in the table space during the load operation, the following output might be returned:

```
SQL0289N  Unable to allocate new pages in table space "DMS4KT".
SQLSTATE=57011
```

```
Agent Type     Node     SQL Code     Result
_______________________________________________________________
LOAD           000      +00000000    Success.
_______________________________________________________________
LOAD           001      -00000289    Error. May require RESTART.
_______________________________________________________________
LOAD           002      +00000000    Success.
_______________________________________________________________
LOAD           003      +00000000    Success.
_______________________________________________________________
PARTITION      001      +00000000    Success.
_______________________________________________________________
PRE_PARTITION  000      +00000000    Success.
_______________________________________________________________
RESULTS:       3 of 4 LOADs completed successfully.
---------------------------------------------------------------

Summary of Partitioning Agents:
Rows Read                   = 0
Rows Rejected               = 0
Rows Partitioned            = 0

Summary of LOAD Agents:
Number of rows read         = 0
Number of rows skipped      = 0
Number of rows loaded       = 0
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 0
```

The output indicates that the load operation returned error SQL0289. The database partition summary indicates that database partition 1 ran out of space. If additional space is added to the containers of the table space on database partition 1, the load operation can be restarted as follows:

```
load from load.del of del restart into table1
```

Migration and version compatibility

The **DB2_PARTITIONEDLOAD_DEFAULT** registry variable can be used to revert to pre-Db2 Universal Database Version 8 load behavior in a multi-partition database.

Note: The **DB2_PARTITIONEDLOAD_DEFAULT** registry variable is deprecated and might be removed in a later release.

Reverting to the pre-Db2 Version 8 behavior of the **LOAD** command in a multi-partition database, allows you to load a file with a valid distribution header into a single database partition without specifying any extra partitioned database configuration options. You can do this by setting the value of **DB2_PARTITIONEDLOAD_DEFAULT** to NO. You might choose to use this option if you want to avoid modifying existing scripts that issue the **LOAD** command against single database partitions. For example, to load a distribution file into database partition 3 of a table that resides in a database partition group with four database partitions, issue the following command:

```
db2set DB2_PARTITIONEDLOAD_DEFAULT=NO
```

Then issue the following commands from the Db2 Command Line Processor:

```
CONNECT RESET

SET CLIENT CONNECT_NODE 3

CONNECT TO DB MYDB

LOAD FROM LOAD.DEL OF DEL REPLACE INTO TABLE1
```

In a multi-partition database, when no multi-partition database load configuration options are specified, the load operation takes place on all the database partitions on which the table is defined. The input file does not require a distribution header, and the MODE option defaults to PARTITION_AND_LOAD. To load a single database partition, the OUTPUT_DBPARTNUMS option must be specified.

Migration of partitioned database environments

Migrating partitioned databases

Migrating partitioned database environments requires that you install the latest release of the database product on all database partition servers, migrate the instances and then migrate the databases.

You can migrate database partition servers from the catalog database partition server or any other database partition server. Should the migration process fail, you can retry migration from the catalog database partition server or any other database partition server again.

Since a migration of this sort is a significant undertaking, a description of the migration procedure, its prerequisites and restrictions, is beyond the scope of this book. A detailed description is provided in the topic "Migrating partitioned database environments" in the *Migration Guide*, which, in addition, will refer you to numerous other topics to review prior to performing the migration.

Using snapshot and event monitors

Using snapshot monitor data to monitor the reorganization of a partitioned table

The following information describes some of the most useful methods of monitoring the global status of a table reorganization.

About this task

There is no separate data group indicating the overall table reorganization status for a partitioned table. A partitioned table uses a data organization scheme in which table data is divided across multiple storage objects, called data partitions or ranges, according to values in one or more table partitioning key columns of the table. However, you can deduce the global status of a table reorganization from the values of elements in the individual data partition data group being reorganized. The following information describes some of the most useful methods of monitoring the global status of a table reorganization.

Determining the number of data partitions being reorganized
You can determine the total number of data partitions being reorganized on a table by counting the number of monitor data blocks for table data that have the same table name and schema name. This value indicates the number of data partitions on which reorganization has started. Examples 1 and 2 indicate that three data partitions are being reorganized.

Identifying the data partition being reorganized
You can deduce the current data partition being reorganized from the phase start time (reorg_phase_start). During the SORT/BUILD/REPLACE phase, the monitor data corresponding to the data partition that is being reorganized shows the most recent phase start time. During the INDEX_RECREATE phase, the phase start time is the same for all the data partitions. In Examples 1 and 2, the INDEX_RECREATE phase is indicated, so the start time is the same for all the data partitions.

Identifying an index rebuild requirement
You can determine if an index rebuild is required by obtaining the value of the maximum reorganize phase element (reorg_max_phase), corresponding to any one of the data partitions being reorganized. If reorg_max_phase has a value of 3 or 4, then an Index Rebuild is required. Examples 1 and 2 report a reorg_max_phase value of 3, indicating an index rebuild is required.

Examples

The following sample output is from a three-node server that contains a table with three data partitions:

```
CREATE TABLE  sales (c1 INT, c2 INT, c3 INT)
    PARTITION BY RANGE (c1)
        (PART P1 STARTING FROM (1)  ENDING AT (10) IN parttbs,
         PART P2 STARTING FROM (11) ENDING AT (20) IN parttbs,
```

```
        PART P3 STARTING FROM (21) ENDING AT (30) IN parttbs)
    DISTRIBUTE BY (c2)
```

Statement executed:

```
REORG TABLE sales ALLOW NO ACCESS ON ALL DBPARTITIONNUMS
```

Example 1:

```
GET SNAPSHOT FOR TABLES ON DPARTDB GLOBAL
```

The output is modified to include table information for the relevant table only.

```
            Table Snapshot

First database connect timestamp     = 06/28/2005 13:46:43.061690
Last reset timestamp                 = 06/28/2005 13:46:47.440046
Snapshot timestamp                   = 06/28/2005 13:46:50.964033
Database name                        = DPARTDB
Database path                        = /work/sales/NODE0000/SQL00001/
Input database alias                 = DPARTDB
Number of accessed tables            = 5

Table List
 Table Schema        = NEWTON
 Table Name          = SALES
 Table Type          = User
 Data Partition Id   = 0
 Data Object Pages   = 3
 Rows Read           = 12
 Rows Written        = 1
 Overflows           = 0
 Page Reorgs         = 0
 Table Reorg Information:
   Node number       = 0
   Reorg Type        =
        Reclaiming
        Table Reorg
        Allow No Access
        Recluster Via Table Scan
        Reorg Data Only
   Reorg Index       = 0
   Reorg Tablespace  = 3
 Long Temp space ID  = 3
   Start Time        = 06/28/2005 13:46:49.816883
   Reorg Phase       = 3 - Index Recreate
   Max Phase         = 3
   Phase Start Time  = 06/28/2005 13:46:50.362918
   Status            = Completed
   Current Counter   = 0
   Max Counter       = 0
   Completion        = 0
   End Time          = 06/28/2005 13:46:50.821244

 Table Reorg Information:
   Node number       = 1
   Reorg Type        =
        Reclaiming
        Table Reorg
        Allow No Access
        Recluster Via Table Scan
        Reorg Data Only
   Reorg Index       = 0
   Reorg Tablespace  = 3
 Long Temp space ID  = 3
   Start Time        = 06/28/2005 13:46:49.822701
   Reorg Phase       = 3 - Index Recreate
   Max Phase         = 3
   Phase Start Time  = 06/28/2005 13:46:50.420741
   Status            = Completed
   Current Counter   = 0
   Max Counter       = 0
   Completion        = 0
   End Time          = 06/28/2005 13:46:50.899543


 Table Reorg Information:
   Node number       = 2
```

```
  Reorg Type        =
       Reclaiming
       Table Reorg
       Allow No Access
       Recluster Via Table Scan
       Reorg Data Only
  Reorg Index       = 0
  Reorg Tablespace  = 3
Long Temp space ID  = 3
  Start Time        = 06/28/2005 13:46:49.814813
  Reorg Phase       = 3 - Index Recreate
  Max Phase         = 3
  Phase Start Time  = 06/28/2005 13:46:50.344277
  Status            = Completed
  Current Counter   = 0
  Max Counter       = 0
  Completion        = 0
  End Time          = 06/28/2005 13:46:50.803619


Table Schema        = NEWTON
Table Name          = SALES
Table Type          = User
Data Partition Id   = 1
Data Object Pages   = 3
Rows Read           = 8
Rows Written        = 1
Overflows           = 0
Page Reorgs         = 0
Table Reorg Information:
  Node number       = 0
  Reorg Type        =
       Reclaiming
       Table Reorg
       Allow No Access
       Recluster Via Table Scan
       Reorg Data Only
  Reorg Index       = 0
  Reorg Tablespace  = 3
Long Temp space ID  = 3
  Start Time        = 06/28/2005 13:46:50.014617
  Reorg Phase       = 3 - Index Recreate
  Max Phase         = 3
  Phase Start Time  = 06/28/2005 13:46:50.362918
  Status            = Completed
  Current Counter   = 0
  Max Counter       = 0
  Completion        = 0
  End Time          = 06/28/2005 13:46:50.821244

Table Reorg Information:
  Node number       = 1
  Reorg Type        =
       Reclaiming
       Table Reorg
       Allow No Access
       Recluster Via Table Scan
       Reorg Data Only
  Reorg Index       = 0
  Reorg Tablespace  = 3
Long Temp space ID  = 3
  Start Time        = 06/28/2005 13:46:50.026278
  Reorg Phase       = 3 - Index Recreate
  Max Phase         = 3
  Phase Start Time  = 06/28/2005 13:46:50.420741
  Status            = Completed
  Current Counter   = 0
  Max Counter       = 0
  Completion        = 0
  End Time          = 06/28/2005 13:46:50.899543

Table Reorg Information:
  Node number       = 2
  Reorg Type        =
       Reclaiming
       Table Reorg
       Allow No Access
       Recluster Via Table Scan
       Reorg Data Only
  Reorg Index       = 0
  Reorg Tablespace  = 3
Long Temp space ID  = 3
```

```
  Start Time         = 06/28/2005 13:46:50.006392
  Reorg Phase        = 3 - Index Recreate
  Max Phase          = 3
  Phase Start Time   = 06/28/2005 13:46:50.344277
  Status             = Completed
  Current Counter    = 0
  Max Counter        = 0
  Completion         = 0
  End Time           = 06/28/2005 13:46:50.803619


Table Schema         = NEWTON
Table Name           = SALES
Table Type           = User
Data Partition Id    = 2
Data Object Pages    = 3
Rows Read            = 4
Rows Written         = 1
Overflows            = 0
Page Reorgs          = 0
Table Reorg Information:
  Node number        = 0
  Reorg Type         =
       Reclaiming
       Table Reorg
       Allow No Access
       Recluster Via Table Scan
       Reorg Data Only
  Reorg Index        = 0
  Reorg Tablespace   = 3
Long Temp space ID   = 3
  Start Time         = 06/28/2005 13:46:50.199971
  Reorg Phase        = 3 - Index Recreate
  Max Phase          = 3
  Phase Start Time   = 06/28/2005 13:46:50.362918
  Status             = Completed
  Current Counter    = 0
  Max Counter        = 0
  Completion         = 0
  End Time           = 06/28/2005 13:46:50.821244

Table Reorg Information:
  Node number        = 1
  Reorg Type         =
       Reclaiming
       Table Reorg
       Allow No Access
       Recluster Via Table Scan
       Reorg Data Only
  Reorg Index        = 0
  Reorg Tablespace   = 3
Long Temp space ID   = 3
  Start Time         = 06/28/2005 13:46:50.223742
  Reorg Phase        = 3 - Index Recreate
  Max Phase          = 3
  Phase Start Time   = 06/28/2005 13:46:50.420741
  Status             = Completed
  Current Counter    = 0
  Max Counter        = 0
  Completion         = 0
  End Time           = 06/28/2005 13:46:50.899543

Table Reorg Information:
  Node number        = 2
  Reorg Type         =
       Reclaiming
       Table Reorg
       Allow No Access
       Recluster Via Table Scan
       Reorg Data Only
  Reorg Index        = 0
  Reorg Tablespace   = 3
Long Temp space ID   = 3
  Start Time         = 06/28/2005 13:46:50.179922
  Reorg Phase        = 3 - Index Recreate
  Max Phase          = 3
  Phase Start Time   = 06/28/2005 13:46:50.344277
  Status             = Completed
  Current Counter    = 0
  Max Counter        = 0
```

```
   Completion        = 0
   End Time          = 06/28/2005 13:46:50.803619
```

Example 2:

```
GET SNAPSHOT FOR TABLES ON DPARTDB AT DBPARTITIONNUM 2
```

The output is modified to include table information for the relevant table only.

```
            Table Snapshot

First database connect timestamp     = 06/28/2005 13:46:43.617833
Last reset timestamp                 =
Snapshot timestamp                   = 06/28/2005 13:46:51.016787
Database name                        = DPARTDB
Database path                        = /work/sales/NODE0000/SQL00001/
Input database alias                 = DPARTDB
Number of accessed tables            = 3

Table List
 Table Schema        = NEWTON
 Table Name          = SALES
 Table Type          = User
 Data Partition Id   = 0
 Data Object Pages   = 1
 Rows Read           = 0
 Rows Written        = 0
 Overflows           = 0
 Page Reorgs         = 0
 Table Reorg Information:
   Node number       = 2
   Reorg Type        =
        Reclaiming
        Table Reorg
        Allow No Access
        Recluster Via Table Scan
        Reorg Data Only
   Reorg Index       = 0
   Reorg Tablespace  = 3
 Long Temp space ID  = 3
   Start Time        = 06/28/2005 13:46:49.814813
   Reorg Phase       = 3 - Index Recreate
   Max Phase         = 3
   Phase Start Time  = 06/28/2005 13:46:50.344277
   Status            = Completed
   Current Counter   = 0
   Max Counter       = 0
   Completion        = 0
   End Time          = 06/28/2005 13:46:50.803619


 Table Schema        = NEWTON
 Table Name          = SALES
 Table Type          = User
 Data Partition Id   = 1
 Data Object Pages   = 1
 Rows Read           = 0
 Rows Written        = 0
 Overflows           = 0
 Page Reorgs         = 0
 Table Reorg Information:
   Node number       = 2
   Reorg Type        =
        Reclaiming
        Table Reorg
        Allow No Access
        Recluster Via Table Scan
        Reorg Data Only
   Reorg Index       = 0
   Reorg Tablespace  = 3
 Long Temp space ID  = 3
   Start Time        = 06/28/2005 13:46:50.006392
   Reorg Phase       = 3 - Index Recreate
   Max Phase         = 3
   Phase Start Time  = 06/28/2005 13:46:50.344277
   Status            = Completed
   Current Counter   = 0
   Max Counter       = 0
```

```
    Completion        = 0
    End Time          = 06/28/2005 13:46:50.803619


  Table Schema        = NEWTON
  Table Name          = SALES
  Table Type          = User
  Data Partition Id   = 2
  Data Object Pages   = 1
  Rows Read           = 4
  Rows Written        = 1
  Overflows           = 0
  Page Reorgs         = 0
  Table Reorg Information:
    Node number       = 2
    Reorg Type        =
         Reclaiming
         Table Reorg
         Allow No Access
         Recluster Via Table Scan
         Reorg Data Only
    Reorg Index       = 0
    Reorg Tablespace  = 3
  Long Temp space ID  = 3
    Start Time        = 06/28/2005 13:46:50.179922
    Reorg Phase       = 3 - Index Recreate
    Max Phase         = 3
    Phase Start Time  = 06/28/2005 13:46:50.344277
    Status            = Completed
    Current Counter   = 0
    Max Counter       = 0
    Completion        = 0
    End Time          = 06/28/2005 13:46:50.803619
```

Example 3:

```
SELECT * FROM SYSIBMADM.SNAPLOCK WHERE tabname = 'SALES';
```

The output is modified to include a subset of table information for the relevant table only.

```
 ...   TBSP_NAME TABNAME LOCK_OBJECT_TYPE   LOCK_MODE  LOCK_STATUS ...
       --------- ------- ------------------ ----------
-----------
 ...   PARTTBS   SALES      ROW_LOCK        X          GRNT        ...
 ...   -         SALES      TABLE_LOCK      IX         GRNT        ...
 ...   PARTTBS   SALES      TABLE_PART_LOCK IX         GRNT        ...
 ...   PARTTBS   SALES      ROW_LOCK        X          GRNT        ...
 ...   -         SALES      TABLE_LOCK      IX         GRNT        ...
 ...   PARTTBS   SALES      TABLE_PART_LOCK IX         GRNT        ...
 ...   PARTTBS   SALES      ROW_LOCK        X          GRNT        ...
 ...   -         SALES      TABLE_LOCK      IX         GRNT        ...
 ...   PARTTBS   SALES      TABLE_PART_LOCK IX         GRNT        ...

 9 record(s) selected.
```

Output from this query (continued).

```
 ... LOCK_ESCALATION LOCK_ATTRIBUTES DATA_PARTITION_ID
DBPARTITIONNUM
     --------------- --------------- ----------------- --------------
 ...               0 INSERT           2                 2
 ...               0 NONE             -                 2
 ...               0 NONE             2                 2
 ...               0 INSERT           0                 0
 ...               0 NONE             -                 0
 ...               0 NONE             0                 0
 ...               0 INSERT           1                 1
 ...               0 NONE             -                 1
 ...               0 NONE             1                 1
```

Example 4:

```
SELECT * FROM SYSIBMADM.SNAPTAB WHERE tabname = 'SALES';
```

The output is modified to include a subset of table information for the relevant table only.

```
... TABSCHEMA TABNAME TAB_FILE_ID TAB_TYPE   DATA_OBJECT_PAGES ROWS_WRITTEN ...
... --------- ------- ----------- ---------- ----------------- ------------ ...
... NEWTON    SALES             2 USER_TABLE                 1            1 ...
... NEWTON    SALES             4 USER_TABLE                 1            1 ...
... NEWTON    SALES             3 USER_TABLE                 1            1 ...

 3 record(s) selected.
```

Output from this query (continued).

```
... OVERFLOW_ACCESSES PAGE_REORGS DBPARTITIONNUM TBSP_ID DATA_PARTITION_ID
... ----------------- ----------- -------------- ------- -----------------
...                 0           0              0       3                 0
...                 0           0              2       3                 2
...                 0           0              1       3                 1
```

Example 5:

```
SELECT * FROM SYSIBMADM.SNAPTAB_REORG WHERE tabname = 'SALES';;
```

The output is modified to include a subset of table information for the relevant table only.

```
 REORG_PHASE   REORG_MAX_PHASE REORG_TYPE                                    ...
    ------------- -------------- ------------------------------------------
INDEX_RECREATE              3  RECLAIM+OFFLINE+ALLOW_NONE+TABLESCAN+DATAONLY  ...
INDEX_RECREATE              3  RECLAIM+OFFLINE+ALLOW_NONE+TABLESCAN+DATAONLY  ...
INDEX_RECREATE              3  RECLAIM+OFFLINE+ALLOW_NONE+TABLESCAN+DATAONLY  ...
INDEX_RECREATE              3  RECLAIM+OFFLINE+ALLOW_NONE+TABLESCAN+DATAONLY  ...
INDEX_RECREATE              3  RECLAIM+OFFLINE+ALLOW_NONE+TABLESCAN+DATAONLY  ...
INDEX_RECREATE              3  RECLAIM+OFFLINE+ALLOW_NONE+TABLESCAN+DATAONLY  ...
INDEX_RECREATE              3  RECLAIM+OFFLINE+ALLOW_NONE+TABLESCAN+DATAONLY  ...
INDEX_RECREATE              3  RECLAIM+OFFLINE+ALLOW_NONE+TABLESCAN+DATAONLY  ...
INDEX_RECREATE              3  RECLAIM+OFFLINE+ALLOW_NONE+TABLESCAN+DATAONLY  ...

9 record(s) selected.
```

Output from this query (continued).

```
... REORG_STATUS REORG_TBSPC_ID DBPARTITIONNUM DATA_PARTITION_ID
    ------------------------- -------------- -----------------
... COMPLETED                3              2                 0
... COMPLETED                3              2                 1
... COMPLETED                3              2                 2
... COMPLETED                3              1                 0
... COMPLETED                3              1                 1
... COMPLETED                3              1                 2
... COMPLETED                3              0                 0
... COMPLETED                3              0                 1
... COMPLETED                3              0                 2
```

Example 6:

The Table Reorg Information includes information about reclaiming extents as part of a reorganization operation. The example that follows shows the relevant output.

```
db2 -v "get snapshot for tables on wsdb"

 Table Reorg Information:
   Reorg Type        =
```

```
        Reclaim Extents
        Allow Write Access
   Reorg Index       = 0
   Reorg Tablespace  = 0
   Start Time        = 10/22/2008 15:49:35.477532
   Reorg Phase       = 12 - Release
   Max Phase         = 3
```

Note: Any snapshot requests from a monitor version before SQLM_DBMON_VERSION9_7 will not return any Reclaim Reorg status to the requesting client.

Global snapshots on partitioned database systems

On a partitioned database system, you can use the snapshot monitor to take a snapshot of the current partition, a specified partition, or all partitions. When taking a global snapshot across all the partitions of a partitioned database, data is aggregated before the results are returned.

Data is aggregated for the different element types as follows:

- **Counters, Time, and Gauges**

  Contains the sum of all like values collected from each partition in the instance. For example, GET SNAPSHOT FOR DATABASE XYZ ON TEST GLOBAL would return the number of rows read (rows_read) from the database for all partitions in the partitioned database instance.

- **Watermarks**

  Returns the highest (for high water) or lowest (for low water) value found for any partition in the partitioned database system. If the value returned is of concern, then snapshots for individual partitions can be taken to determine if a particular partition is over utilized, or if the problem is instance-wide.

- **Timestamp**

  Set to the timestamp value for the partition where the snapshot monitor instance agent is attached. Note that all timestamp values are under control of the `timestamp` monitor switch.

- **Information**

  Returns the most significant information for a partition that may be impeding work. For example, for the element `appl_status`, if the status on one partition was UOW Executing, and on another partition Lock Wait, Lock Wait would be returned, since it is the state that's holding up execution of the application.

You can also reset counters, set monitor switches, and retrieve monitor switch settings for individual partitions or all partitions in your partitioned database.

Note: When taking a global snapshot, if one or more partitions encounter an error, then data is collected from the partitions where the snapshot was successful and a warning (sqlcode 1629) is also returned. If a global get or update of monitor switches, or a counter reset fails on one or more partitions, then those partitions will not have their switches set, or data reset.

Creating an event monitor for partitioned databases, or for databases in a Db2 pureScale environment

Generally speaking, event monitors on partitioned database systems, or in a Db2 pureScale environment work similarly to ones that run on single-member databases. However, there are some differences to keep in mind when you create an event monitor for these environments.

Procedure

- Specify the partition to be monitored.

```
CREATE EVENT MONITOR tabmon FOR TABLES
                    WRITE TO FILE '/tmp/tabevents'
                    ON PARTITION 3
```

`tabmon` represents the name of the event monitor.

`/tmp/tab events` is the name of the directory path (on UNIX) where the event monitor is to write the event files.

3 represents the partition number to be monitored.

- Specify if the event monitor data is to be collected at a local or global scope.
  For example, to collect event monitor reports from all partitions issue the following statement:

```
CREATE EVENT MONITOR dlmon FOR DEADLOCKS
                    WRITE TO FILE '/tmp/dlevents'
                    ON PARTITION 3 GLOBAL
```

  Note: Only deadlock and deadlock with details event monitors can be defined as GLOBAL.

  All partitions report deadlock-related event records to partition 3.
- To collect event monitor reports from only the local partition issue the following statement:

```
CREATE EVENT MONITOR dlmon FOR TABLES
                    WRITE TO FILE '/tmp/dlevents'
                    ON PARTITION 3 LOCAL
```

  This is the default behavior for file and pipe event monitors in partitioned databases. The LOCAL and GLOBAL clauses are ignored for write-to-table event monitors.
- It is possible to review the monitor partition and scope values for existing event monitors. To do this query the SYSCAT.EVENTMONITORS table with the following statement:

```
SELECT EVMONNAME, NODENUM, MONSCOPE FROM SYSCAT.EVENTMONITORS
```

Results

After an event monitor is created and activated, it records monitoring data as its specified events occur.

Developing a good backup and recovery strategy

Crash recovery

Transactions (or units of work) against a database can be interrupted unexpectedly. If a failure occurs before all of the changes that are part of the unit of work are completed, committed, and written to disk, the database is left in an inconsistent and unusable state. *Crash recovery* is the process by which the database is moved back to a consistent and usable state. This is done by rolling back incomplete transactions and completing committed transactions that were still in memory when the crash occurred (Figure 43 on page 217).

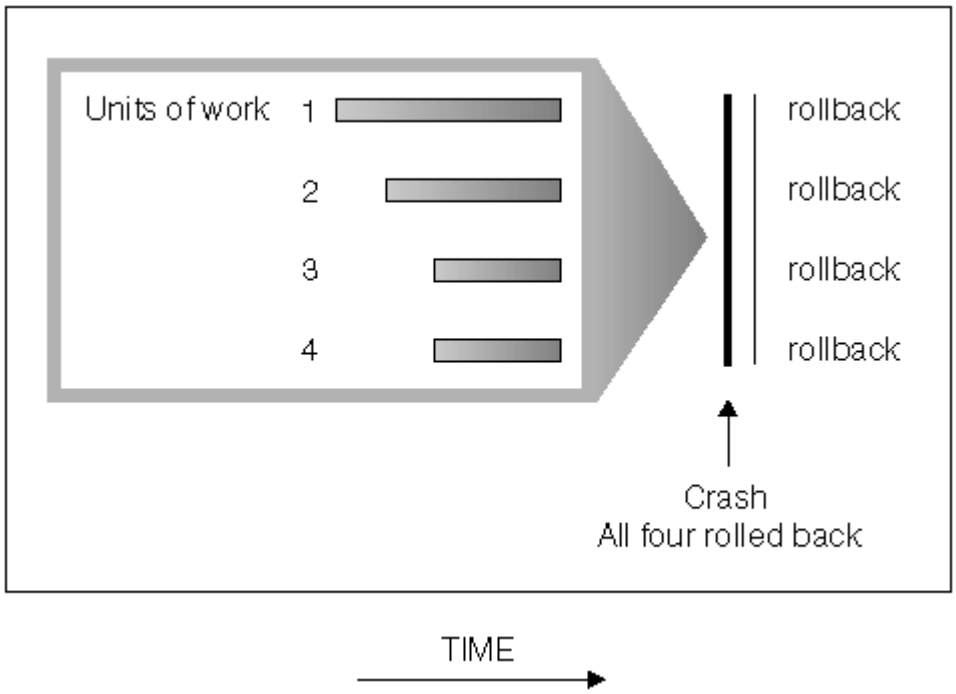


Figure 43. Rolling back units of work (crash recovery)

If the database or the database manager fails, the database can be left in an inconsistent state. The contents of the database might include changes made by transactions that were incomplete at the time of failure. The database might also be missing changes that were made by transactions that completed before the failure but which were not yet flushed to disk. A crash recovery operation must be performed in order to roll back the partially completed transactions and to write to disk the changes of completed transactions that were previously made only in memory.

Conditions that can necessitate a crash recovery include:

- A power failure on the machine, causing the database manager and the database partitions on it to go down.
- A hardware failure such as memory, disk, CPU, or network failure.
- A serious operating system error that causes the Db2 instance to end abnormally.

If you want crash recovery to be performed automatically by the database manager, enable the automatic restart (**autorestart**) database configuration parameter by setting it to ON. (This is the default value.) If you do not want automatic restart behavior, set the **autorestart** database configuration parameter to OFF. As a result, you must issue the **RESTART DATABASE** command when a database failure occurs. If the database I/O was suspended before the crash occurred, you must specify the **WRITE RESUME** option of the **RESTART DATABASE** command in order for the crash recovery to continue.

If you are using the IBM Db2 pureScale Feature, there are two specific types of crash recovery to be aware of: *member crash recovery* and *group crash recovery*. Member crash recovery is the process of recovering a portion of a database using the log stream of a single member after a member failure. Member crash recovery, which is usually initiated automatically as a part of a member restart, is an online operation-meaning that other members can still access the database. Multiple members can be undergoing member crash recovery at the same time. Group crash recovery is the process of recovering a database using multiple members' log streams after a failure that causes no viable cluster caching facility to remain in the cluster. Group crash recovery is also usually initiated automatically (as a part of a group restart) and the database is inaccessible while it is in progress, as with Db2 crash recovery operations outside of a Db2 pureScale environment.

If crash recovery occurs on a database that is enabled for rollforward recovery (that is, the **logarchmeth1** configuration parameter is not set to OFF), and an error occurs during crash recovery that is attributable to an individual table space, that table space is taken offline, and cannot be accessed until

it is repaired. Crash recovery continues on other table spaces. At the completion of crash recovery, the other table spaces in the database are accessible, and connections to the database can be established. However, if the table space that is taken offline is the table space that contains the system catalogs, it must be repaired before any connections are permitted. This behavior does not apply to Db2 pureScale environments. If an error occurs during member crash recovery or group crash recovery, the crash recovery operation fails.

If the database is configured for connectivity during crash recovery, the database might become connectable while crash recovery is in progress. Tables, indexes or objects that are still undergoing rollback will be locked in exclusive mode or super exclusive mode. For more information, see Database accessibility during backward phase of crash recovery or HADR takeover.

Recovering from transaction failures in a partitioned database environment

If a transaction failure occurs in a partitioned database environment, database recovery is usually necessary on both the failed database partition server and any other database partition server that was participating in the transaction.

There are two types of database recovery:

- Crash recovery occurs on the failed database partition server after the failure condition is corrected.
- *Database partition failure recovery* on the other (still active) database partition servers occurs immediately after the failure has been detected.

In a partitioned database environment, the database partition server on which a transaction is submitted is the coordinator partition, and the first agent that processes the transaction is the coordinator agent. The coordinator agent is responsible for distributing work to other database partition servers, and it keeps track of which ones are involved in the transaction. When the application issues a COMMIT statement for a transaction, the coordinator agent commits the transaction by using the two-phase commit protocol. During the first phase, the coordinator partition distributes a PREPARE request to all the other database partition servers that are participating in the transaction. These servers then respond with one of the following:

READ-ONLY
No data change occurred at this server

YES
Data change occurred at this server

NO
Because of an error, the server is not prepared to commit

If one of the servers responds with a NO, the transaction is rolled back. Otherwise, the coordinator partition begins the second phase.

During the second phase, the coordinator partition writes a COMMIT log record, then distributes a COMMIT request to all the servers that responded with a YES. After all the other database partition servers have committed, they send an acknowledgement of the COMMIT to the coordinator partition. The transaction is complete when the coordinator agent has received all COMMIT acknowledgments from all the participating servers. At this point, the coordinator agent writes a FORGET log record.

Transaction failure recovery on an active database partition server

If any database partition server detects that another server is down, all work that is associated with the failed database partition server is stopped:

- If the still active database partition server is the coordinator partition for an application, and the application was running on the failed database partition server (and not ready to COMMIT), the coordinator agent is interrupted to do failure recovery. If the coordinator agent is in the second phase of COMMIT processing, SQL0279N is returned to the application, which in turn loses its database connection. Otherwise, the coordinator agent distributes a ROLLBACK request to all other servers participating in the transaction, and SQL1229N is returned to the application.

- If the failed database partition server was the coordinator partition for the application, then agents that are still working for the application on the active servers are interrupted to do failure recovery. The transaction is rolled back locally on each database partition where the transaction is not in prepared state. On those database partitions where the transaction is in a prepared state, the transaction becomes an indoubt transaction. The coordinator database partition is not aware that the transaction is indoubt on some database partitions because the coordinator database partition is not available.
- If the application connected to the failed database partition server (before it failed), but neither the local database partition server nor the failed database partition server is the coordinator partition, agents working for this application are interrupted. The coordinator partition will either send a ROLLBACK or a DISCONNECT message to the other database partition servers. The transaction will only be indoubt on database partition servers that are still active if the coordinator partition returns SQL0279.

Any process (such as an agent or deadlock detector) that attempts to send a request to the failed server is informed that it cannot send the request.

Transaction failure recovery on the failed database partition server

If the transaction failure causes the database manager to end abnormally, you can issue the **db2start** command with the RESTART option to restart the database manager once the database partition has been restarted. If you cannot restart the database partition, you can issue **db2start** to restart the database manager on a different database partition.

If the database manager ends abnormally, database partitions on the server can be left in an inconsistent state. To make them usable, crash recovery can be triggered on a database partition server:

- Explicitly, through the **RESTART DATABASE** command
- Implicitly, through a CONNECT request when the *autorestart* database configuration parameter has been set to ON

Crash recovery reapplies the log records in the active log files to ensure that the effects of all complete transactions are in the database. After the changes have been reapplied, all uncommitted transactions are rolled back locally, *except* for indoubt transactions. There are two types of indoubt transaction in a partitioned database environment:

- On a database partition server that is not the coordinator partition, a transaction is indoubt if it is prepared but not yet committed.
- On the coordinator partition, a transaction is indoubt if it is committed but not yet logged as complete (that is, the FORGET record is not yet written). This situation occurs when the coordinator agent has not received all the COMMIT acknowledgments from all the servers that worked for the application.

Crash recovery attempts to resolve all the indoubt transactions by doing one of the following. The action that is taken depends on whether the database partition server was the coordinator partition for an application:

- If the server that restarted is not the coordinator partition for the application, it sends a query message to the coordinator agent to discover the outcome of the transaction.
- If the server that restarted *is* the coordinator partition for the application, it sends a message to all the other agents (subordinate agents) that the coordinator agent is still waiting for COMMIT acknowledgments.

It is possible that crash recovery might not be able to resolve all the indoubt transactions as part of crash recovery. For example, some of the database partition servers might not be available. If the coordinator partition completes crash recovery before other database partitions involved in the transaction, crash recovery will not be able to resolve the indoubt transaction. This is expected because crash recovery is performed by each database partition independently. In Db2 V11.5.3 and later versions, and when **DB2_DPF_ASYNC_INDOUBT_RESOLUTION** is enabled, the coordinator partition will continue to attempt to resolve the indoubt transactions with the other database partitions until the indoubt is finally resolved. Prior to Db2 V11.5.3, or when **DB2_DPF_ASYNC_INDOUBT_RESOLUTION** is OFF, the coordinator partition will only make one attempt to resolve the indoubt transaction and if the indoubt transaction cannot be

resolved the SQL warning message SQL1061W is returned. Because indoubt transactions hold resources, such as locks and active log space, it is possible to get to a point where no changes can be made to the database because the active log space is being held up by indoubt transactions. For this reason, you should determine whether indoubt transactions remain after crash recovery, and recover all database partition servers that are required to resolve the indoubt transactions as quickly as possible.

Note: In a partitioned database server environment, the RESTART database command is run on a per-node basis. In order to ensure that the database is restarted on all nodes, use the following recommended command:

```
db2_all "db2 restart database <database_name>"
```

If one or more servers that are required to resolve an indoubt transaction cannot be recovered in time, and access is required to database partitions on other servers, you can manually resolve the indoubt transaction by making an heuristic decision. You can use the **LIST INDOUBT TRANSACTIONS** command to query, commit, and roll back the indoubt transaction on the server.

Note: The **LIST INDOUBT TRANSACTIONS** command is also used in a distributed transaction environment. To distinguish between the two types of indoubt transactions, the *originator* field in the output that is returned by the **LIST INDOUBT TRANSACTIONS** command displays one of the following:

- Db2 Enterprise Server Edition, which indicates that the transaction originated in a partitioned database environment.
- XA, which indicates that the transaction originated in a distributed environment.

Identifying the failed database partition server

When a database partition server fails, the application will typically receive one of the following SQLCODEs. The method for detecting which database manager failed depends on the SQLCODE received:

SQL0279N
This SQLCODE is received when a database partition server involved in a transaction is terminated during COMMIT processing.

SQL1224N
This SQLCODE is received when the database partition server that failed is the coordinator partition for the transaction.

SQL1229N
This SQLCODE is received when the database partition server that failed is not the coordinator partition for the transaction.

Determining which database partition server failed is a two-step process.

1. Find the partition server that detected the failure by examining the SQLCA. The SQLCA associated with SQLCODE SQL1229N contains the node number of the server that detected the error in the sixth array position of the *sqlerrd* field. (The node number that is written for the server corresponds to the node number in the `db2nodes.cfg` file.)
2. Examine the administration notification log on the server found in step one for the node number of the failed server.

Note: If multiple logical nodes are being used on a processor, the failure of one logical node can cause other logical nodes on the same processor to fail.

Recovering from the failure of a database partition server

You can recover from a failed database partition server by identifying and correcting the issue that caused the failure.

Procedure

To recover from the failure of a database partition server, perform the following steps.

1. Correct the problem that caused the failure.

2. Restart the database manager by issuing the **db2start** command from any database partition server.
3. Restart the database by issuing the **RESTART DATABASE** command on the failed database partition server or servers.

Rebuilding partitioned databases

To rebuild a partitioned database, rebuild each database partition separately. For each database partition, beginning with the catalog partition, first restore all the table spaces that you require. Any table spaces that are not restored are placed in restore pending state. Once all the database partitions are restored, you then issue the **ROLLFORWARD DATABASE** command on the catalog partition to roll all of the database partitions forward.

About this task

Note: If, at a later date, you need to restore any table spaces that were not originally included in the rebuild phase, you need to make sure that when you subsequently roll the table space forward that the rollforward utility keeps all the data across the database partitions synchronized. If a table space is missed during the original restore and rollforward operation, it might not be detected until there is an attempt to access the data and a data access error occurs. You will then need to restore and roll the missing table space forward to get it back in sync with the rest of the partitions.

To rebuild a partitioned database using table space level backup images, consider the following example.

In this example, there is a recoverable database called SAMPLE with three database partitions:

- Database partition 1 contains table spaces SYSCATSPACE, USERSP1 and USERSP2, and is the catalog partition
- Database partition 2 contains table spaces USERSP1 and USERSP3
- Database partition 3 contains table spaces USERSP1, USERSP2 and USERSP3

The following backups have been taken, where BK*xy* represents backup number *x* on partition *y*:

- BK11 is a backup of SYSCATSPACE, USERSP1 and USERSP2
- BK12 is a backup of USERSP2 and USERSP3
- BK13 is a backup of USERSP1, USERSP2 and USERSP3
- BK21 is a backup of USERSP1
- BK22 is a backup of USERSP1
- BK23 is a backup of USERSP1
- BK31 is a backup of USERSP2
- BK33 is a backup of USERSP2
- BK42 is a backup of USERSP3
- BK43 is a backup of USERSP3

The following procedure demonstrates using the **RESTORE DATABASE** and **ROLLFORWARD DATABASE** commands, issued through the CLP, to rebuild the entire database to the end of logs.

Procedure

1. On database partition 1, issue a **RESTORE DATABASE** command with the **REBUILD** option:

```
db2 restore db sample rebuild with all tablespaces in database
   taken at BK31 without prompting
```

2. On database partition 2, issue a **RESTORE DATABASE** command with the **REBUILD** option:

```
db2 restore db sample rebuild with tablespaces in database
   taken at BK42 without prompting
```

3. On database partition 3, issue a **RESTORE DATABASE** command with the **REBUILD** option:

```
db2 restore db sample rebuild with all tablespaces in database
   taken at BK43 without prompting
```

4. On the catalog partition, issue a **ROLLFORWARD DATABASE** command with the **TO END OF LOGS** option:

```
db2 rollforward db sample to end of logs
```

5. Issue a **ROLLFORWARD DATABASE** command with the **STOP** option:

```
db2 rollforward db sample stop
```

What to do next
At this point the database is connectable on all database partitions and all table spaces are in NORMAL state.

Recovering data using db2adutl

You can perform cross-node recovery using the **db2adutl** command, **logarchopt1** and **vendoropt** database configuration parameters. This recovery is demonstrated in examples from a few different Tivoli Storage Manager (TSM) environments.

Note: Tivoli Storage Manager (TSM) Version 7.1.8+ and Version 8.1.2+ introduce significant enhancements for improved security between client and server communication. Starting with 7.1.8 and 8.1.2 the trusted communications agent (TCA) is no longer available, and users must configure each Db2 instance as an authorized user with access to the node password. The examples provided below assume that a legacy TCA authentication method is in use, which require that the '-fromowner=<owner>' option is specified in the LOGARCHOPT1 and VENDOROPT database configuration options, or the TSM OPTIONS restore option, in order to perform cross-node recovery. When the authorized user method is used, the '-fromowner=<owner>' option should be omitted. For more details about using a cross-node recovery technique in the authorized user access model, see Configuration changes needed for IBM Spectrum Protect (formerly Tivoli Storage Manager) client versions starting with 7.1.8 and 8.1.2.

For the following examples, computer 1 is called `bar` and is running the AIX operating system. The user on this machine is `roecken`. The database on `bar` is called `zample`. Computer 2 is called `dps`. This computer is also running the AIX operating system, and the user is `regress9`.

Example 1: TSM server manages passwords automatically (PASSWORDACCESS option set to GENERATE)

This cross-node recovery example shows how to set up two computers so that you can recover data from one computer to another when log archives and backups are stored on a TSM server and where passwords are managed using the PASSWORDACCESS=GENERATE option.

Note: After updating the database configuration, you might have to take an offline backup of the database.

1. To enable the database for log archiving for the `bar` computer to the TSM server, update the database configuration parameter **logarchmeth1** for the `zample` database using the following command:

```
bar:/home/roecken> db2 update db cfg for zample using LOGARCHMETH1 tsm
```

   The following information is returned:

```
DB20000I  The UPDATE DATABASE CONFIGURATION command completed successfully.
```

2. Disconnect all users and applications from the database using the following command:

```
db2 force applications all
```

3. Verify that there are no applications connected to the database using the following command:

```
db2 list applications
```

You should receive a message that says that no data was returned.

Note: In a partitioned database environment, you must perform this step on all database partitions.

4. Create a backup of the database on the TSM server using the following command:

```
db2 backup db zample use tsm
```

Information similar to the following is returned:

```
Backup successful. The timestamp for this backup imagge is : 20090216151025
```

Note: In a partitioned database environment, you must perform this step on all database partitions. The order in which you perform this step on the database partitions differs depending on whether you are performing an online backup or an offline backup. For more information, see Backing up data.

5. Connect to the `zample` database using the following command:

```
db2 connect to zample
```

6. Generate new transaction logs for the database by creating a table and loading data into the TSM server using the following command:

```
bar:/home/roecken> db2 load from mr of del modified by noheader replace
   into employee copy yes use tsm
```

where in this example, the table is called `employee`, and the data is being loaded from a delimited ASCII file called `mr`. The **COPY** YES option is specified to make a copy of the data that is loaded, and the **USE TSM** option specifies that the copy of the data is stored on the TSM server.

Note: You can specify the **COPY** YES option only if the database is enabled for roll-forward recovery; that is, the **logarchmeth1** database configuration parameter must be set to USEREXIT, LOGRETAIN, DISK, or TSM.

To indicate its progress, the load utility returns a series of messages:

```
SQL3109N  The utility is beginning to load data from file "/home/roecken/mr".

SQL3500W  The utility is beginning the "LOAD" phase at time "02/16/2009
15:12:13.392633".

SQL3519W  Begin Load Consistency Point. Input record count = "0".

SQL3520W  Load Consistency Point was successful.

SQL3110N  The utility has completed processing.  "1" rows were read from the
input file.

SQL3519W  Begin Load Consistency Point. Input record count = "1".

SQL3520W  Load Consistency Point was successful.

SQL3515W  The utility has finished the "LOAD" phase at time "02/16/2009
15:12:13.445718".


Number of rows read         = 1
Number of rows skipped      = 0
Number of rows loaded       = 1
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 1
```

7. After the data has been loaded into the table, confirm that there is one backup image, one load copy image, and one log file on the TSM server by running the following query on the zample database:

```
bar:/home/roecken/sqllib/adsm> db2adutl query db zample
```

The following information is returned:

```
Retrieving FULL DATABASE BACKUP information.
    1 Time: 20090216151025  Oldest log: S0000000.LOG  Log stream: 0
    Sessions: 1

Retrieving INCREMENTAL DATABASE BACKUP information.
  No INCREMENTAL DATABASE BACKUP images found for ZAMPLE

Retrieving DELTA DATABASE BACKUP information.
  No DELTA DATABASE BACKUP images found for ZAMPLE

Retrieving TABLESPACE BACKUP information.
  No TABLESPACE BACKUP images found for ZAMPLE

Retrieving INCREMENTAL TABLESPACE BACKUP information.
  No INCREMENTAL TABLESPACE BACKUP images found for ZAMPLE

Retrieving DELTA TABLESPACE BACKUP information.
  No DELTA TABLESPACE BACKUP images found for ZAMPLE

Retrieving LOAD COPY information.
    1 Time: 20090216151213

Retrieving LOG ARCHIVE information.
   Log file: S0000000.LOG, Chain Num: 0, Log stream: 0,
   Taken at: 2009-02-16-15.10.38
```

8. To enable cross-node recovery, you must give access to the objects associated with the bar computer to another computer and account. In this example, give access to the computer dps and the user regress9 using the following command:

```
bar:/home/roecken/sqllib/adsm> db2adutl grant user regress9
   on nodename dps for db zample
```

The following information is returned:

```
Successfully added permissions for regress9 to access ZAMPLE on node dps.
```

Note: You can confirm the results of the **db2adutl** grant operation by issuing the following command to retrieve the current access list for the current node:

```
bar:/home/roecken/sqllib/adsm> db2adutl queryaccess
```

The following information is returned:

```
Node                 Username             Database Name   Type
-------------------------------------------------------------
DPS                  regress9             ZAMPLE          A
-------------------------------------------------------------
 Access Types:  B - backup images   L - logs   A - both
```

9. In this example, computer 2, dps, is not yet set up for cross-node recovery of the zample database. Verify that there is no data associated with this user and computer on the TSM server using the following command:

```
dps:/home/regress9/sqllib/adsm> db2adutl query db zample
```

The following information is returned:

```
  --- Database directory is empty ---
Warning: There are no file spaces created by Db2 on the ADSM server
Warning: No Db2 backup images found in ADSM for any alias.
```

10. Query the TSM server for a list of objects for the zample database associated with user roecken and computer bar using the following command:

```
dps:/home/regress9/sqllib/adsm> db2adutl query db zample nodename
     bar owner roecken
```

The following information is returned:

```
--- Database directory is empty ---

Query for database ZAMPLE

Retrieving FULL DATABASE BACKUP information.
    1 Time: 20090216151025  Oldest log: S0000000.LOG  Log stream: 0
    Sessions: 1

Retrieving INCREMENTAL DATABASE BACKUP information.
  No INCREMENTAL DATABASE BACKUP images found for ZAMPLE

Retrieving DELTA DATABASE BACKUP information.
  No DELTA DATABASE BACKUP images found for ZAMPLE

Retrieving TABLESPACE BACKUP information.
  No TABLESPACE BACKUP images found for ZAMPLE

Retrieving INCREMENTAL TABLESPACE BACKUP information.
  No INCREMENTAL TABLESPACE BACKUP images found for ZAMPLE

Retrieving DELTA TABLESPACE BACKUP information.
  No DELTA TABLESPACE BACKUP images found for ZAMPLE

Retrieving LOAD COPY information.
    1 Time: 20090216151213

Retrieving LOG ARCHIVE information.
   Log file: S0000000.LOG, Chain Num: 0, Log stream: 0,
   Taken at: 2009-02-16-15.10.38
```

This information matches the TSM information that was generated previously and confirms that you can restore this image onto the dps computer.

11. Restore the zample database from the TSM server to the dps computer using the following command:

```
dps:/home/regress9> db2 restore db zample use tsm options
"'-fromnode=bar -fromowner=roecken'" without prompting
```

The following information is returned:

```
DB20000I  The RESTORE DATABASE command completed successfully.
```

Note: If the zample database already existed on dps, the **OPTIONS** parameter would be omitted, and the database configuration parameter **vendoropt** would be used. This configuration parameter overrides the **OPTIONS** parameter for a backup or restore operation.

12. Perform a roll-forward operation to apply the transactions recorded in the zample database log file when a new table was created and new data loaded. In this example, the following attempt for the roll-forward operation will fail because the roll-forward utility cannot find the log files because the user and computer information is not specified:

```
dps:/home/regress9> db2 rollforward db zample to end of logs and stop
```

The command returns the following error:

```
SQL4970N  Roll-forward recovery on database "ZAMPLE" cannot reach the
specified stop point (end-of-log or point-in-time) because of missing log
file(s) on node(s) "0".
```

Force the roll-forward utility to look for log files associated with another computer using the proper **logarchopt** value. In this example, use the following command to set the **logarchopt1** database configuration parameter and search for log files associated with user `roecken` and computer `bar`:

```
dps:/home/regress9> db2 update db cfg for zample using logarchopt1
"'-fromnode=bar -fromowner=roecken'"
```

13. Enable the roll-forward utility to use the backup and load copy images by setting the **vendoropt** database configuration parameter using the following command:

```
dps:/home/regress9> db2 update db cfg for zample using VENDOROPT
"'-fromnode=bar -fromowner=roecken'"
```

14. You can finish the cross-node data recovery by applying the transactions recorded in the `zample` database log file using the following command:

```
dps:/home/regress9> db2 rollforward db zample to end of logs and stop
```

The following information is returned:

```
                                 Rollforward Status

 Input database alias                   = zample
 Number of members have returned status = 1


 Member number  Rollforward  Next log to  Log files processed        Last committed
transaction
                status       be read
 -------------  -----------  -----------  -------------------------
------------------------------
         0      not pending              S0000000.LOG-S0000000.LOG
2009-05-06-15.28.11.000000 UTC


DB20000I  The ROLLFORWARD command completed successfully.
```

The database `zample` on computer `dps` under user `regress9` has been recovered to the same point as the database on computer`bar` under user `roecken`.

Example 2: Passwords are user-managed (PASSWORDACCESS option set to PROMPT)

This cross-node recovery example shows how to set up two computers so that you can recover data from one computer to another when log archives and backups are stored on a TSM server and where passwords are managed by the users. In these environments, extra information is required, specifically the TSM nodename and password of the computer where the objects were created.

1. Update the client `dsm.sys` file by adding the following line because computer `bar` is the name of the source computer

```
NODENAME bar
```

Note: On Windows operating systems, this file is called the `dsm.opt` file. When you update this file, you must reboot your system for the changes to take effect.

2. Query the TSM server for the list of objects associated with user `roecken` and computer `bar` using the following command:

```
dps:/home/regress9/sqllib/adsm> db2adutl query db zample nodename bar
owner roecken password *******
```

The following information is returned:

```
Query for database ZAMPLE

Retrieving FULL DATABASE BACKUP information.
    1 Time: 20090216151025  Oldest log: S0000000.LOG  Log stream: 0
    Sessions: 1

Retrieving INCREMENTAL DATABASE BACKUP information.
  No INCREMENTAL DATABASE BACKUP images found for ZAMPLE

Retrieving DELTA DATABASE BACKUP information.
  No DELTA DATABASE BACKUP images found for ZAMPLE

Retrieving TABLESPACE BACKUP information.
  No TABLESPACE BACKUP images found for ZAMPLE

Retrieving INCREMENTAL TABLESPACE BACKUP information.
  No INCREMENTAL TABLESPACE BACKUP images found for ZAMPLE

Retrieving DELTA TABLESPACE BACKUP information.
  No DELTA TABLESPACE BACKUP images found for ZAMPLE

Retrieving LOAD COPY information.
    1 Time: 20090216151213

Retrieving LOG ARCHIVE information.
   Log file: S0000000.LOG, Chain Num: 0, Log stream: 0,
   Taken at: 2009-02-16-15.10.38
```

3. If the `zample` database does not exist on computer dps, perform the following steps:
   a. Create an empty `zample` database using the following command:

      ```
      dps:/home/regress9> db2 create db zample
      ```

   b. Update the database configuration parameter **tsm_nodename** using the following command:

      ```
      dps:/home/regress9> db2 update db cfg for zample using tsm_nodename bar
      ```

   c. Update the database configuration parameter **tsm_password** using the following command:

      ```
      dps:/home/regress9> db2 update db cfg for zample using
      tsm_password ********
      ```

4. Attempt to restore the `zample` database using the following command:

   ```
   dps:/home/regress9> db2 restore db zample use tsm options
   "'-fromnode=bar -fromowner=roecken'" without prompting
   ```

   The restore operation completes successfully, but a warning is issued:

   ```
   SQL2540W  Restore is successful, however a warning "2523" was
   encountered during Database Restore while processing in No
   Interrupt mode.
   ```

5. Perform a roll-forward operation using the following command:

   ```
   dps:/home/regress9> db2 rollforward db zample to end of logs and stop
   ```

   In this example, because the restore operation replaced the database configuration file, the roll-forward utility cannot find the correct log files and the following error message is returned:

   ```
   SQL1268N  Roll-forward recovery stopped due to error "-2112880618"
   while retrieving log file "S0000000.LOG" for database "ZAMPLE" on node "0".
   ```

   Reset the following TSM database configuration values to the correct values:

a. Set the **tsm_nodename** configuration parameter using the following command:

```
dps:/home/regress9> db2 update db cfg for zample using tsm_nodename bar
```

b. Set the **tsm_password** database configuration parameter using the following command:

```
dps:/home/regress9> db2 update db cfg for zample using tsm_password *******
```

c. Set the **logarchopt1** database configuration parameter so that the roll-forward utility can find the correct log files using the following command:

```
dps:/home/regress9> db2 update db cfg for zample using logarchopt1
"'-fromnode=bar -fromowner=roecken'"
```

d. Set the **vendoropt** database configuration parameter so that the load recovery file can also be used during the roll-forward operation using the following command:

```
dps:/home/regress9> db2 update db cfg for zample using VENDOROPT
"'-fromnode=bar -fromowner=roecken'"
```

6. You can finish the cross-node recovery by performing the roll-forward operation using the following command:

```
dps:/home/regress9> db2 rollforward db zample to end of logs and stop
```

The following information is returned:

```
                                  Rollforward Status

 Input database alias                   = zample
 Number of members have returned status = 1


 Member number  Rollforward  Next log to  Log files processed        Last committed
transaction
                status       be read
 -------------  -----------  -----------  -------------------------
------------------------------
         0      not pending               S0000000.LOG-S0000000.LOG
2009-05-06-15.28.11.000000 UTC


DB20000I  The ROLLFORWARD command completed successfully.
```

The database `zample` on computer `dps` under user `regress9` has been recovered to the same point as the database on computer`bar` under user `roecken`

Example 3: TSM server is configured to use client proxy nodes

This cross-node recovery example shows how to set up two computers as proxy nodes so that you can recover data from one computer to another when log archives and backups are stored on a TSM server and where passwords are managed using the PASSWORDACCESS=GENERATE option.

Note: After updating the database configuration, you might have to take an offline backup of the database.

In this example, the computers `bar` and `dps` are registered under the proxy name of `clusternode`. The computers are already setup as proxy nodes.

1. Register the computers `bar` and `dps` on the TSM server as proxy nodes using the following commands:

```
REGISTER NODE clusternode mypassword
GRANT PROXYNODE TARGET=clusternode AGENT=bar,dps
```

2. To enable the database for log archiving to the TSM server, update the database configuration parameter **logarchmeth1** for the zample database using the following command:

```
bar:/home/roecken> db2 update db cfg for zample using
   LOGARCHMETH1 tsm logarchopt1 "'-asnodename=clusternode'"
```

   The following information is returned:

```
DB20000I  The UPDATE DATABASE CONFIGURATION command completed successfully.
```

3. Disconnect all users and applications from the database using the following command:

```
db2 force applications all
```

4. Verify that there are no applications connected to the database using the following command:

```
db2 list applications
```

   You should receive a message that says that no data was returned.

   Note: In a partitioned database environment, you must perform this step on all database partitions.

5. Create a backup of the database on the TSM server using the following command:

```
db2 backup db zample use tsm options "'-asnodename=clusternode'"
```

   Information similar to the following is returned:

```
Backup successful. The timestamp for this backup image is : 20090216151025
```

   Instead of specifying the **-asnodename** option on the **BACKUP DATABASE** command, you can update the **vendoropt** database configuration parameter instead.

   Note: In a partitioned database environment, you must perform this step on all database partitions. The order in which you perform this step on the database partitions differs depending on whether you are performing an online backup or an offline backup. For more information, see Backing up data.

6. Connect to the zample database using the following command:

```
db2 connect to zample
```

7. Generate new transaction logs for the database by creating a table and loading data into the TSM server using the following command:

```
bar:/home/roecken> db2 load from mr of del modified by noheader
    replace into employee copy yes use tsm
```

   where in this example, the table is called employee, and the data is being loaded from a delimited ASCII file called mr. The **COPY** YES option is specified to make a copy of the data that is loaded, and the **USE TSM** option specifies that the copy of the data is stored on the TSM server.

   Note: You can specify the **COPY** YES option only if the database is enabled for roll-forward recovery; that is, the **logarchmeth1** database configuration parameter must be set to USEREXIT, LOGRETAIN, DISK, or TSM.

   To indicate its progress, the load utility returns a series of messages:

```
SQL3109N  The utility is beginning to load data from file "/home/roecken/mr".

SQL3500W  The utility is beginning the "LOAD" phase at time "02/16/2009
15:12:13.392633".

SQL3519W  Begin Load Consistency Point. Input record count = "0".

SQL3520W  Load Consistency Point was successful.
```

```
SQL3110N  The utility has completed processing.  "1" rows were read from the
input file.

SQL3519W  Begin Load Consistency Point. Input record count = "1".

SQL3520W  Load Consistency Point was successful.

SQL3515W  The utility has finished the "LOAD" phase at time "02/16/2009
15:12:13.445718".


Number of rows read         = 1
Number of rows skipped      = 0
Number of rows loaded       = 1
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 1
```

8. After the data has been loaded into the table, confirm that there is one backup image, one load copy image, and one log file on the TSM server by running the following query on the `zample` database:

```
bar:/home/roecken/sqllib/adsm> db2adutl query db zample
   options "-asnodename=clusternode"
```

   The following information is returned:

```
Retrieving FULL DATABASE BACKUP information.
    1 Time: 20090216151025  Oldest log: S0000000.LOG  Log stream: 0
    Sessions: 1

Retrieving INCREMENTAL DATABASE BACKUP information.
  No INCREMENTAL DATABASE BACKUP images found for ZAMPLE

Retrieving DELTA DATABASE BACKUP information.
  No DELTA DATABASE BACKUP images found for ZAMPLE

Retrieving TABLESPACE BACKUP information.
  No TABLESPACE BACKUP images found for ZAMPLE

Retrieving INCREMENTAL TABLESPACE BACKUP information.
  No INCREMENTAL TABLESPACE BACKUP images found for ZAMPLE

Retrieving DELTA TABLESPACE BACKUP information.
  No DELTA TABLESPACE BACKUP images found for ZAMPLE

Retrieving LOAD COPY information.
    1 Time: 20090216151213

Retrieving LOG ARCHIVE information.
   Log file: S0000000.LOG, Chain Num: 0, Log stream: 0,
   Taken at: 2009-02-16-15.10.38
```

9. In this example, computer 2, `dps`, is not yet set up for cross-node recovery of the `zample` database. Verify that there is no data associated with this user and computer using the following command:

```
dps:/home/regress9/sqllib/adsm> db2adutl query db zample
```

   The following information is returned:

```
  --- Database directory is empty ---
Warning: There are no file spaces created by Db2 on the ADSM server
Warning: No Db2 backup images found in ADSM for any alias.
```

10. Query the TSM server for a list of objects for the `zample` database associated with the proxy node `clusternode` using the following command:

```
dps:/home/regress9/sqllib/adsm> db2adutl query db zample
   options="-asnodename=clusternode"
```

The following information is returned:

```
--- Database directory is empty ---

Query for database ZAMPLE

Retrieving FULL DATABASE BACKUP information.
    1 Time: 20090216151025  Oldest log: S0000000.LOG  Log stream: 0
    Sessions: 1

Retrieving INCREMENTAL DATABASE BACKUP information.
  No INCREMENTAL DATABASE BACKUP images found for ZAMPLE

Retrieving DELTA DATABASE BACKUP information.
  No DELTA DATABASE BACKUP images found for ZAMPLE

Retrieving TABLESPACE BACKUP information.
  No TABLESPACE BACKUP images found for ZAMPLE

Retrieving INCREMENTAL TABLESPACE BACKUP information.
  No INCREMENTAL TABLESPACE BACKUP images found for ZAMPLE

Retrieving DELTA TABLESPACE BACKUP information.
  No DELTA TABLESPACE BACKUP images found for ZAMPLE

Retrieving LOAD COPY information.
    1 Time: 20090216151213

Retrieving LOG ARCHIVE information.
   Log file: S0000000.LOG, Chain Num: 0, Log stream: 0,
   Taken at: 2009-02-16-15.10.38
```

This information matches the TSM information that was generated previously and confirms that you can restore this image onto the dps computer.

11. Restore the zample database from the TSM server to the dps computer using the following command:

```
dps:/home/regress9> db2 restore db zample use tsm options
   "'-asnodename=clusternode'" without prompting
```

The following information is returned:

```
DB20000I  The RESTORE DATABASE command completed successfully.
```

Note: If the zample database already existed on dps, the **OPTIONS** parameter would be omitted, and the database configuration parameter **vendoropt** would be used. This configuration parameter overrides the **OPTIONS** parameter for a backup or restore operation.

12. Perform a roll-forward operation to apply the transactions recorded in the zample database log file when a new table was created and new data loaded. In this example, the following attempt for the roll-forward operation will fail because the roll-forward utility cannot find the log files because the user and computer information is not specified:

```
dps:/home/regress9> db2 rollforward db zample to end of logs and stop
```

The command returns the following error:

```
SQL4970N  Roll-forward recovery on database "ZAMPLE" cannot reach the
specified stop point (end-of-log or point-in-time) because of missing log
file(s) on node(s) "0".
```

Force the roll-forward utility to look for log files on another computer using the proper **logarchopt** value. In this example, use the following command to set the **logarchopt1** database configuration parameter and search for log files associated with user roecken and computer bar:

```
dps:/home/regress9> db2 update db cfg for zample using logarchopt1
   "'-asnodename=clusternode'"
```

13. Enable the roll-forward utility to use the backup and load copy images by setting the **vendoropt** database configuration parameter using the following command:

```
dps:/home/regress9> db2 update db cfg for zample using VENDOROPT
   "'-asnodename=clusternode'"
```

14. You can finish the cross-node data recovery by applying the transactions recorded in the zample database log file using the following command:

```
dps:/home/regress9> db2 rollforward db zample to end of logs and stop
```

The following information is returned:

```
                                  Rollforward Status

 Input database alias                   = zample
 Number of members have returned status = 1


 Member number  Rollforward  Next log to  Log files processed        Last committed
transaction
                status       be read
 -------------  -----------  -----------  -------------------------
------------------------------
         0      not pending               S0000000.LOG-S0000000.LOG
2009-05-06-15.28.11.000000 UTC


DB20000I  The ROLLFORWARD command completed successfully.
```

The database zample on computer dps under user regress9 has been recovered to the same point as the database on computer bar under user roecken.

Example 4: TSM server is configured to use client proxy nodes in a Db2 pureScale environment

This example shows how to set up two members as proxy nodes so that you can recover data from one member to the other when log archives and backups are stored on a TSM server and where passwords are managed using the PASSWORDACCESS=GENERATE option.

Note: After updating the database configuration, you might have to take an offline backup of the database.

In this example, the members member1 and member2 are registered under the proxy name of clusternode. In Db2 pureScale environments, you can perform backup or data recovery operations from any member. In this example, data will be recovered from member2

1. Register the members member1 and member2 on the TSM server as proxy nodes using the following commands:

```
REGISTER NODE clusternode mypassword
GRANT PROXYNODE TARGET=clusternode AGENT=member1,member2
```

2. To enable the database for log archiving to the TSM server, update the database configuration parameter **logarchmeth1** for the zample database using the following command:

```
member1:/home/roecken> db2 update db cfg for zample using
   LOGARCHMETH1 tsm logarchopt1 "'-asnodename=clusternode'"
```

Note: In Db2 pureScale environments, you can set the global **logarchmeth1** database configuration parameters once from any member.

The following information is returned:

```
DB20000I  The UPDATE DATABASE CONFIGURATION command completed successfully.
```

3. Disconnect all users and applications from the database using the following command:

```
db2 force applications all
```

4. Verify that there are no applications connected to the database using the following command:

```
db2 list applications global
```

You should receive a message that says that no data was returned.

5. Create a backup of the database on the TSM server using the following command:

```
db2 backup db zample use tsm options '-asnodename=clusternode'
```

Information similar to the following is returned:

```
Backup successful. The timestamp for this backup image is : 20090216151025
```

Instead of specifying the **-asnodename** option on the **BACKUP DATABASE** command, you can update the **vendoropt** database configuration parameter instead.

Note: In Db2 pureScale environments, you can run this command from any member to back up all data for the database.

6. Connect to the zample database using the following command:

```
db2 connect to zample
```

7. Generate new transaction logs for the database by creating a table and loading data into the TSM server using the following command:

```
member1:/home/roecken> db2 load from mr of del modified by noheader replace
      into employee copy yes use tsmwhere
```

where in this example, the table is called employee, and the data is being loaded from a delimited ASCII file called mr. The **COPY** YES option is specified to make a copy of the data that is loaded, and the **USE TSM** option specifies that the copy of the data is stored on the TSM server.

Note: You can specify the **COPY** YES option only if the database is enabled for roll-forward recovery; that is, the **logarchmeth1** database configuration parameter must be set to USEREXIT, LOGRETAIN, DISK, or TSM.

To indicate its progress, the load utility returns a series of messages:

```
SQL3109N  The utility is beginning to load data from file "/home/roecken/mr".

SQL3500W  The utility is beginning the "LOAD" phase at time "02/16/2009
15:12:13.392633".

SQL3519W  Begin Load Consistency Point. Input record count = "0".

SQL3520W  Load Consistency Point was successful.

SQL3110N  The utility has completed processing.  "1" rows were read from the
input file.

SQL3519W  Begin Load Consistency Point. Input record count = "1".

SQL3520W  Load Consistency Point was successful.

SQL3515W  The utility has finished the "LOAD" phase at time "02/16/2009
15:12:13.445718".
```

```
   Number of rows read         = 1
   Number of rows skipped      = 0
   Number of rows loaded       = 1
   Number of rows rejected     = 0
   Number of rows deleted      = 0
   Number of rows committed    = 1
```

8. After the data has been loaded into the table, confirm that there is one backup image, one load copy image, and one log file on the TSM server by running the following query on the `zample` database:

```
   member1:/home/roecken/sqllib/adsm> db2adutl query db zample
      options "-asnodename=clusternode"
```

The following information is returned:

```
   Retrieving FULL DATABASE BACKUP information.
       1 Time: 20090216151025  Oldest log: S0000000.LOG  Log stream: 0
       Sessions: 1

   Retrieving INCREMENTAL DATABASE BACKUP information.
     No INCREMENTAL DATABASE BACKUP images found for ZAMPLE

   Retrieving DELTA DATABASE BACKUP information.
     No DELTA DATABASE BACKUP images found for ZAMPLE

   Retrieving TABLESPACE BACKUP information.
     No TABLESPACE BACKUP images found for ZAMPLE

   Retrieving INCREMENTAL TABLESPACE BACKUP information.
     No INCREMENTAL TABLESPACE BACKUP images found for ZAMPLE

   Retrieving DELTA TABLESPACE BACKUP information.
     No DELTA TABLESPACE BACKUP images found for ZAMPLE

   Retrieving LOAD COPY information.
       1 Time: 20090216151213

Retrieving LOG ARCHIVE information.

   Log file: S0000000.LOG, Chain Num: 1, Log stream: 1, Taken at:
2009-02-16-13.01.10

   Log file: S0000000.LOG, Chain Num: 1, Log stream: 0, Taken at:
2009-02-16-13.01.11

   Log file: S0000000.LOG, Chain Num: 1, Log stream: 2, Taken at:
2009-02-16-13.01.19

   Log file: S0000001.LOG, Chain Num: 1, Log stream: 0, Taken at:
2009-02-16-13.02.49

   Log file: S0000001.LOG, Chain Num: 1, Log stream: 1, Taken at:
2009-02-16-13.02.49

   Log file: S0000001.LOG, Chain Num: 1, Log stream: 2, Taken at:
2009-02-16-13.02.49

   Log file: S0000002.LOG, Chain Num: 1, Log stream: 1, Taken at:
2009-02-16-13.03.15

   Log file: S0000002.LOG, Chain Num: 1, Log stream: 2, Taken at:
2009-02-16-13.03.15

   Log file: S0000002.LOG, Chain Num: 1, Log stream: 0, Taken at:
2009-02-16-13.03.16
```

9. Query the TSM server for a list of objects for the zample database associated with the proxy node clusternode using the following command:

```
member2:/home/regress9/sqllib/adsm> db2adutl query db zample
   options="-asnodename=clusternode"
```

The following information is returned:

```
   --- Database directory is empty ---

   Query for database ZAMPLE

   Retrieving FULL DATABASE BACKUP information.
       1 Time: 20090216151025  Oldest log: S0000000.LOG  Log stream: 0
       Sessions: 1

   Retrieving INCREMENTAL DATABASE BACKUP information.
     No INCREMENTAL DATABASE BACKUP images found for ZAMPLE

   Retrieving DELTA DATABASE BACKUP information.
     No DELTA DATABASE BACKUP images found for ZAMPLE

   Retrieving TABLESPACE BACKUP information.
     No TABLESPACE BACKUP images found for ZAMPLE

   Retrieving INCREMENTAL TABLESPACE BACKUP information.
     No INCREMENTAL TABLESPACE BACKUP images found for ZAMPLE

   Retrieving DELTA TABLESPACE BACKUP information.
     No DELTA TABLESPACE BACKUP images found for ZAMPLE

   Retrieving LOAD COPY information.
       1 Time: 20090216151213

Retrieving LOG ARCHIVE information.

   Log file: S0000000.LOG, Chain Num: 1, Log stream: 1, Taken at:
2009-02-16-13.01.10

   Log file: S0000000.LOG, Chain Num: 1, Log stream: 0, Taken at:
2009-02-16-13.01.11

   Log file: S0000000.LOG, Chain Num: 1, Log stream: 2, Taken at:
2009-02-16-13.01.19

   Log file: S0000001.LOG, Chain Num: 1, Log stream: 0, Taken at:
2009-02-16-13.02.49

   Log file: S0000001.LOG, Chain Num: 1, Log stream: 1, Taken at:
2009-02-16-13.02.49

   Log file: S0000001.LOG, Chain Num: 1, Log stream: 2, Taken at:
2009-02-16-13.02.49

   Log file: S0000002.LOG, Chain Num: 1, Log stream: 1, Taken at:
2009-02-16-13.03.15

   Log file: S0000002.LOG, Chain Num: 1, Log stream: 2, Taken at:
2009-02-16-13.03.15

   Log file: S0000002.LOG, Chain Num: 1, Log stream: 0, Taken at:
2009-02-16-13.03.16
```

This information matches the TSM information that was generated previously and confirms that you can restore this image onto the member2 member.

10. Restore the zample database on the TSM server from the member2 member using the following command:

```
member2:/home/regress9> db2 restore db zample use tsm options
   '-asnodename=clusternode' without prompting
```

The following information is returned:

```
DB20000I  The RESTORE DATABASE command completed successfully.
```

Note: If the zample database already existed on member2, the **OPTIONS** parameter would be omitted, and the database configuration parameter **vendoropt** would be used. This configuration parameter overrides the **OPTIONS** parameter for a backup or restore operation.

11. Enable the roll-forward utility to use the backup and load copy images by setting the **vendoropt** database configuration parameter using the following command:

```
member2:/home/regress9> db2 update db cfg for zample using VENDOROPT
   "'-asnodename=clusternode'"
```

Note: In Db2 pureScale environments, you can set the global **vendoropt** database configuration parameters once from any member.

12. You can finish the cross-member data recovery by applying the transactions recorded in the zample database log file using the following command:

```
member2:/home/regress9> db2 rollforward db zample to end of logs and stop
```

The following information is returned:

```
                                  Rollforward Status

 Input database alias                   = zample
 Number of members have returned status = 3


 Member number  Rollforward  Next log to  Log files processed        Last committed
transaction
                status       be read
 -------------  -----------  -----------  -------------------------
------------------------------
             0  not pending               S0000001.LOG-S0000012.LOG
2009-05-06-15.28.11.000000 UTC
             1  not pending               S0000001.LOG-S0000012.LOG
2009-05-06-15.28.11.000000 UTC
             2  not pending               S0000001.LOG-S0000012.LOG
2009-05-06-15.28.11.000000 UTC

DB20000I  The ROLLFORWARD command completed successfully.
```

The database zample on member member2 under user regress9 has been recovered to the same point as the database on member member1 under user roecken.

Synchronizing clocks in a partitioned database environment

You should maintain relatively synchronized system clocks across the database partition servers to ensure smooth database operations and unlimited forward recoverability. Time differences among the database partition servers, plus any potential operational and communications delays for a transaction should be less than the value specified for the *max_time_diff* (maximum time difference among nodes) database manager configuration parameter.

To ensure that the log record time stamps reflect the sequence of transactions in a partitioned database environment, Db2 uses the system clock and the virtual timestamp stored in the SQLOGCTL.LFH file on each machine as the basis for the time stamps in the log records. If, however, the system clock is set ahead, the log clock is automatically set ahead with it. Although the system clock can be set back, the clock for the logs cannot, and remains at the *same* advanced time until the system clock matches this

time. The clocks are then in synchrony. The implication of this is that a short term system clock error on a database node can have a long lasting effect on the time stamps of database logs.

For example, assume that the system clock on database partition server A is mistakenly set to November 7, 2005 when the year is 2003, and assume that the mistake is corrected *after* an update transaction is committed in the database partition at that database partition server. If the database is in continual use, and is regularly updated over time, any point between November 7, 2003 and November 7, 2005 is virtually unreachable through rollforward recovery. When the COMMIT on database partition server A completes, the time stamp in the database log is set to 2005, and the log clock remains at November 7, 2005 until the system clock matches this time. If you attempt to roll forward to a point in time within this time frame, the operation will stop at the first time stamp that is beyond the specified stop point, which is November 7, 2003.

Although Db2 cannot control updates to the system clock, the *max_time_diff* database manager configuration parameter reduces the chances of this type of problem occurring:

- The configurable values for this parameter range from 1 minute to 24 hours.
- When the first connection request is made to a non-catalog partition, the database partition server sends its time to the catalog partition for the database. The catalog partition then checks that the time on the database partition requesting the connection, and its own time are within the range specified by the *max_time_diff* parameter. If this range is exceeded, the connection is refused.
- An update transaction that involves more than two database partition servers in the database must verify that the clocks on the participating database partition servers are in synchrony before the update can be committed. If two or more database partition servers have a time difference that exceeds the limit allowed by *max_time_diff*, the transaction is rolled back to prevent the incorrect time from being propagated to other database partition servers.

Troubleshooting

Troubleshooting partitioned database environments

Issuing commands in partitioned database environments

In a partitioned database environment, you might want to issue commands to be run on computers in the instance, or on database partition servers. You can do so using the **rah** command or the **db2_all** command. The **rah** command allows you to issue commands that you want to run at computers in the instance.

If you want the commands to run at database partition servers in the instance, you run the **db2_all** command. This section provides an overview of these commands. The information that follows applies to partitioned database environments only.

On Windows, to run the **rah** command or the **db2_all** command, you must be logged on with a user account that is a member of the Administrators group.

On Linux and UNIX operating systems, your login shell can be a Korn shell or any other shell; however, there are differences in the way the different shells handle commands containing special characters.

Also, on Linux and UNIX operating systems, **rah** uses the remote shell program specified by the **DB2RSHCMD** registry variable. You can select between the two remote shell programs: ssh (for additional security), or rsh. If **DB2RSHCMD** is not set, rsh is used. The ssh remote shell program is used to prevent the transmission of passwords in clear text in UNIX operating system environments.

If a command runs on one database partition server and you want it to run on all of them, use **db2_all**. The exception is the **db2trc** command, which runs on all the logical database partition servers on a computer. If you want to run **db2trc** on all logical database partition servers on all computers, use **rah**.

Note: The **db2_all** command does not support commands that require interactive user input.

Chapter 4. Performance issues

Performance issues in database design

Performance enhancing features

Table partitioning and multidimensional clustering tables

In a table that is both multidimensional clustered and data partitioned, columns can be used both in the table partitioning range-partition-spec and in the multidimensional clustering (MDC) key. A table that is both multidimensional clustered and partitioned can achieve a finer granularity of data partition and block elimination than could be achieved by either functionality alone.

There are also many applications where it is useful to specify different columns for the MDC key than those on which the table is partitioned. It should be noted that table partitioning is multicolumn, while MDC is multi-dimension.

Characteristics of a mainstream Db2 data warehouse

The following recommendations were focused on typical, mainstream warehouses that were new for Db2 V9.1. The following characteristics are assumed:

- The database runs on multiple machines or multiple AIX logical partitions.
- Partitioned database environments are used (tables are created using the DISTRIBUTE BY HASH clause).
- There are four to fifty data partitions.
- The table for which MDC and table partitioning is being considered is a major fact table.
- The table has 100,000,000 to 100,000,000,000 rows.
- New data is loaded at various time frames: nightly, weekly, monthly.
- Daily ingest volume is 10 thousand to 10 million records.
- Data volumes vary: The biggest month is 5X the size of the smallest month. Likewise, the biggest dimensions (product line, region) have a 5X size range.
- 1 to 5 years of detailed data is retained.
- Expired data is rolled out monthly or quarterly.
- Tables use a wide range of query types. However, the workload is mostly analytical queries with the following characteristics, relative to OLTP workloads:
  - larger results sets with up to 2 million rows
  - most or all queries are hitting views, not base tables
- SQL clauses selecting data by ranges (BETWEEN clause), items in lists, and so on.

Characteristics of a mainstream Db2 V9.1 data warehouse fact table

A typical warehouse fact table, might use the following design:

- Create data partitions on the Month column.
- Define a data partition for each period you roll-out, for example, 1 month, 3 months.
- Create MDC dimensions on Day and on 1 to 4 additional dimensions. Typical dimensions are: product line and region.
- All data partitions and MDC clusters are spread across all database partitions.

MDC and table partitioning provide overlapping sets of benefits. The following table lists potential needs in your organization and identifies a recommended organization scheme based on the characteristics identified previously.

Table 15. Using table partitioning with MDC tables

Issue	Recommended scheme	Recommendation
Data availability during roll-out	Table partitioning	Use the DETACH PARTITION clause to roll out large amounts of data with minimal disruption.
Query performance	Table partitioning and MDC	MDC is best for querying multiple dimensions. Table partitioning helps through data partition elimination.
Minimal reorganization	MDC	MDC maintains clustering, which reduces the need to reorganize.
Rollout a month or more of data during a traditional offline window	Table partitioning	Data partitioning addresses this need fully. MDC adds nothing and would be less suitable on its own.
Rollout a month or more of data during a micro-offline window (less than 1 minute)	Table partitioning	Data partitioning addresses this need fully. MDC adds nothing and would be less suitable on its own.
Rollout a month or more of data while keeping the table fully available for business users submitting queries without any loss of service.	MDC	MDC only addresses this need somewhat. Table partitioning would not be suitable due to the short period the table goes offline.
Load data daily (**LOAD** or **INGEST** command)	Table partitioning and MDC	MDC provides most of the benefit here. Table partitioning provides incremental benefits.
Load data "continually" (**LOAD** command with ALLOW READ ACCESS or **INGEST** command)	Table partitioning and MDC	MDC provides most of the benefit here. Table partitioning provides incremental benefits.
Query execution performance for "traditional BI" queries	Table partitioning and MDC	MDC is especially good for querying cubes/multiple dimensions. Table partitioning helps via partition elimination.
Minimize reorganization pain, by avoiding the need for reorganization or reducing the pain associated with performing the task	MDC	MDC maintains clustering which reduces the need to reorg. If MDC is used, data partitioning does not provide incremental benefits. However if MDC is not used, table partitioning helps reduce the need for reorg by maintaining some course grain clustering at the partition level.

Example 1:

Consider a table with key columns YearAndMonth and Province. A reasonable approach to planning this table might be to partition by date with 2 months per data partition. In addition, you might also organize

by Province, so that all rows for a particular province within any two month date range are clustered together, as shown in Figure 44 on page 241.

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

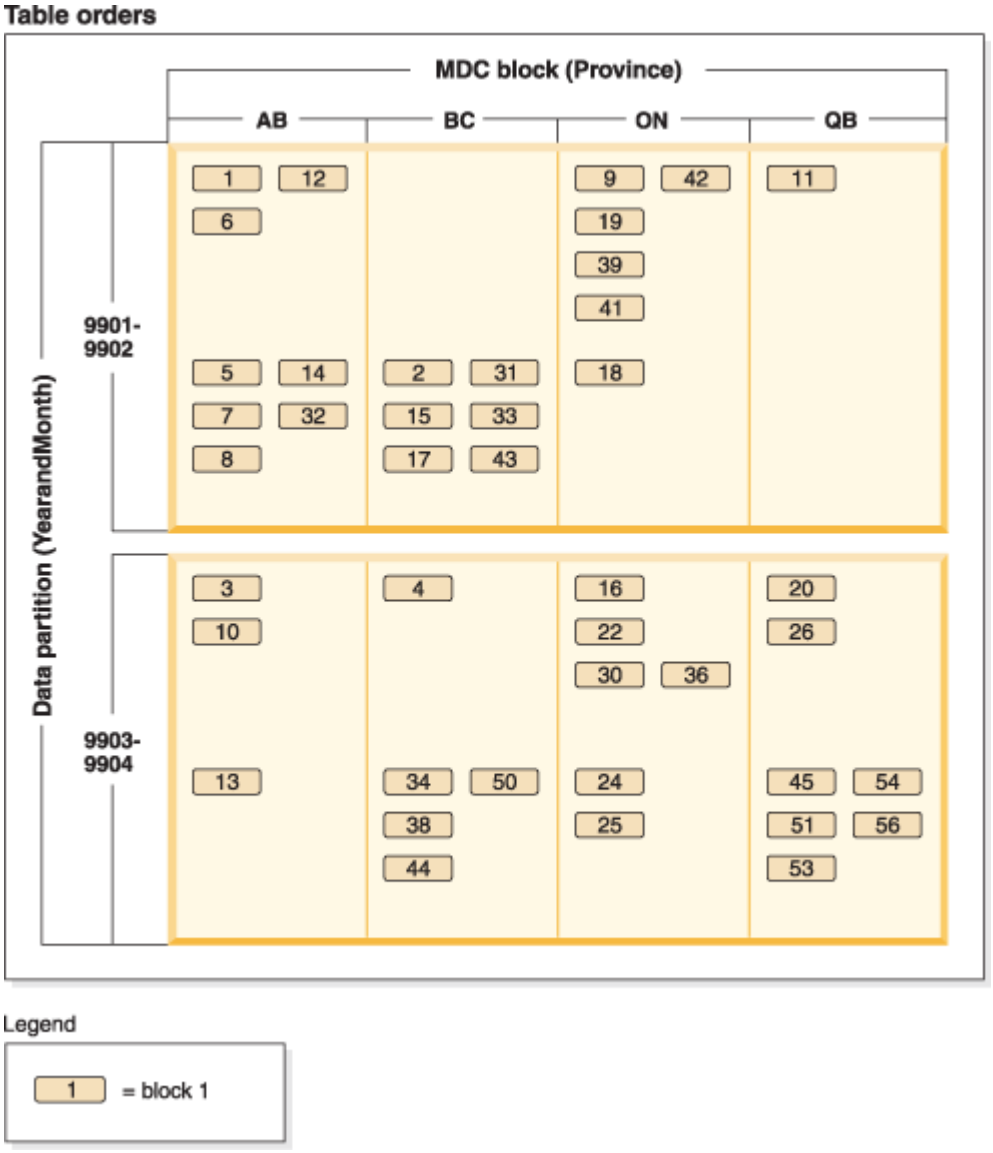


Figure 44. A table partitioned by YearAndMonth and organized by Province

Example 2:

Finer granularity can be achieved by adding YearAndMonth to the ORGANIZE BY DIMENSIONS clause, as shown in Figure 45 on page 242.

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

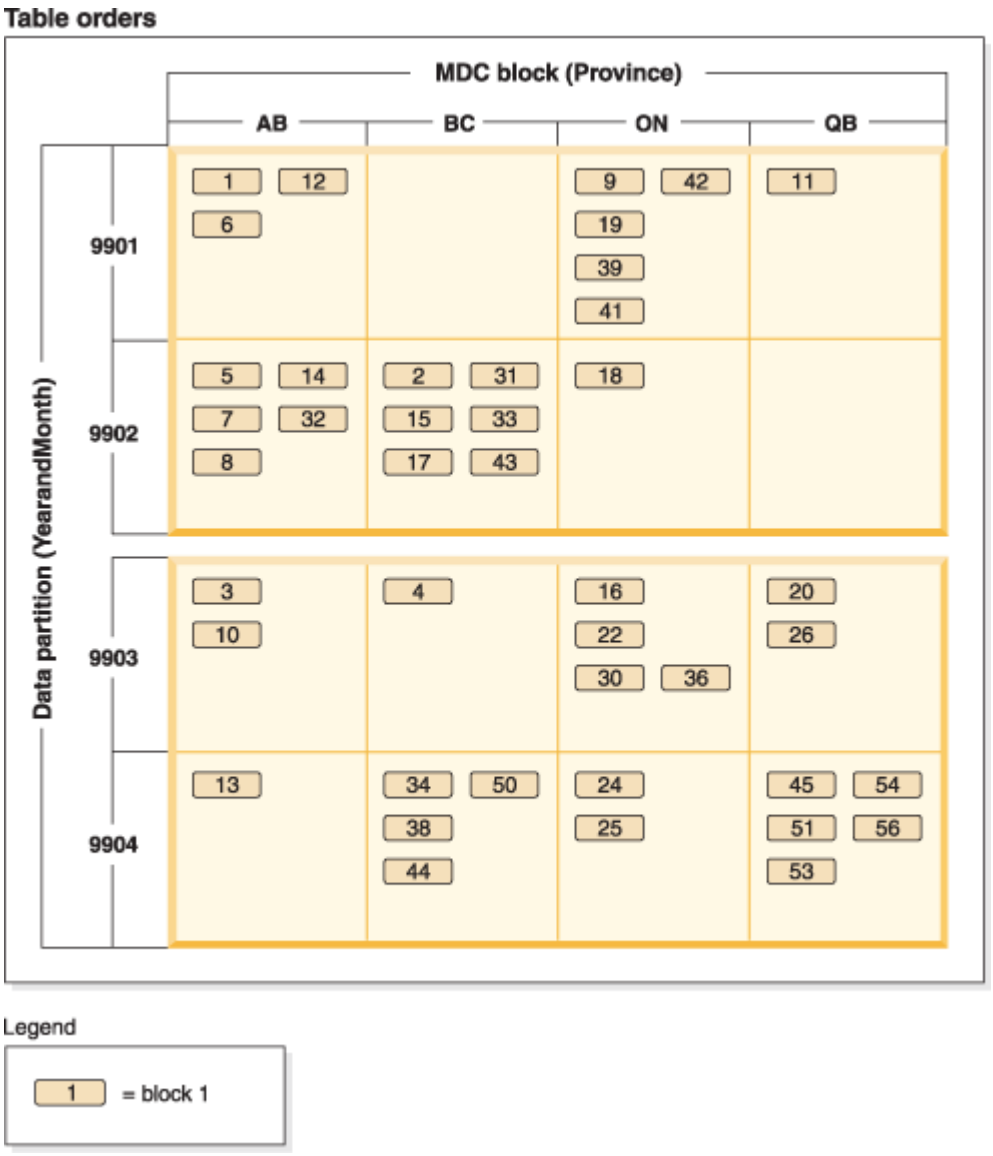


Figure 45. A table partitioned by YearAndMonth and organized by Province and YearAndMonth

In cases where the partitioning is such that there is only a single value in each range, nothing is gained by including the table partitioning column in the MDC key.

Considerations

- Compared to a basic table, both MDC tables and partitioned tables require more storage. These storage needs are additive but are considered reasonable given the benefits.
- If you choose not to combine table partitioning and MDC functionality in your partitioned database environment, table partitioning is best in cases where you can confidently predict the data distribution, which is generally the case for the types of systems discussed here. Otherwise, MDC should be considered.
- For a data-partitioned MDC table created with Db2 Version 9.7 Fix Pack 1 or later releases, the MDC block indexes on the table are partitioned. For a data-partitioned MDC table created with Db2 V9.7 or earlier releases, the MDC block indexes on the table are nonpartitioned.

Optimization strategies for partitioned tables

Data partition elimination refers to the database server's ability to determine, based on query predicates, that only a subset of the data partitions in a table need to be accessed to answer a query. Data partition elimination is particularly useful when running decision support queries against a partitioned table.

A partitioned table uses a data organization scheme in which table data is divided across multiple storage objects, called data partitions or ranges, according to values in one or more table partitioning key columns of the table. Data from a table is partitioned into multiple storage objects based on specifications provided in the PARTITION BY clause of the CREATE TABLE statement. These storage objects can be in different table spaces, in the same table space, or a combination of both.

The following example demonstrates the performance benefits of data partition elimination.

```
create table custlist(
  subsdate date, province char(2), accountid int)
  partition by range(subsdate) (
    starting from '1/1/1990' in ts1,
    starting from '1/1/1991' in ts1,
    starting from '1/1/1992' in ts1,
    starting from '1/1/1993' in ts2,
    starting from '1/1/1994' in ts2,
    starting from '1/1/1995' in ts2,
    starting from '1/1/1996' in ts3,
    starting from '1/1/1997' in ts3,
    starting from '1/1/1998' in ts3,
    starting from '1/1/1999' in ts4,
    starting from '1/1/2000' in ts4,
    starting from '1/1/2001'
    ending '12/31/2001' in ts4)
```

Assume that you are only interested in customer information for the year 2000.

```
select * from custlist
  where subsdate between '1/1/2000' and '12/31/2000'
```

As Figure 46 on page 243 shows, the database server determines that only one data partition in table space TS4 must be accessed to resolve this query.

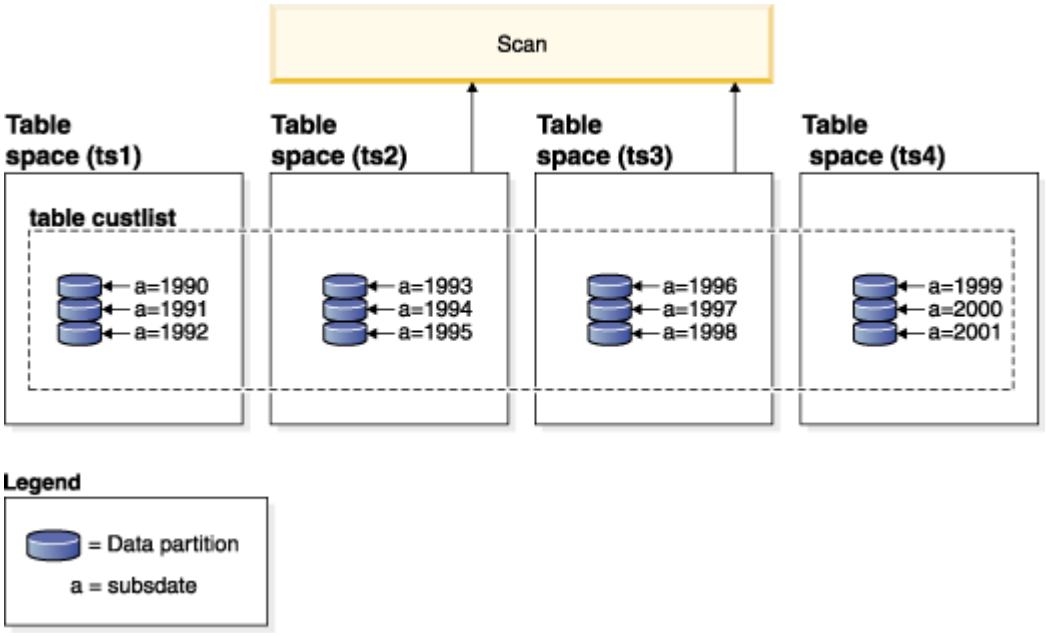


Figure 46. The performance benefits of data partition elimination

Another example of data partition elimination is based on the following scheme:

```
create table multi (
  sale_date date, region char(2))
  partition by (sale_date) (
    starting '01/01/2005'
```

```
       ending '12/31/2005'
       every 1 month)

   create index sx on multi(sale_date)

   create index rx on multi(region)
```

Assume that you issue the following query:

```
   select * from multi
     where sale_date between '6/1/2005'
       and '7/31/2005' and region = 'NW'
```

Without table partitioning, one likely plan is index ANDing. Index ANDing performs the following tasks:

- Reads all relevant index entries from each index
- Saves both sets of row identifiers (RIDs)
- Matches RIDs to determine which occur in both indexes
- Uses the RIDs to fetch the rows

As Figure 47 on page 244 demonstrates, with table partitioning, the index is read to find matches for both REGION and SALE_DATE, resulting in the fast retrieval of matching rows.

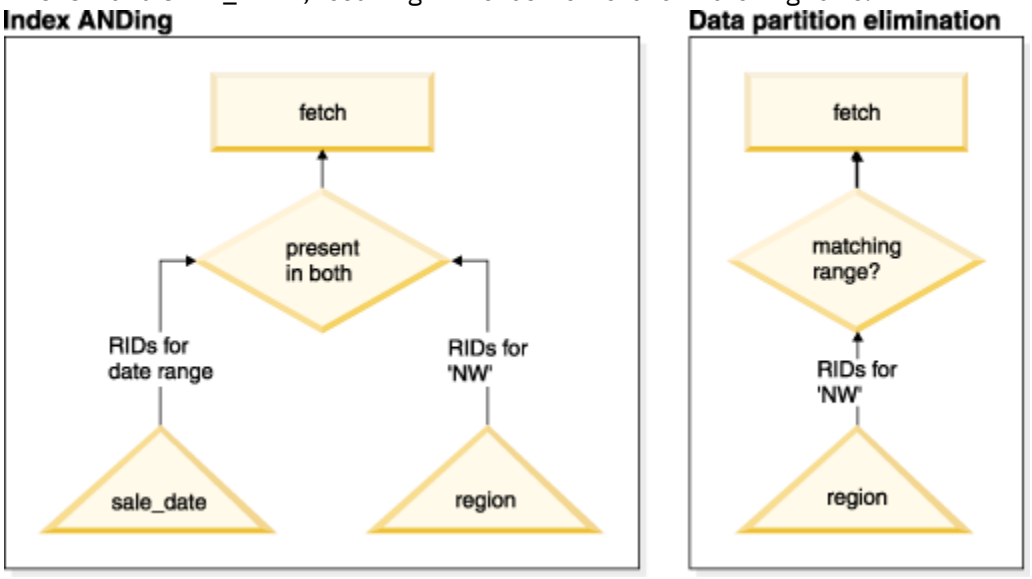


Figure 47. Optimizer decision path for both table partitioning and index ANDing

Db2 Explain

You can also use the explain facility to determine the data partition elimination plan that was chosen by the query optimizer. The "DP Elim Predicates" information shows which data partitions are scanned to resolve the following query:

```
   select * from custlist
     where subsdate between '12/31/1999' and '1/1/2001'
```

```
Arguments:
---------
DPESTFLG: (Number of data partitions accessed are Estimated)
          FALSE
DPLSTPRT: (List of data partitions accessed)
          9-11
DPNUMPRT: (Number of data partitions accessed)
         3

DP Elim Predicates:
------------------
Range 1)
     Stop  Predicate: (Q1.A <= '01/01/2001')
    Start Predicate: ('12/31/1999' <= Q1.A)
```

```
Objects Used in Access Plan:
 --------------------------

     Schema: MRSRINI
     Name:          CUSTLIST
     Type:          Data Partitioned Table
         Time of creation:               2005-11-30-14.21.33.857039
         Last statistics update:          2005-11-30-14.21.34.339392
          Number of columns:               3
          Number of rows:                100000
         Width of rows:                19
         Number of buffer pool pages:          1200
          Number of data partitions:           12
          Distinct row values:                No
          Tablespace name:                <VARIOUS>
```

Multi-column support

Data partition elimination works in cases where multiple columns are used as the table partitioning key. For example:

```
create table sales (
  year int, month int)
    partition by range(year, month) (
    starting from (2001,1)
    ending at (2001,3) in ts1,
      ending at (2001,6) in ts2,
      ending at (2001,9) in ts3,
      ending at (2001,12) in ts4,
      ending at (2002,3) in ts5,
      ending at (2002,6) in ts6,
      ending at (2002,9) in ts7,
      ending at (2002,12) in ts8)

select * from sales where year = 2001 and month < 8
```

The query optimizer deduces that only data partitions in TS1, TS2, and TS3 must be accessed to resolve this query.

Note: In the case where multiple columns make up the table partitioning key, data partition elimination is only possible when you have predicates on the leading columns of the composite key, because the non-leading columns that are used for the table partitioning key are not independent.

Multi-range support

It is possible to obtain data partition elimination with data partitions that have multiple ranges (that is, those that are ORed together). Using the SALES table that was created in the previous example, execute the following query:

```
select * from sales
  where (year = 2001 and month <= 3)
    or (year = 2002 and month >= 10)
```

The database server only accesses data for the first quarter of 2001 and the last quarter of 2002.

Generated columns

You can use generated columns as table partitioning keys. For example:

```
create table sales (
  a int, b int generated always as (a / 5))
  in ts1,ts2,ts3,ts4,ts5,ts6,ts7,ts8,ts9,ts10
  partition by range(b) (
    starting from (0)
    ending at (1000) every (50))
```

In this case, predicates on the generated column are used for data partition elimination. In addition, when the expression that is used to generate the columns is monotonic, the database server translates

predicates on the source columns into predicates on the generated columns, which enables data partition elimination on the generated columns. For example:

```
select * from sales where a > 35
```

The database server generates an extra predicate on b (b > 7) from a (a > 35), thus allowing data partition elimination.

Join predicates

Join predicates can also be used in data partition elimination, if the join predicate is pushed down to the table access level. The join predicate is only pushed down to the table access level on the inner join of a nested loop join (NLJN).

Consider the following tables:

```
create table t1 (a int, b int)
  partition by range(a,b) (
    starting from (1,1)
    ending (1,10) in ts1,
    ending (1,20) in ts2,
    ending (2,10) in ts3,
    ending (2,20) in ts4,
    ending (3,10) in ts5,
    ending (3,20) in ts6,
    ending (4,10) in ts7,
    ending (4,20) in ts8)

create table t2 (a int, b int)
```

The following two predicates will be used:

```
P1: T1.A = T2.A
P2: T1.B > 15
```

In this example, the exact data partitions that will be accessed at compile time cannot be determined, due to unknown outer values of the join. In this case, as well as cases where host variables or parameter markers are used, data partition elimination occurs at run time when the necessary values are bound.

During run time, when T1 is the inner of an NLJN, data partition elimination occurs dynamically, based on the predicates, for every outer value of T2.A. During run time, the predicates T1.A = 3 and T1.B > 15 are applied for the outer value T2.A = 3, which qualifies the data partitions in table space TS6 to be accessed.

Suppose that column A in tables T1 and T2 have the following values:

Outer table T2: column A	Inner table T1: column A	Inner table T1: column B	Inner table T1: data partition location
2	3	20	TS6
3	2	10	TS3
3	2	18	TS4
	3	15	TS6
	1	40	TS3

To perform a nested loop join (assuming a table scan for the inner table), the database manager performs the following steps:

1. Reads the first row from T2. The value for A is 2.
2. Binds the T2.A value (which is 2) to the column T2.A in the join predicate T1.A = T2.A. The predicate becomes T1.A = 2.
3. Applies data partition elimination using the predicates T1.A = 2 and T1.B > 15. This qualifies data partitions in table space TS4.

4. After applying T1.A = 2 and T1.B > 15, scans the data partitions in table space TS4 of table T1 until a row is found. The first qualifying row found is row 3 of T1.
5. Joins the matching row.
6. Scans the data partitions in table space TS4 of table T1 until the next match (T1.A = 2 and T1.B > 15) is found. No more rows are found.
7. Repeats steps 1 through 6 for the next row of T2 (replacing the value of A with 3) until all the rows of T2 have been processed.

Indexes over XML data

Starting in Db2 Version 9.7 Fix Pack 1, you can create an index over XML data on a partitioned table as either partitioned or nonpartitioned. The default is a partitioned index.

Partitioned and nonpartitioned XML indexes are maintained by the database manager during table insert, update, and delete operations in the same way as any other relational indexes on a partitioned table are maintained. Nonpartitioned indexes over XML data on a partitioned table are used in the same way as indexes over XML data on a nonpartitioned table to speed up query processing. Using the query predicate, it is possible to determine that only a subset of the data partitions in the partitioned table need to be accessed to answer the query.

Data partition elimination and indexes over XML columns can work together to enhance query performance. Consider the following partitioned table:

```
create table employee (a int, b xml, c xml)
  index in tbspx
  partition by (a) (
    starting 0 ending 10,
    ending 20,
    ending 30,
    ending 40)
```

Now consider the following query:

```
select * from employee
  where a > 21
  and xmlexist('$doc/Person/Name/First[.="Eric"]'
    passing "EMPLOYEE"."B" as "doc")
```

The optimizer can immediately eliminate the first two partitions based on the predicate a > 21. If the nonpartitioned index over XML data on column B is chosen by the optimizer in the query plan, an index scan using the index over XML data will be able to take advantage of the data partition elimination result from the optimizer and only return results belonging to partitions that were not eliminated by the relational data partition elimination predicates.

Optimization strategies for MDC tables

If you create multidimensional clustering (MDC) tables, the performance of many queries might improve, because the optimizer can apply additional optimization strategies. These strategies are primarily based on the improved efficiency of block indexes, but the advantage of clustering on more than one dimension also permits faster data retrieval.

MDC table optimization strategies can also exploit the performance advantages of intrapartition parallelism and interpartition parallelism. Consider the following specific advantages of MDC tables:

- Dimension block index lookups can identify the required portions of the table and quickly scan only the required blocks.
- Because block indexes are smaller than record identifier (RID) indexes, lookups are faster.
- Index ANDing and ORing can be performed at the block level and combined with RIDs.
- Data is guaranteed to be clustered on extents, which makes retrieval faster.
- Rows can be deleted faster when rollout can be used.

Consider the following simple example for an MDC table named SALES with dimensions defined on the REGION and MONTH columns:

```
select * from sales
  where month = 'March' and region = 'SE'
```

For this query, the optimizer can perform a dimension block index lookup to find blocks in which the month of March and the SE region occur. Then it can scan only those blocks to quickly fetch the result set.

Rollout deletion

When conditions permit delete using rollout, this more efficient way to delete rows from MDC tables is used. The required conditions are:

- The DELETE statement is a searched DELETE, not a positioned DELETE (the statement does not use the WHERE CURRENT OF clause).
- There is no WHERE clause (all rows are to be deleted), or the only conditions in the WHERE clause apply to dimensions.
- The table is not defined with the DATA CAPTURE CHANGES clause.
- The table is not the parent in a referential integrity relationship.
- The table does not have ON DELETE triggers defined.
- The table is not used in any MQTs that are refreshed immediately.
- A cascaded delete operation might qualify for rollout if its foreign key is a subset of the table's dimension columns.
- The DELETE statement cannot appear in a SELECT statement executing against the temporary table that identifies the set of affected rows prior to a triggering SQL operation (specified by the OLD TABLE AS clause on the CREATE TRIGGER statement).

During a rollout deletion, the deleted records are not logged. Instead, the pages that contain the records are made to look empty by reformatting parts of the pages. The changes to the reformatted parts are logged, but the records themselves are not logged.

The default behavior, *immediate cleanup rollout*, is to clean up RID indexes at delete time. This mode can also be specified by setting the **DB2_MDC_ROLLOUT** registry variable to IMMEDIATE, or by specifying IMMEDIATE on the SET CURRENT MDC ROLLOUT MODE statement. There is no change in the logging of index updates, compared to a standard delete operation, so the performance improvement depends on how many RID indexes there are. The fewer RID indexes, the better the improvement, as a percentage of the total time and log space.

An estimate of the amount of log space that is saved can be made with the following formula:

```
S + 38*N - 50*P
```

where *N* is the number of records deleted, *S* is total size of the records deleted, including overhead such as null indicators and VARCHAR lengths, and *P* is the number of pages in the blocks that contain the deleted records. This figure is the reduction in actual log data. The savings on active log space required is double that value, due to the saving of space that was reserved for rollback.

Alternatively, you can have the RID indexes updated after the transaction commits, using *deferred cleanup rollout*. This mode can also be specified by setting the **DB2_MDC_ROLLOUT** registry variable to DEFER, or by specifying DEFERRED on the SET CURRENT MDC ROLLOUT MODE statement. In a deferred rollout, RID indexes are cleaned up asynchronously in the background after the delete commits. This method of rollout can result in significantly faster deletion times for very large deletes, or when a number of RID indexes exist on the table. The speed of the overall cleanup operation is increased, because during a deferred index cleanup, the indexes are cleaned up in parallel, whereas in an immediate index cleanup, each row in the index is cleaned up one by one. Moreover, the transactional log space requirement for the DELETE statement is significantly reduced, because the asynchronous index cleanup logs the index updates by index page instead of by index key.

Note: Deferred cleanup rollout requires additional memory resources, which are taken from the database heap. If the database manager is unable to allocate the memory structures it requires, the deferred cleanup rollout fails, and a message is written to the administration notification log.

When to use a deferred cleanup rollout

If delete performance is the most important factor, and there are RID indexes defined on the table, use deferred cleanup rollout. Note that prior to index cleanup, index-based scans of the rolled-out blocks suffer a small performance penalty, depending on the amount of rolled-out data. The following issues should also be considered when deciding between immediate index cleanup and deferred index cleanup:

- Size of the delete operation

  Choose deferred cleanup rollout for very large deletions. In cases where dimensional DELETE statements are frequently issued on many small MDC tables, the overhead to asynchronously clean index objects might outweigh the benefit of time saved during the delete operation.
- Number and type of indexes

  If the table contains a number of RID indexes, which require row-level processing, use deferred cleanup rollout.
- Block availability

  If you want the block space freed by the delete operation to be available immediately after the DELETE statement commits, use immediate cleanup rollout.
- Log space

  If log space is limited, use deferred cleanup rollout for large deletions.
- Memory constraints

  Deferred cleanup rollout consumes additional database heap space on all tables that have deferred cleanup pending.

To disable rollout behavior during deletions, set the **DB2_MDC_ROLLOUT** registry variable to OFF or specify NONE on the SET CURRENT MDC ROLLOUT MODE statement.

Note: In Db2 Version 9.7 and later releases, deferred cleanup rollout is not supported on a data partitioned MDC table with partitioned RID indexes. Only the NONE and IMMEDIATE modes are supported. The cleanup rollout type will be IMMEDIATE if the **DB2_MDC_ROLLOUT** registry variable is set to DEFER, or if the CURRENT MDC ROLLOUT MODE special register is set to DEFERRED to override the **DB2_MDC_ROLLOUT** setting.

If only nonpartitioned RID indexes exist on the MDC table, deferred index cleanup rollout is supported.

Indexes

Indexes in partitioned tables

Index behavior on partitioned tables

Indexes on partitioned tables operate similarly to indexes on nonpartitioned tables. However, indexes on partitioned tables are stored using a different storage model, depending on whether the indexes are partitioned or nonpartitioned.

Although the indexes for a regular nonpartitioned table all reside in a shared index object, a *nonpartitioned index* on a partitioned table is created in its own index object in a single table space, even if the data partitions span multiple table spaces. Both database managed space (DMS) and system managed space (SMS) table spaces support the use of indexes in a different location than the table data. Each nonpartitioned index can be placed in its own table space, including large table spaces. Each index table space must use the same storage mechanism as the data partitions, either DMS or SMS. Indexes in large table spaces can contain up to 2^{29} pages. All of the table spaces must be in the same database partition group.

A *partitioned index* uses an index organization scheme in which index data is divided across multiple *index partitions*, according to the partitioning scheme of the table. Each index partition refers only to table rows in the corresponding data partition. All index partitions for a specific data partition reside in the same index object.

Starting in Db2 Version 9.7 Fix Pack 1, user-created indexes over XML data on XML columns in partitioned tables can be either partitioned or nonpartitioned. The default is partitioned. System-generated XML region indexes are always partitioned, and system-generated column path indexes are always nonpartitioned. In Db2 V9.7, indexes over XML data are nonpartitioned.

Benefits of a nonpartitioned index include:

- The fact that indexes can be reorganized independently of one another
- Improved performance of drop index operations
- The fact that when individual indexes are dropped, space becomes immediately available to the system without the need for index reorganization

Benefits of a partitioned index include:

- Improved data roll-in and roll-out performance
- Less contention on index pages, because the index is partitioned
- An index B-tree structure for each index partition, which can result in the following benefits:
  - Improved insert, update, delete, and scan performance because the B-tree for an index partition normally contains fewer levels than an index that references all data in the table
  - Improved scan performance and concurrency when partition elimination is in effect. Although partition elimination can be used for both partitioned and nonpartitioned index scans, it is more effective for partitioned index scans because each index partition contains keys for only the corresponding data partition. This configuration can result in having to scan fewer keys and fewer index pages than a similar query over a nonpartitioned index.

Although a nonpartitioned index always preserves order on the index columns, a partitioned index might lose some order across partitions in certain scenarios; for example, if the partitioning columns do not match the index columns, and more than one partition is to be accessed.

During online index creation, concurrent read and write access to the table is permitted. After an online index is built, changes that were made to the table during index creation are applied to the new index. Write access to the table is blocked until index creation completes and the transaction commits. For partitioned indexes, each data partition is quiesced to read-only access *only* while changes that were made to that data partition (during the creation of the index partition) are applied.

Partitioned index support becomes particularly beneficial when you are rolling data in using the ALTER TABLE...ATTACH PARTITION statement. If nonpartitioned indexes exist (not including the XML columns path index, if the table has XML data), issue a SET INTEGRITY statement after partition attachment. This statement is necessary for nonpartitioned index maintenance, range validation, constraints checking, and materialized query table (MQT) maintenance. Nonpartitioned index maintenance can be time-consuming and require large amounts of log space. Use partitioned indexes to avoid this maintenance cost.

If there are nonpartitioned indexes (except XML columns path indexes) on the table to maintain after an attach operation, the SET INTEGRITY...ALL IMMEDIATE UNCHECKED statement behaves as though it were a SET INTEGRITY...IMMEDIATE CHECKED statement. All integrity processing, nonpartitioned index maintenance, and table state transitions are performed as though a SET INTEGRITY...IMMEDIATE CHECKED statement was issued.

The Figure 48 on page 251 diagram shows two nonpartitioned indexes on a partitioned table, with each index in a separate table space.

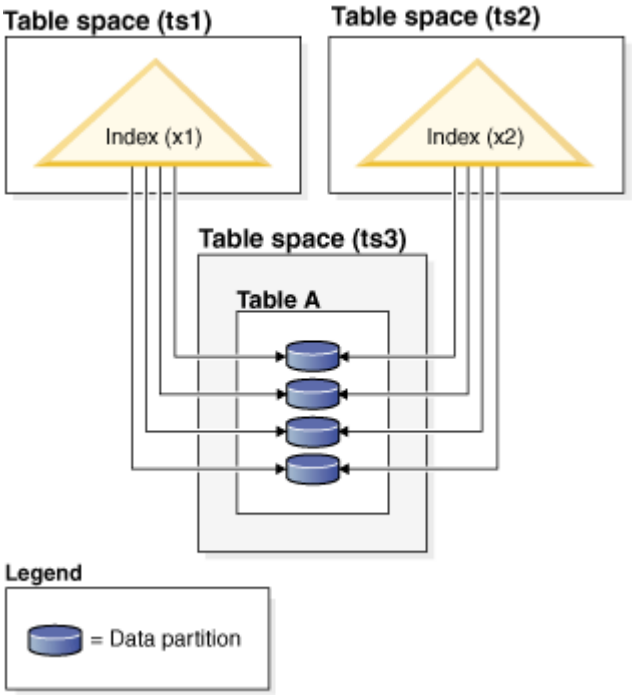


Figure 48. Nonpartitioned indexes on a partitioned table

The Figure 49 on page 252 diagram shows a partitioned index on a partitioned table that spans two database partitions and resides in a single table space.

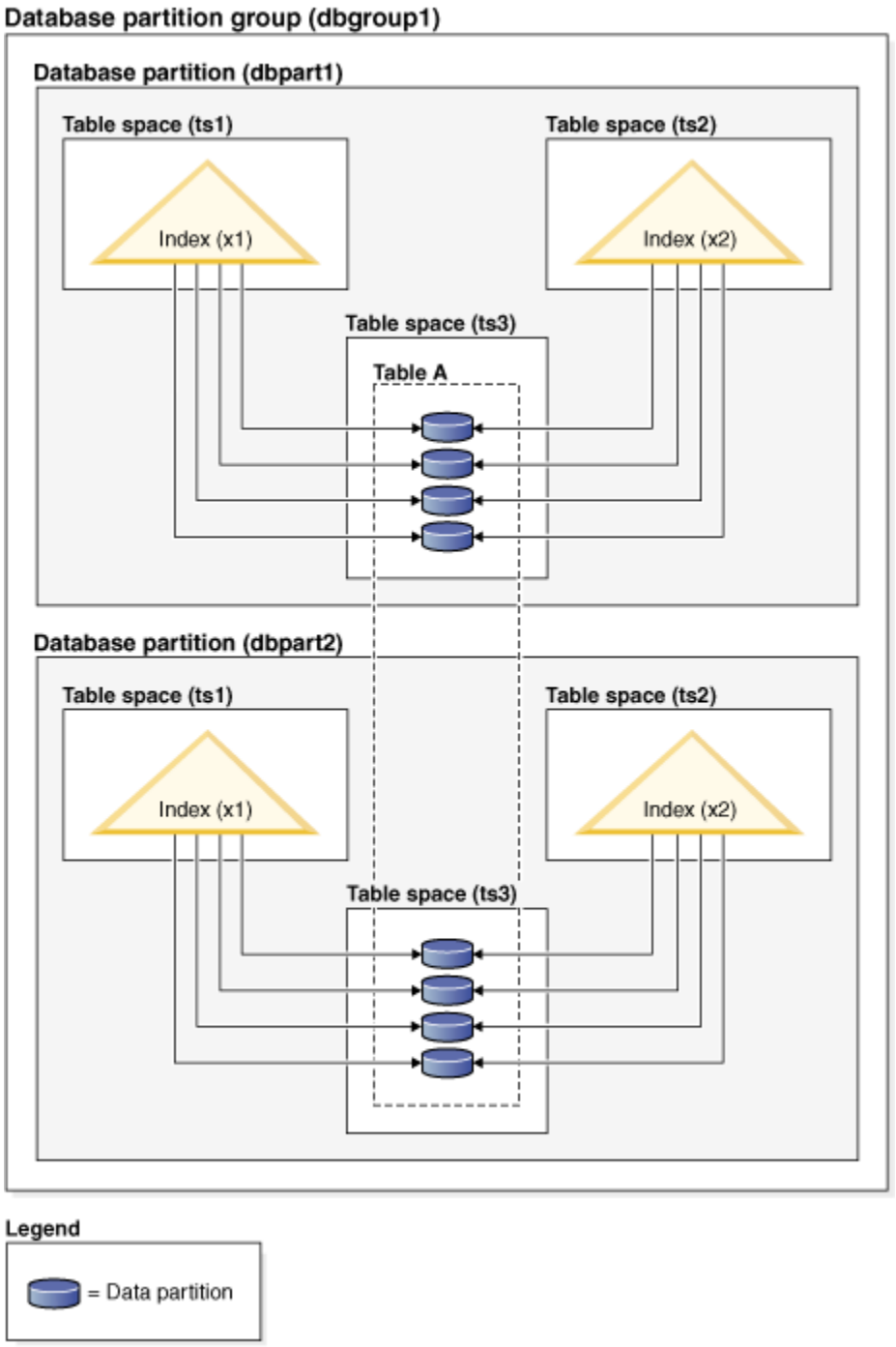


Figure 49. Nonpartitioned index on a table that is both distributed and partitioned

The Figure 50 on page 253 diagram shows a mix of partitioned and nonpartitioned indexes on a partitioned table.

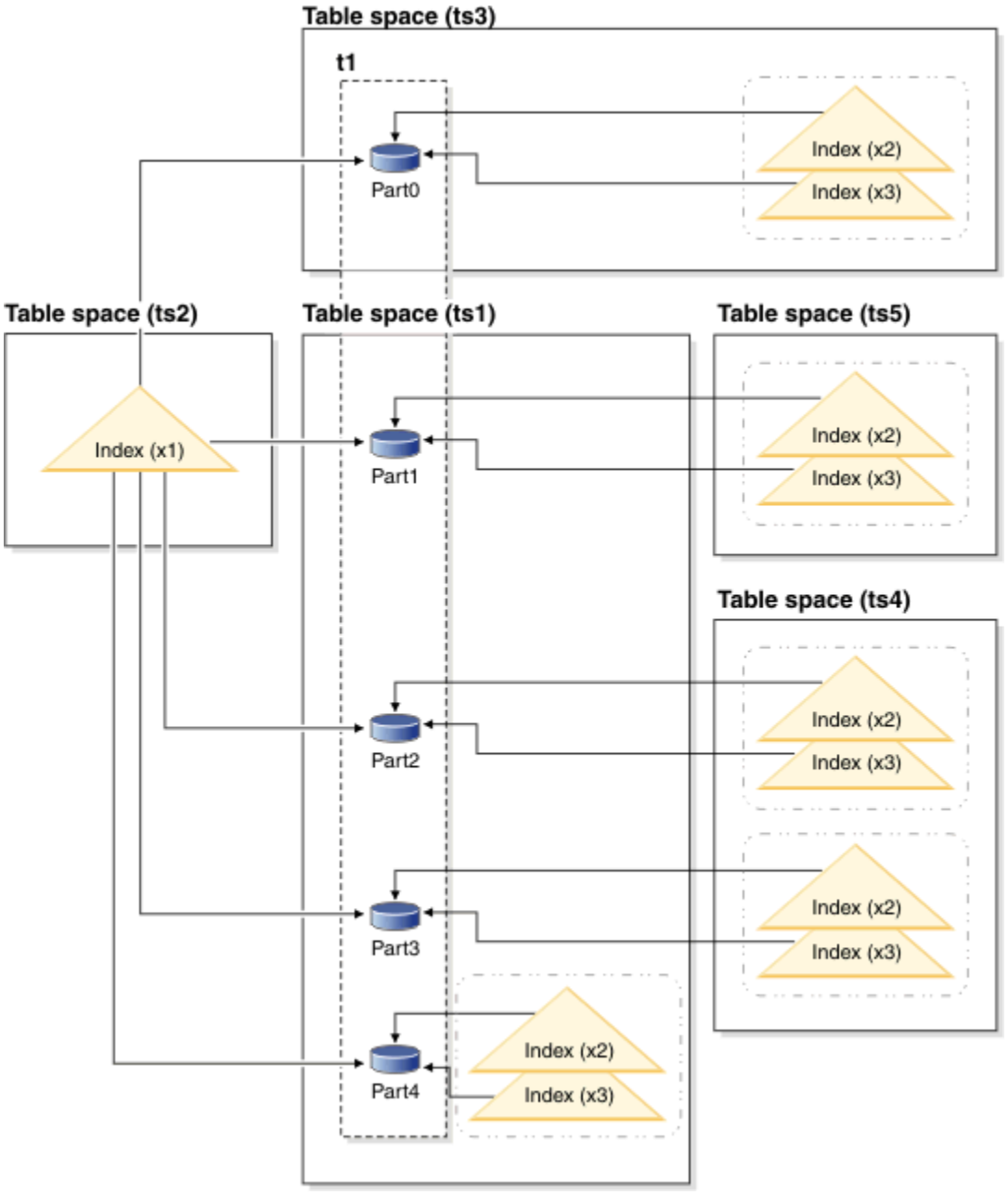


Figure 50. Partitioned and nonpartitioned indexes on a partitioned table

The nonpartitioned index X1 refers to rows in all of the data partitions. By contrast, the partitioned indexes X2 and X3 refer only to rows in the data partition with which they are associated. Table space TS3 also shows the index partitions sharing the table space of the data partitions with which they are associated. This configuration is the default for partitioned indexes.

You can override the default location for nonpartitioned and partitioned indexes, although the way that you do this is different for each. With nonpartitioned indexes, you can specify a table space when you create the index; for partitioned indexes, you need to determine the table spaces in which the index partitions are stored when you create the table.

Nonpartitioned indexes

To override the index location for nonpartitioned indexes, use the IN clause on the CREATE INDEX statement to specify an alternative table space location for the index. You can place different indexes in different table spaces, as required. If you create a partitioned table without specifying where to place its nonpartitioned indexes, and you then create an index by using a CREATE INDEX statement that does not specify a table space, the index is created in the table space of the first attached or

visible data partition. Each of the following three possible cases is evaluated in order, starting with case 1, to determine where the index is to be created. This evaluation to determine table space placement for the index stops when a matching case is found.

Case 1:
When an index table space is specified in the CREATE INDEX...IN *tbspace* statement, use the specified table space for this index.

Case 2:
When an index table space is specified in the CREATE TABLE... INDEX IN *tbspace* statement, use the specified table space for this index.

Case 3:
When no table space is specified, choose the table space that is used by the first attached or visible data partition.

Partitioned indexes

By default, index partitions are placed in the same table space as the data partitions that they reference. To override this default behavior, you must use the INDEX IN clause for each data partition that you define by using the CREATE TABLE statement. In other words, if you plan to use partitioned indexes for a partitioned table, you must anticipate where you want the index partitions to be stored when you create the table. If you try to use the INDEX IN clause when creating a partitioned index, you receive an error message.

Example 1: Given partitioned table SALES (a int, b int, c int), create a unique index A_IDX.

```
    create unique index a_idx on sales (a)
```

Because the table SALES is partitioned, index a_idx is also created as a partitioned index.

Example 2: Create index B_IDX.

```
    create index b_idx on sales (b)
```

Example 3: To override the default location for the index partitions in a partitioned index, use the INDEX IN clause for each partition that you define when creating the partitioned table. In the example that follows, indexes for the table Z are created in table space TS3.

```
create table z (a int, b int)
      partition by range (a) (starting from (1)
      ending at (100)  index in ts3)

   create index c_idx on z (a) partitioned
```

Clustering of nonpartitioned indexes on partitioned tables

Clustering indexes offer the same benefits for partitioned tables as they do for regular tables. However, care must be taken with the table partitioning key definitions when choosing a clustering index.

You can create a clustering index on a partitioned table using any clustering key. The database server attempts to use the clustering index to cluster data locally within each data partition. During a clustered insert operation, an index lookup is performed to find a suitable record identifier (RID). This RID is used as a starting point in the table when looking for space in which to insert the record. To achieve optimal clustering with good performance, there should be a correlation between the index columns and the table partitioning key columns. One way to ensure such correlation is to prefix the index columns with the table partitioning key columns, as shown in the following example:

```
    partition by range (month, region)
    create index...(month, region, department) cluster
```

Although the database server does not enforce this correlation, there is an expectation that all keys in the index will be grouped together by partition IDs to achieve good clustering. For example, suppose that a

table is partitioned on QUARTER and a clustering index is defined on DATE. There is a relationship between QUARTER and DATE, and optimal clustering of the data with good performance can be achieved because all keys of any data partition are grouped together within the index. Figure 51 on page 255 shows that optimal scan performance is achieved only when clustering correlates with the table partitioning key.

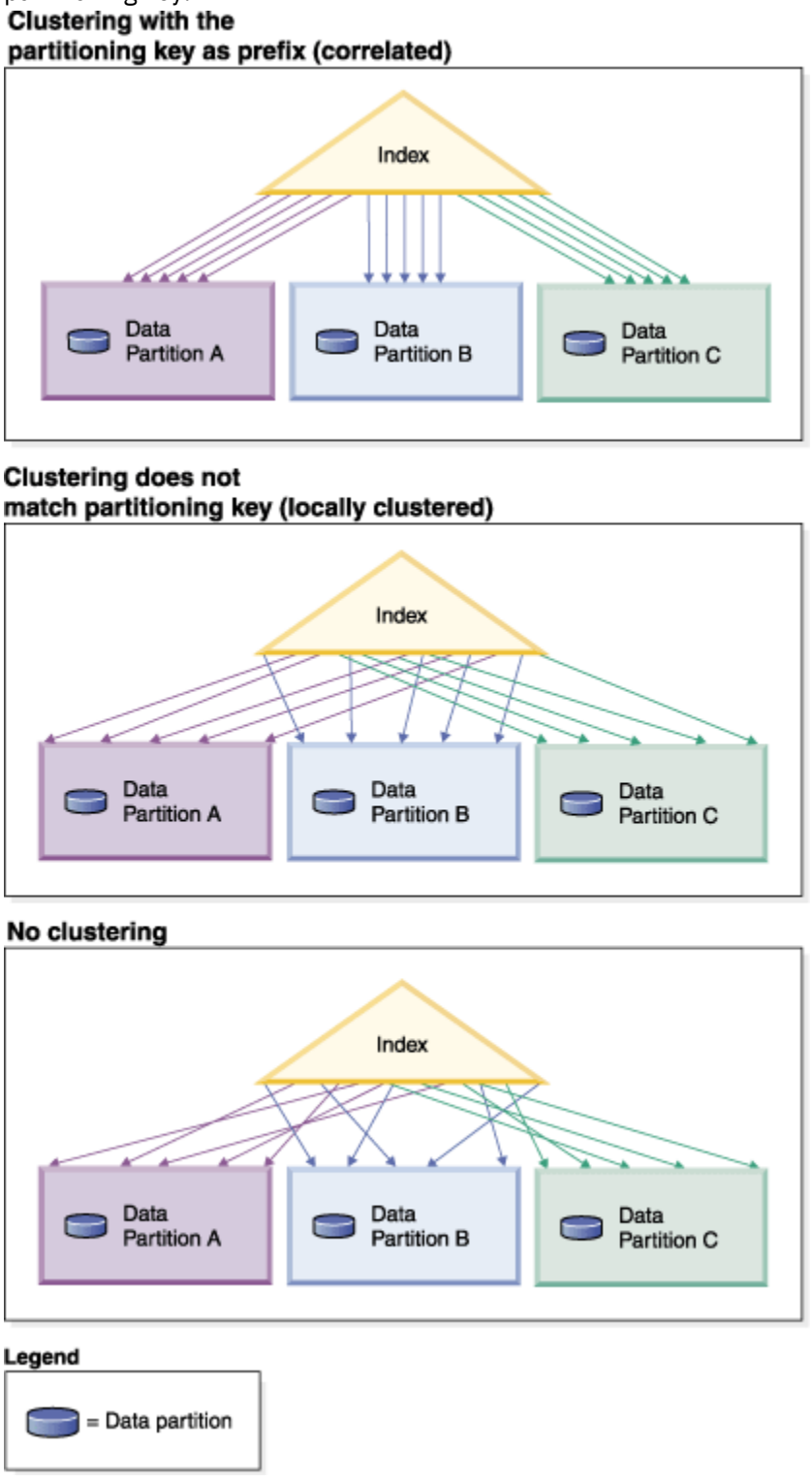


Figure 51. The possible effects of a clustered index on a partitioned table.

Benefits of clustering include:

- Rows are in key order within each data partition.

- Clustering indexes improve the performance of scans that traverse the table in key order, because the scanner fetches the first row of the first page, then each row in that same page before moving on to the next page. This means that only one page of the table needs to be in the buffer pool at any given time. In contrast, if the table is not clustered, rows are likely fetched from different pages. Unless the buffer pool can hold the entire table, most pages will likely be fetched more than once, greatly slowing down the scan.

If the clustering key is not correlated with the table partitioning key, but the data is locally clustered, you can still achieve the full benefit of the clustered index if there is enough space in the buffer pool to hold one page of each data partition. This is because each fetched row from a particular data partition is near the row that was previously fetched from that same partition (see the second example in Figure 51 on page 255).

Design advisor

Using the Design Advisor to convert from a single-partition to a multi-partition database

You can use the Design Advisor to help you convert a single-partition database into a multi-partition database.

About this task

In addition to making suggestions about new indexes, materialized query tables (MQTs), and multidimensional clustering (MDC) tables, the Design Advisor can provide you with suggestions for distributing data.

Procedure

1. Use the **db2licm** command to register the partitioned database environment license key.
2. Create at least one table space in a multi-partition database partition group.

   Note: The Design Advisor can only suggest data redistribution to existing table spaces.
3. Run the Design Advisor with the partitioning option specified on the **db2advis** command.
4. Modify the **db2advis** output file slightly before running the DDL statements that were generated by the Design Advisor.

   Because database partitioning must be set up before you can run the DDL script that the Design Advisor generates, suggestions are commented out of the script that is returned. It is up to you to transform your tables in accordance with the suggestions.

Managing concurrency

Lock modes for MDC and ITC tables and RID index scans

The type of lock that a multidimensional clustering (MDC) or insert time clustering (ITC) table obtains during a table or RID index scan depends on the isolation level that is in effect and on the data access plan that is being used.

The following tables show the types of locks that are obtained for MDC and ITC tables under each isolation level for different access plans. Each entry has three parts: the table lock, the block lock, and the row lock. A hyphen indicates that a particular lock granularity is not available.

Tables 9-14 show the types of locks that are obtained for RID index scans when the reading of data pages is deferred. Under the UR isolation level, if there are predicates on include columns in the index, the isolation level is upgraded to CS and the locks are upgraded to an IS table lock, an IS block lock, or NS row locks.

- Table 1. Lock Modes for Table Scans with No Predicates

- Table 2. Lock Modes for Table Scans with Predicates on Dimension Columns Only
- Table 3. Lock Modes for Table Scans with Other Predicates (sargs, resids)
- Table 4. Lock Modes for RID Index Scans with No Predicates
- Table 5. Lock Modes for RID Index Scans with a Single Qualifying Row
- Table 6. Lock Modes for RID Index Scans with Start and Stop Predicates Only
- Table 7. Lock Modes for RID Index Scans with Index Predicates Only
- Table 8. Lock Modes for RID Index Scans with Other Predicates (sargs, resids)
- Table 9. Lock Modes for Index Scans Used for Deferred Data Page Access: RID Index Scan with No Predicates
- Table 10. Lock Modes for Index Scans Used for Deferred Data Page Access: After a RID Index Scan with No Predicates
- Table 11. Lock Modes for Index Scans Used for Deferred Data Page Access: RID Index Scan with Predicates (sargs, resids)
- Table 12. Lock Modes for Index Scans Used for Deferred Data Page Access: After a RID Index Scan with Predicates (sargs, resids)
- Table 13. Lock Modes for Index Scans Used for Deferred Data Page Access: RID Index Scan with Start and Stop Predicates Only
- Table 14. Lock Modes for Index Scans Used for Deferred Data Page Access: After a RID Index Scan with Start and Stop Predicates Only

Note: Lock modes can be changed explicitly with the *lock-request-clause* of a SELECT statement.

Table 16. Lock Modes for Table Scans with No Predicates

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	S/-/-	U/-/-	SIX/IX/X	X/-/-	X/-/-
RS	IS/IS/NS	IX/IX/U	IX/IX/U	IX/X/-	IX/I/-
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-

Table 17. Lock Modes for Table Scans with Predicates on Dimension Columns Only

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	S/-/-	U/-/-	SIX/IX/X	U/-/-	SIX/X/-
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/U/-	X/X/-
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/U/-	X/X/-
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/U/-	X/X/-

Table 18. Lock Modes for Table Scans with Other Predicates (sargs, resids)

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	S/-/-	U/-/-	SIX/IX/X	U/-/-	SIX/IX/X

Table 18. Lock Modes for Table Scans with Other Predicates (sargs, resids) (continued)

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

Table 19. Lock Modes for RID Index Scans with No Predicates

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	S/-/-	IX/IX/S	IX/IX/X	X/-/-	X/-/-
RS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/X	X/X/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/X	X/X/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	X/X/X	X/X/X

Table 20. Lock Modes for RID Index Scans with a Single Qualifying Row

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/IS/S	IX/IX/U	IX/IX/X	X/X/X	X/X/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/X	X/X/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/X	X/X/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	X/X/X	X/X/X

Table 21. Lock Modes for RID Index Scans with Start and Stop Predicates Only

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/IS/S	IX/IX/S	IX/IX/X	IX/IX/X	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X

Table 22. Lock Modes for RID Index Scans with Index Predicates Only

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/S/S	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X

Table 22. Lock Modes for RID Index Scans with Index Predicates Only (continued)

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

Table 23. Lock Modes for RID Index Scans with Other Predicates (sargs, resids)

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/S/S	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

Table 24. Lock Modes for Index Scans Used for Deferred Data Page Access: RID Index Scan with No Predicates

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/S/S	IX/IX/S		X/-/-	
RS	IN/IN/-	IN/IN/-		IN/IN/-	
CS	IN/IN/-	IN/IN/-		IN/IN/-	
UR	IN/IN/-	IN/IN/-		IN/IN/-	

Table 25. Lock Modes for Index Scans Used for Deferred Data Page Access: After a RID Index Scan with No Predicates

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IN/IN/-	IX/IX/S	IX/IX/X	X/-/-	X/-/-
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X

Table 26. Lock Modes for Index Scans Used for Deferred Data Page Access: RID Index Scan with Predicates (sargs, resids)

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/S/-	IX/IX/S		IX/IX/S	
RS	IN/IN/-	IN/IN/-		IN/IN/-	
CS	IN/IN/-	IN/IN/-		IN/IN/-	
UR	IN/IN/-	IN/IN/-		IN/IN/-	

Table 27. Lock Modes for Index Scans Used for Deferred Data Page Access: After a RID Index Scan with Predicates (sargs, resids)

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IN/IN/-	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

Table 28. Lock Modes for Index Scans Used for Deferred Data Page Access: RID Index Scan with Start and Stop Predicates Only

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/IS/S	IX/IX/S		IX/IX/X	
RS	IN/IN/-	IN/IN/-		IN/IN/-	
CS	IN/IN/-	IN/IN/-		IN/IN/-	
UR	IN/IN/-	IN/IN/-		IN/IN/-	

Table 29. Lock Modes for Index Scans Used for Deferred Data Page Access: After a RID Index Scan with Start and Stop Predicates Only

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IN/IN/-	IX/IX/S	IX/IX/X	IX/IX/X	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IS/-/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

Lock modes for MDC block index scans

The type of lock that a multidimensional clustering (MDC) table obtains during a block index scan depends on the isolation level that is in effect and on the data access plan that is being used.

The following tables show the types of locks that are obtained for MDC tables under each isolation level for different access plans. Each entry has three parts: the table lock, the block lock, and the row lock. A hyphen indicates that a particular lock granularity is not available.

Tables 5-12 show the types of locks that are obtained for block index scans when the reading of data pages is deferred.

- Table 1. Lock Modes for Index Scans with No Predicates
- Table 2. Lock Modes for Index Scans with Predicates on Dimension Columns Only
- Table 3. Lock Modes for Index Scans with Start and Stop Predicates Only
- Table 4. Lock Modes for Index Scans with Predicates
- Table 5. Lock Modes for Index Scans Used for Deferred Data Page Access: Block Index Scan with No Predicates
- Table 6. Lock Modes for Index Scans Used for Deferred Data Page Access: After a Block Index Scan with No Predicates
- Table 7. Lock Modes for Index Scans Used for Deferred Data Page Access: Block Index Scan with Predicates on Dimension Columns Only
- Table 8. Lock Modes for Index Scans Used for Deferred Data Page Access: After a Block Index Scan with Predicates on Dimension Columns Only
- Table 9. Lock Modes for Index Scans Used for Deferred Data Page Access: Block Index Scan with Start and Stop Predicates Only
- Table 10. Lock Modes for Index Scans Used for Deferred Data Page Access: After a Block Index Scan with Start and Stop Predicates Only
- Table 11. Lock Modes for Index Scans Used for Deferred Data Page Access: Block Index Scan with Other Predicates (sargs, resids)
- Table 12. Lock Modes for Index Scans Used for Deferred Data Page Access: After a Block Index Scan with Other Predicates (sargs, resids)

Note: Lock modes can be changed explicitly with the *lock-request-clause* of a SELECT statement.

Table 30. Lock Modes for Index Scans with No Predicates

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	S/--/--	IX/IX/S	IX/IX/X	X/--/--	X/--/--
RS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/--	X/X/--
CS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/--	X/X/--
UR	IN/IN/-	IX/IX/U	IX/IX/X	X/X/--	X/X/--

Table 31. Lock Modes for Index Scans with Predicates on Dimension Columns Only

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/-/-	IX/IX/S	IX/IX/X	X/-/-	X/-/-
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-

Table 31. Lock Modes for Index Scans with Predicates on Dimension Columns Only (continued)

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-

Table 32. Lock Modes for Index Scans with Start and Stop Predicates Only

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/S/-	IX/IX/S	IX/IX/S	IX/IX/S	IX/IX/S
RS	IX/IX/S	IX/IX/U	IX/IX/X	IX/IX/-	IX/IX/-
CS	IX/IX/S	IX/IX/U	IX/IX/X	IX/IX/-	IX/IX/-
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/-	IX/IX/-

Table 33. Lock Modes for Index Scans with Predicates

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/S/-	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

Table 34. Lock Modes for Index Scans Used for Deferred Data Page Access: Block Index Scan with No Predicates

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/S/--	IX/IX/S		X/--/--	
RS	IN/IN/--	IN/IN/--		IN/IN/--	
CS	IN/IN/--	IN/IN/--		IN/IN/--	
UR	IN/IN/--	IN/IN/--		IN/IN/--	

Table 35. Lock Modes for Index Scans Used for Deferred Data Page Access: After a Block Index Scan with No Predicates

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IN/IN/--	IX/IX/S	IX/IX/X	X/--/--	X/--/--

Table 35. Lock Modes for Index Scans Used for Deferred Data Page Access: After a Block Index Scan with No Predicates (continued)

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/--	X/X/--
CS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/--	X/X/--
UR	IN/IN/--	IX/IX/U	IX/IX/X	X/X/--	X/X/--

Table 36. Lock Modes for Index Scans Used for Deferred Data Page Access: Block Index Scan with Predicates on Dimension Columns Only

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/S/--	IX/IX/--		IX/S/--	
RS	IS/IS/NS	IX/--/--		IX/--/--	
CS	IS/IS/NS	IX/--/--		IX/--/--	
UR	IN/IN/--	IX/--/--		IX/--/--	

Table 37. Lock Modes for Index Scans Used for Deferred Data Page Access: After a Block Index Scan with Predicates on Dimension Columns Only

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IN/IN/--	IX/IX/S	IX/IX/X	IX/S/--	IX/X/--
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/U/--	IX/X/--
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/U/--	IX/X/--
UR	IN/IN/--	IX/IX/U	IX/IX/X	IX/U/--	IX/X/--

Table 38. Lock Modes for Index Scans Used for Deferred Data Page Access: Block Index Scan with Start and Stop Predicates Only

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/S/--	IX/IX/--		IX/X/--	
RS	IN/IN/--	IN/IN/--		IN/IN/--	
CS	IN/IN/--	IN/IN/--		IN/IN/--	
UR	IN/IN/--	IN/IN/--		IN/IN/--	

Table 39. Lock Modes for Index Scans Used for Deferred Data Page Access: After a Block Index Scan with Start and Stop Predicates Only

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IN/IN/--	IX/IX/X		IX/X/--	
RS	IS/IS/NS	IN/IN/--		IN/IN/--	
CS	IS/IS/NS	IN/IN/--		IN/IN/--	
UR	IS/--/--	IN/IN/--		IN/IN/--	

Table 40. Lock Modes for Index Scans Used for Deferred Data Page Access: Block Index Scan with Other Predicates (sargs, resids)

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/S/--	IX/IX/--		IX/IX/--	
RS	IN/IN/--	IN/IN/--		IN/IN/--	
CS	IN/IN/--	IN/IN/--		IN/IN/--	
UR	IN/IN/--	IN/IN/--		IN/IN/--	

Table 41. Lock Modes for Index Scans Used for Deferred Data Page Access: After a Block Index Scan with Other Predicates (sargs, resids)

Isolation level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IN/IN/--	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/--	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

Locking behavior on partitioned tables

In addition to an overall table lock, there is a lock for each data partition of a partitioned table.

This allows for finer granularity and increased concurrency compared to a nonpartitioned table. The data partition lock is identified in output from the **db2pd** command, event monitors, administrative views, and table functions.

When a table is accessed, a table lock is obtained first, and then data partition locks are obtained as required. Access methods and isolation levels might require the locking of data partitions that are not represented 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 cost of reacquiring data partition locks later.

Data partition locks also carry the cost of ensuring access to table spaces. For nonpartitioned tables, table space access is handled by table locks. Data partition locking occurs even if there is an exclusive or share lock at the table level.

Finer granularity allows one transaction to have exclusive access to a specific data partition and to avoid row locking while other transactions are accessing other data partitions. This can be the result of the plan that is chosen for a mass update, or because of the 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 mode. 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 might be accessed, then a non-intent lock might be chosen at the table level to prevent data partition deadlocks between concurrent transactions.

LOCK TABLE statements

For partitioned tables, the only lock acquired by the LOCK TABLE statement is a table-level lock. This prevents row locking by subsequent data manipulation language (DML) statements, and avoids deadlocks at the row, block, or data partition level. The IN EXCLUSIVE MODE option can be used to guarantee exclusive access when updating indexes, which is useful in limiting the growth of indexes during a large update.

Effect of the LOCKSIZE TABLE option on the ALTER TABLE statement

The LOCKSIZE TABLE option ensures that a table is locked in share or exclusive mode with no intent locks. For a partitioned table, this locking strategy is applied to both the table lock and to data partition locks.

Row- and block-level lock escalation

Row- and block-level locks in partitioned tables can be escalated to the data partition level. When this occurs, a table is more accessible to other transactions, even if a data partition is escalated to share, exclusive, or super exclusive mode, because other data partitions remain unaffected. The notification log entry for an escalation includes the impacted data partition and the name of the table.

Exclusive access to a nonpartitioned index cannot be ensured by lock escalation. For exclusive access, one of the following conditions must be true:

- The statement must use an exclusive table-level lock
- An explicit LOCK TABLE IN EXCLUSIVE MODE statement must be issued
- The table must have the LOCKSIZE TABLE attribute

In the case of partitioned indexes, exclusive access to an index partition is ensured by lock escalation of the data partition to an exclusive or super exclusive access mode.

Interpreting lock information

The SNAPLOCK administrative view can help you to interpret lock information that is returned for a partitioned table. The following SNAPLOCK administrative view was captured during an offline index reorganization.

```
SELECT SUBSTR(TABNAME, 1, 15) TABNAME, TAB_FILE_ID, SUBSTR(TBSP_NAME, 1, 15)
TBSP_NAME,
       DATA_PARTITION_ID, LOCK_OBJECT_TYPE, LOCK_MODE, LOCK_ESCALATION L_ESCALATION
  FROM SYSIBMADM.SNAPLOCK
  WHERE TABNAME like 'TP1' and LOCK_OBJECT_TYPE like 'TABLE_%'
  ORDER BY TABNAME, DATA_PARTITION_ID, LOCK_OBJECT_TYPE, TAB_FILE_ID, LOCK_MODE


TABNAME TAB_FILE_ID TBSP_NAME  DATA_PARTITION_ID LOCK_OBJECT_TYPE LOCK_MODE
L_ESCALATION
------- ----------- ---------- ----------------- --------------- ---------
------------
TP1           32768 -                          -1 TABLE_LOCK
Z                       0
TP1               4 USERSPACE1                   0 TABLE_PART_LOCK
Z                       0
TP1               5 USERSPACE1                   1 TABLE_PART_LOCK
```

```
Z                    0
TP1                  6 USERSPACE1                  2 TABLE_PART_LOCK
Z                    0
TP1                  7 USERSPACE1                  3 TABLE_PART_LOCK
Z                    0
TP1                  8 USERSPACE1                  4 TABLE_PART_LOCK
Z                    0
TP1                  9 USERSPACE1                  5 TABLE_PART_LOCK
Z                    0
TP1                 10 USERSPACE1                  6 TABLE_PART_LOCK
Z                    0
TP1                 11 USERSPACE1                  7 TABLE_PART_LOCK
Z                    0
TP1                 12 USERSPACE1                  8 TABLE_PART_LOCK
Z                    0
TP1                 13 USERSPACE1                  9 TABLE_PART_LOCK
Z                    0
TP1                 14 USERSPACE1                 10 TABLE_PART_LOCK
Z                    0
TP1                 15 USERSPACE1                 11 TABLE_PART_LOCK
Z                    0
TP1                  4 USERSPACE1                  - TABLE_LOCK
Z                    0
TP1                  5 USERSPACE1                  - TABLE_LOCK
Z                    0
TP1                  6 USERSPACE1                  - TABLE_LOCK
Z                    0
TP1                  7 USERSPACE1                  - TABLE_LOCK
Z                    0
TP1                  8 USERSPACE1                  - TABLE_LOCK
Z                    0
TP1                  9 USERSPACE1                  - TABLE_LOCK
Z                    0
TP1                 10 USERSPACE1                  - TABLE_LOCK
Z                    0
TP1                 11 USERSPACE1                  - TABLE_LOCK
Z                    0
TP1                 12 USERSPACE1                  - TABLE_LOCK
Z                    0
TP1                 13 USERSPACE1                  - TABLE_LOCK
Z                    0
TP1                 14 USERSPACE1                  - TABLE_LOCK
Z                    0
TP1                 15 USERSPACE1                  - TABLE_LOCK
Z                    0
TP1                 16 USERSPACE1                  - TABLE_LOCK
Z                    0

  26 record(s) selected.
```

In this example, a lock object of type TABLE_LOCK and a DATA_PARTITION_ID of -1 are used to control access to and concurrency on the partitioned table TP1. The lock objects of type TABLE_PART_LOCK are used to control most access to and concurrency on each data partition.

There are additional lock objects of type TABLE_LOCK captured in this output (TAB_FILE_ID 4 through 16) that do not have a value for DATA_PARTITION_ID. A lock of this type, where an object with a TAB_FILE_ID and a TBSP_NAME correspond to a data partition or index on the partitioned table, might be used to control concurrency with the online backup utility.

Agent management

Agents in a partitioned database

In a partitioned database environment, or an environment in which intrapartition parallelism has been enabled, each database partition has its own pool of agents from which subagents are drawn.

Because of this pool, subagents do not have to be created and destroyed each time one is needed or has finished its work. The subagents can remain as associated agents in the pool and can be used by the

database manager for new requests from the application with which they are associated or from new applications.

The impact on both performance and memory consumption within the system is strongly related to how your agent pool is tuned. The database manager configuration parameter for agent pool size (**num_poolagents**) affects the total number of agents and subagents that can be kept associated with applications on a database partition. If the pool size is too small and the pool is full, a subagent disassociates itself from the application it is working on and terminates. Because subagents must be constantly created and reassociated with applications, performance suffers.

By default, **num_poolagents** is set to AUTOMATIC with a value of 100, and the database manager automatically manages the number of idle agents to pool.

If the value of **num_poolagents** is manually set too low, one application could fill the pool with associated subagents. Then, when another application requires a new subagent and has no subagents in its agent pool, it will recycle inactive subagents from the agent pools of other applications. This behavior ensures that resources are fully utilized.

If the value of **num_poolagents** is manually set too high, associated subagents might sit unused in the pool for long periods of time, using database manager resources that are not available for other tasks.

When the connection concentrator is enabled, the value of **num_poolagents** does not necessarily reflect the exact number of agents that might be idle in the pool at any one time. Agents might be needed temporarily to handle higher workload activity.

In addition to database agents, other asynchronous database manager activities run as their own process or thread, including:

- Database I/O servers or I/O prefetchers
- Database asynchronous page cleaners
- Database loggers
- Database deadlock detectors
- Communication and IPC listeners
- Table space container rebalancers

Optimizing access plans

Index access and cluster ratios

When it chooses an access plan, the optimizer estimates the number of I/Os that are required to fetch pages from disk to the buffer pool. This estimate includes a prediction of buffer pool usage, because additional I/Os are not required to read rows from a page that is already in the buffer pool.

For index scans, information from the system catalog helps the optimizer to estimate the I/O cost of reading data pages into a buffer pool. It uses information from the following columns in the SYSCAT.INDEXES view:

- CLUSTERRATIO information indicates the degree to which the table data is clustered in relation to this index. The higher the number, the better the rows are ordered in index key sequence. If table rows are in close to index-key sequence, rows can be read from a data page while the page is in the buffer. If the value of this column is -1, the optimizer uses PAGE_FETCH_PAIRS and CLUSTERFACTOR information, if it is available.
- The PAGE_FETCH_PAIRS column contains pairs of numbers that model the number of I/Os required to read the data pages into buffer pools of various sizes, together with CLUSTERFACTOR information. Data is collected for these columns only if you invoke the **RUNSTATS** command against the index, specifying the DETAILED clause.

If index clustering statistics are not available, the optimizer uses default values, which assume poor clustering of the data with respect to the index. The degree to which the data is clustered can have a significant impact on performance, and you should try to keep one of the indexes that are defined on the

table close to 100 percent clustered. In general, only one index can be one hundred percent clustered, except when the keys for an index represent a superset of the keys for the clustering index, or when there is an actual correlation between the key columns of the two indexes.

When you reorganize a table, you can specify an index that will be used to cluster the rows and keep them clustered during insert processing. Because update and insert operations can make a table less clustered in relation to the index, you might need to periodically reorganize the table. To reduce the number of reorganizations for a table that experiences frequent insert, update, or delete operations, specify the PCTFREE clause on the ALTER TABLE statement.

Table and index management for MDC and ITC tables

Table and index organization for multidimensional (MDC) and insert time clustering (ITC) tables is based on the same logical structures as standard table organization.

Like standard tables, MDC and ITC tables are organized into pages that contain rows of data divided into columns. The rows on each page are identified by record IDs (RIDs). However, the pages for MDC and ITC tables are grouped into extent-sized blocks. For example, Figure 52 on page 269, shows a table with an extent size of four. The first four pages, numbered 0 through 3, represent the first block in the table. The next four pages, numbered 4 through 7, represent the second block in the table.

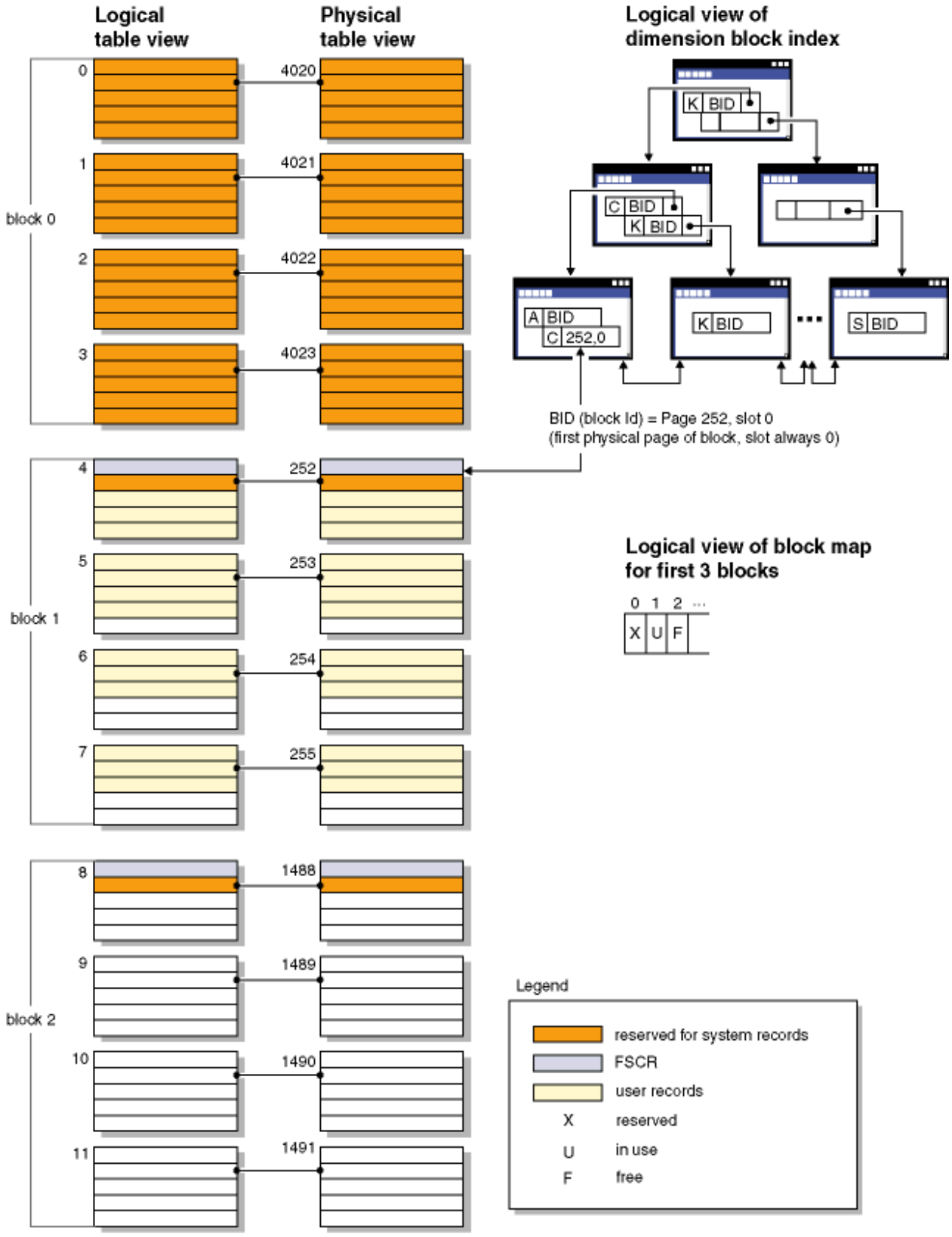


Figure 52. Logical table, record, and index structure for MDC and ITC tables

The first block contains special internal records, including the free space control record (FSCR), that are used by the Db2 server to manage the table. In subsequent blocks, the first page contains the FSCR. An FSCR maps the free space for new records that exists on each page of the block. This available free space is used when inserting records into the table.

As the name implies, MDC tables cluster data on more than one dimension. Each dimension is determined by a column or set of columns that you specify in the ORGANIZE BY DIMENSIONS clause of the CREATE TABLE statement. When you create an MDC table, the following two indexes are created automatically:

- A dimension-block index, which contains pointers to each occupied block for a single dimension
- A composite-block index, which contains all dimension key columns, and which is used to maintain clustering during insert and update activity

The optimizer considers access plans that use dimension-block indexes when it determines the most efficient access plan for a particular query. When queries have predicates on dimension values, the optimizer can use the dimension-block index to identify-and fetch from-the extents that contain these values. Because extents are physically contiguous pages on disk, this minimizes I/O and leads to better performance.

You can also create specific RID indexes if analysis of data access plans indicates that such indexes would improve query performance.

As the name implies, ITC tables cluster data based on row insert time. The differences between MDC and ITC tables are:

- block indexes are not used for any data access,
- only a single composite block index is created for the table, and that index consists of a virtual dimension, and
- the index is never chosen by the optimizer for plans because the column it contains cannot be referenced by any SQL statement.

MDC and ITC tables can have their empty blocks released to the table space.

Optimization strategies for intrapartition parallelism

The optimizer can choose an access plan to execute a query in parallel within a single database partition if a degree of parallelism is specified when the SQL statement is compiled.

At run time, multiple database agents called subagents are created to execute the query. The number of subagents is less than or equal to the degree of parallelism that was specified when the SQL statement was compiled.

To parallelize an access plan, the optimizer divides it into a portion that is run by each subagent and a portion that is run by the coordinating agent. The subagents pass data through table queues to the coordinating agent or to other subagents. In a partitioned database environment, subagents can send or receive data through table queues from subagents in other database partitions.

intrapartition parallel scan strategies

Relational scans and index scans can be performed in parallel on the same table or index. For parallel relational scans, the table is divided into ranges of pages or rows, which are assigned to subagents. A subagent scans its assigned range and is assigned another range when it has completed work on the current range.

For parallel index scans, the index is divided into ranges of records based on index key values and the number of index entries for a key value. The parallel index scan proceeds like a parallel table scan, with subagents being assigned a range of records. A subagent is assigned a new range when it has completed work on the current range.

Parallel table scans can be run against range partitioned tables, and similarly, parallel index scans can be run against partitioned indexes. For a parallel scan, partitioned indexes are divided into ranges of records, based on index key values and the number of key entries for a key value. When a parallel scan begins, subagents are assigned a range of records, and once the subagent completes a range, it is assigned a new range. The index partitions are scanned sequentially with subagents potentially scanning unreserved index partitions at any point in time without waiting for each other. Only the subset of index partitions that is relevant to the query based on data partition elimination analysis is scanned.

The optimizer determines the scan unit (either a page or a row) and the scan granularity.

Parallel scans provide an even distribution of work among the subagents. The goal of a parallel scan is to balance the load among the subagents and to keep them equally busy. If the number of busy subagents equals the number of available processors, and the disks are not overworked with I/O requests, the machine resources are being used effectively.

Other access plan strategies might cause data imbalance as the query executes. The optimizer chooses parallel strategies that maintain data balance among subagents.

intrapartition parallel sort strategies

The optimizer can choose one of the following parallel sort strategies:

- Round-robin sort

  This is also known as a *redistribution sort*. This method uses shared memory to efficiently redistribute the data as evenly as possible to all subagents. It uses a round-robin algorithm to provide the even distribution. It first creates an individual sort for each subagent. During the insert phase, subagents insert into each of the individual sorts in a round-robin fashion to achieve a more even distribution of data.

- Partitioned sort

  This is similar to the round-robin sort in that a sort is created for each subagent. The subagents apply a hash function to the sort columns to determine into which sort a row should be inserted. For example, if the inner and outer tables of a merge join are a partitioned sort, a subagent can use merge join to join the corresponding table portions and execute in parallel.

- Replicated sort

  This sort is used if each subagent requires all of the sort output. One sort is created and subagents are synchronized as rows are inserted into the sort. When the sort is complete, each subagent reads the entire sort. If the number of rows is small, this sort can be used to rebalance the data stream.

- Shared sort

  This sort is the same as a replicated sort, except that subagents open a parallel scan on the sorted result to distribute the data among the subagents in a way that is similar to a round-robin sort.

intrapartition parallel temporary tables

Subagents can cooperate to produce a temporary table by inserting rows into the same table. This is called a *shared temporary table*. The subagents can open private scans or parallel scans on the shared temporary table, depending on whether the data stream is to be replicated or split.

intrapartition parallel aggregation strategies

Aggregation operations can be performed by subagents in parallel. An aggregation operation requires the data to be ordered on the grouping columns. If a subagent can be guaranteed to receive all the rows for a set of grouping column values, it can perform a complete aggregation. This can happen if the stream is already split on the grouping columns because of a previous partitioned sort.

Otherwise, the subagent can perform a partial aggregation and use another strategy to complete the aggregation. Some of these strategies are:

- Send the partially aggregated data to the coordinator agent through a merging table queue. The coordinator agent completes the aggregation.
- Insert the partially aggregated data into a partitioned sort. The sort is split on the grouping columns and guarantees that all rows for a set of grouping columns are contained in one sort partition.
- If the stream needs to be replicated to balance processing, the partially aggregated data can be inserted into a replicated sort. Each subagent completes the aggregation using the replicated sort, and receives an identical copy of the aggregation result.

intrapartition parallel join strategies

Join operations can be performed by subagents in parallel. Parallel join strategies are determined by the characteristics of the data stream.

A join can be parallelized by partitioning or by replicating the data stream on the inner and outer tables of the join, or both. For example, a nested-loop join can be parallelized if its outer stream is partitioned for a parallel scan and the inner stream is again evaluated independently by each subagent. A merged join can be parallelized if its inner and outer streams are value-partitioned for partitioned sorts.

Data filtering and data skew can cause workloads between subagents to become imbalanced while a query executes. The inefficiency of imbalanced workloads is magnified by joins and other computationally expensive operations. The optimizer looks for sources of imbalance in the query's access plan and applies a balancing strategy, ensuring that work is evenly divided between the subagents. For an unordered outer data stream, the optimizer balances the join using the REBAL operator on the outer data stream. For an ordered data stream (where ordered data is produced by an index access or a sort), the optimizer balances the data using a shared sort. A shared sort will be not be used if the sort overflows into the temporary tables, due to the high cost of a sort overflow.

Joins

A *join* is the process of combining data from two or more tables based on some common domain of information. Rows from one table are paired with rows from another table when information in the corresponding rows match on the basis of the joining criterion (the *join predicate*).

For example, consider the following two tables:

TABLE1	
PROJ	**PROJ_ID**
A	1
B	2
C	3
D	4

TABLE2	
PROJ_ID	**NAME**
1	Sam
3	Joe
4	Mary
1	Sue
2	Mike

To join TABLE1 and TABLE2, such that the PROJ_ID columns have the same values, use the following SQL statement:

```
select proj, x.proj_id, name
  from table1 x, table2 y
  where x.proj_id = y.proj_id
```

In this case, the appropriate join predicate is: `where x.proj_id = y.proj_id`.

The query yields the following result set:

PROJ	PROJ_ID	NAME
A	1	Sam
A	1	Sue
B	2	Mike
C	3	Joe
D	4	Mary

Depending on the nature of any join predicates, as well as any costs determined on the basis of table and index statistics, the optimizer chooses one of the following join methods:

- Nested-loop join

- Merge join
- Hash join

When two tables are joined, one table is selected as the outer table and the other table is regarded as the inner table of the join. The outer table is accessed first and is scanned only once. Whether the inner table is scanned multiple times depends on the type of join and the indexes that are available. Even if a query joins more than two tables, the optimizer joins only two tables at a time. If necessary, temporary tables are created to hold intermediate results.

You can provide explicit join operators, such as INNER or LEFT OUTER JOIN, to determine how tables are used in the join. Before you alter a query in this way, however, you should allow the optimizer to determine how to join the tables, and then analyze query performance to decide whether to add join operators.

Database partition group impact on query optimization
In partitioned database environments, the optimizer recognizes and uses the collocation of tables when it determines the best access plan for a query.

If tables are frequently involved in join queries, they should be divided among database partitions in such a way that the rows from each table being joined are located on the same database partition. During the join operation, the collocation of data from both joined tables prevents the movement of data from one database partition to another. Place both tables in the same database partition group to ensure that the data is collocated.

Depending on the size of the table, spreading data over more database partitions reduces the estimated time to execute a query. The number of tables, the size of the tables, the location of the data in those tables, and the type of query (such as whether a join is required) all affect the cost of the query.

Join strategies for partitioned databases
Join strategies for a partitioned database environment can be different than strategies for a nonpartitioned database environment. Additional techniques can be applied to standard join methods to improve performance.

Table collocation should be considered for tables that are frequently joined. In a partitioned database environment, *table collocation* refers to a state that occurs when two tables that have the same number of compatible partitioning keys are stored in the same database partition group. When this happens, join processing can be performed at the database partition where the data is stored, and only the result set needs to be moved to the coordinator database partition.

Table queues

Descriptions of join techniques in a partitioned database environment use the following terminology:

- *Table queue* (sometimes referred to as TQ) is a mechanism for transferring rows between database partitions, or between processors in a single-partition database.
- *Directed table queue* (sometimes referred to as DTQ) is a table queue in which rows are hashed to one of the receiving database partitions.
- *Broadcast table queue* (sometimes referred to as BTQ) is a table queue in which rows are sent to all of the receiving database partitions, but are not hashed.

A table queue is used to pass table data:

- From one database partition to another when using interpartition parallelism
- Within a database partition when using intrapartition parallelism
- Within a database partition when using a single-partition database

Each table queue passes the data in a single direction. The compiler decides where table queues are required, and includes them in the plan. When the plan is executed, connections between the database partitions initiate the table queues. The table queues close as processes end.

There are several types of table queues:

- Asynchronous table queues

  These table queues are known as asynchronous, because they read rows in advance of any fetch requests from an application. When a FETCH statement is issued, the row is retrieved from the table queue.

  Asynchronous table queues are used when you specify the FOR FETCH ONLY clause on the SELECT statement. If you are only fetching rows, the asynchronous table queue is faster.
- Synchronous table queues

  These table queues are known as synchronous, because they read one row for each FETCH statement that is issued by an application. At each database partition, the cursor is positioned on the next row to be read from that database partition.

  Synchronous table queues are used when you do not specify the FOR FETCH ONLY clause on the SELECT statement. In a partitioned database environment, if you are updating rows, the database manager will use synchronous table queues.
- Merging table queues

  These table queues preserve order.
- Non-merging table queues

  These table queues, also known as *regular table queues*, do not preserve order.
- Listener table queues (sometimes referred to as LTQ)

  These table queues are used with correlated subqueries. Correlation values are passed down to the subquery, and the results are passed back up to the parent query block using this type of table queue.

Join methods for partitioned databases

Several join methods are available for partitioned database environments, including: collocated joins, broadcast outer-table joins, directed outer-table joins, directed inner-table and outer-table joins, broadcast inner-table joins, and directed inner-table joins.

In the following diagrams, q1, q2, and q3 refer to table queues. The referenced tables are divided across two database partitions, and the arrows indicate the direction in which the table queues are sent. The coordinator database partition is database partition 0.

If the join method chosen by the compiler is hash join, the filters created at each remote database partition may be used to eliminate tuples before they are sent to the database partition where the hash join is processed, thus improving performance.

Collocated joins

A collocated join occurs locally on the database partition on which the data resides. The database partition sends the data to the other database partitions after the join is complete. For the optimizer to consider a collocated join, the joined tables must be collocated, and all pairs of the corresponding distribution keys must participate in the equality join predicates. Figure 53 on page 275 provides an example.

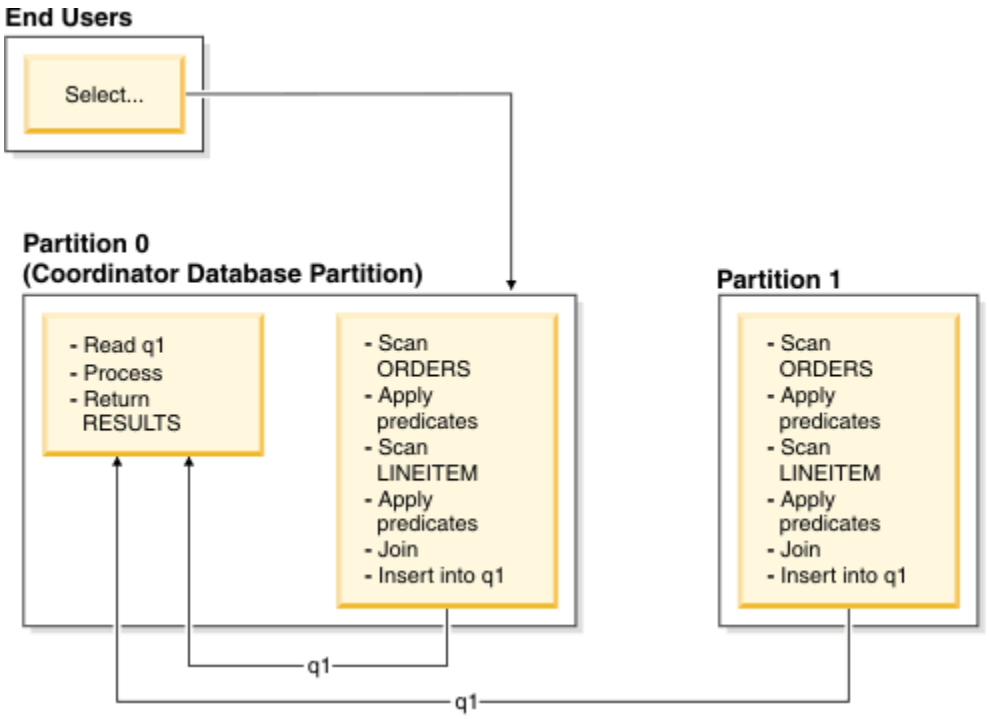


Figure 53. Collocated Join Example

The LINEITEM and ORDERS tables are both partitioned on the ORDERKEY column. The join is performed locally at each database partition. In this example, the join predicate is assumed to be: `orders.orderkey = lineitem.orderkey`.

Replicated materialized query tables (MQTs) enhance the likelihood of collocated joins.

Broadcast outer-table joins

Broadcast outer-table joins represent a parallel join strategy that can be used if there are no equality join predicates between the joined tables. It can also be used in other situations in which it proves to be the most cost-effective join method. For example, a broadcast outer-table join might occur when there is one very large table and one very small table, neither of which is split on the join predicate columns. Instead of splitting both tables, it might be cheaper to broadcast the smaller table to the larger table. Figure 54 on page 276 provides an example.

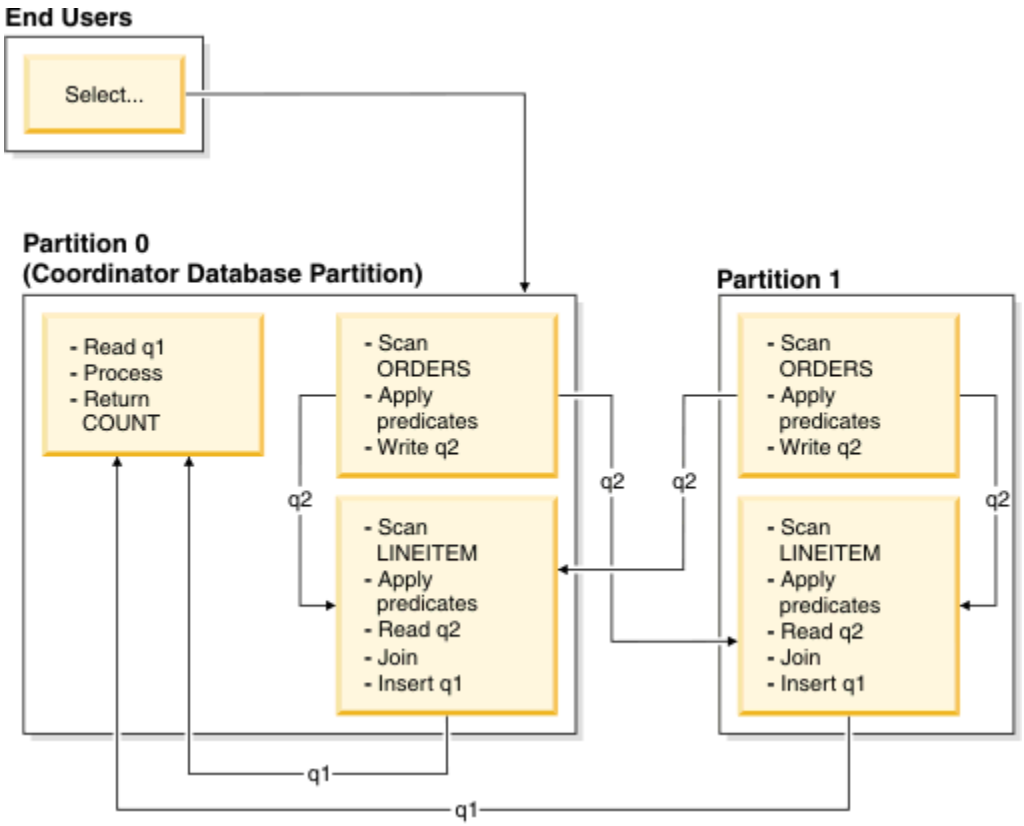
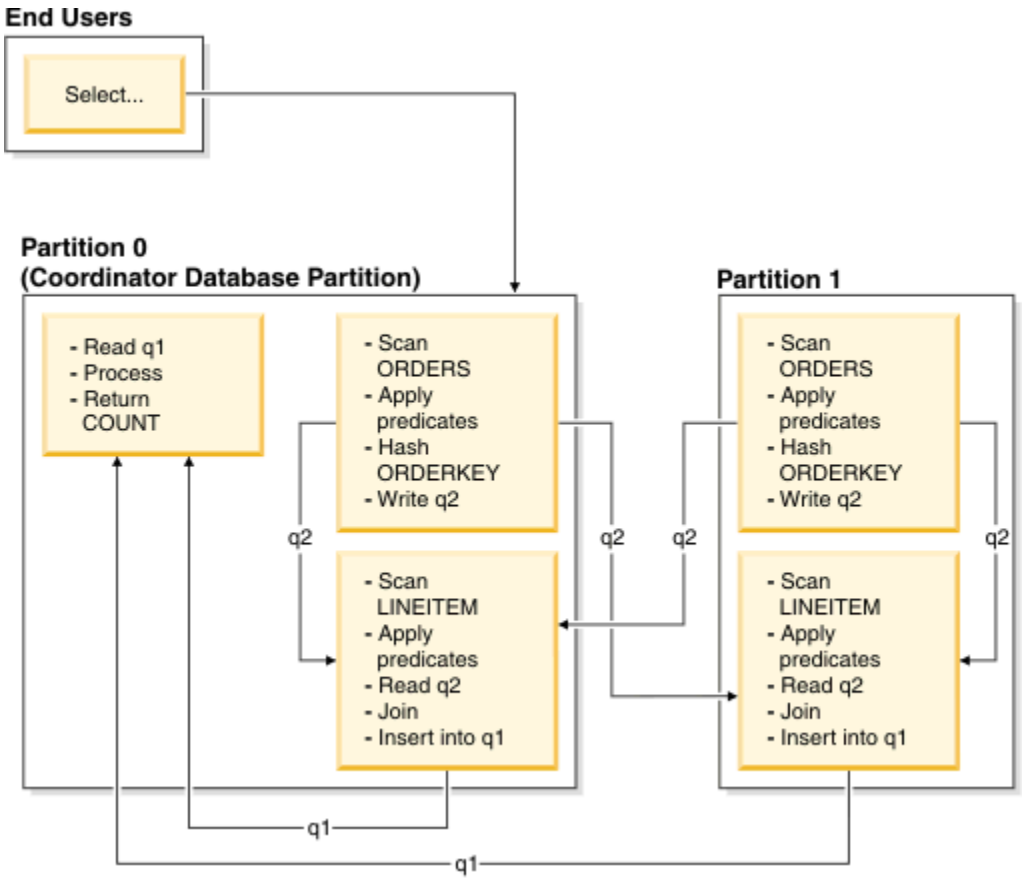


Figure 54. Broadcast Outer-Table Join Example

The ORDERS table is sent to all database partitions that have the LINEITEM table. Table queue q2 is broadcast to all database partitions of the inner table.

Directed outer-table joins

In the directed outer-table join strategy, each row of the outer table is sent to one portion of the inner table, based on the splitting attributes of the inner table. The join occurs on this database partition. Figure 55 on page 277 provides an example.



The LINEITEM table is partitioned on the ORDERKEY column. The ORDERS table is partitioned on a different column. The ORDERS table is hashed and sent to the correct database partition of the LINEITEM table. In this example, the join predicate is assumed to be: `orders.orderkey = lineitem.orderkey`.

Figure 55. Directed Outer-Table Join Example

Directed inner-table and outer-table joins

In the directed inner-table and outer-table join strategy, rows of both the outer and inner tables are directed to a set of database partitions, based on the values of the joining columns. The join occurs on these database partitions. Figure 56 on page 278 provides an example.

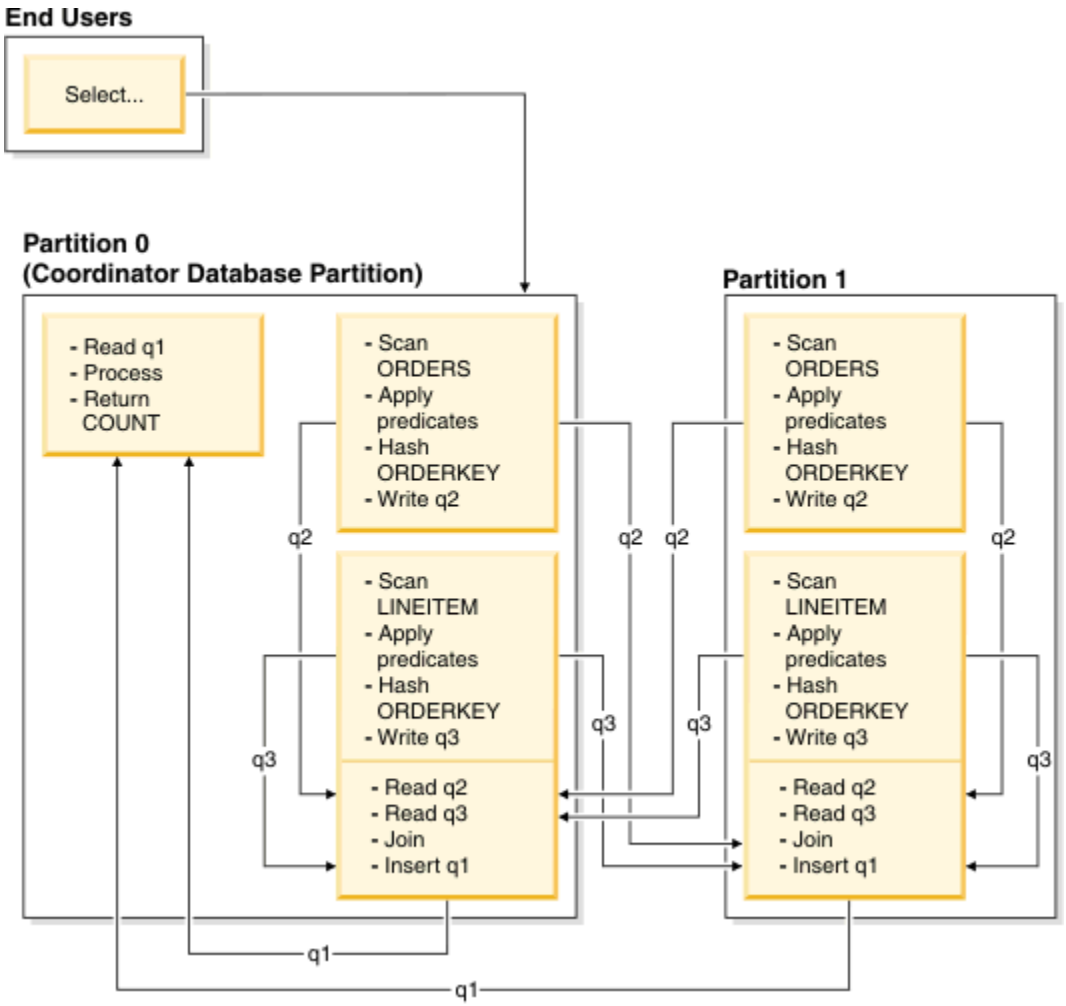
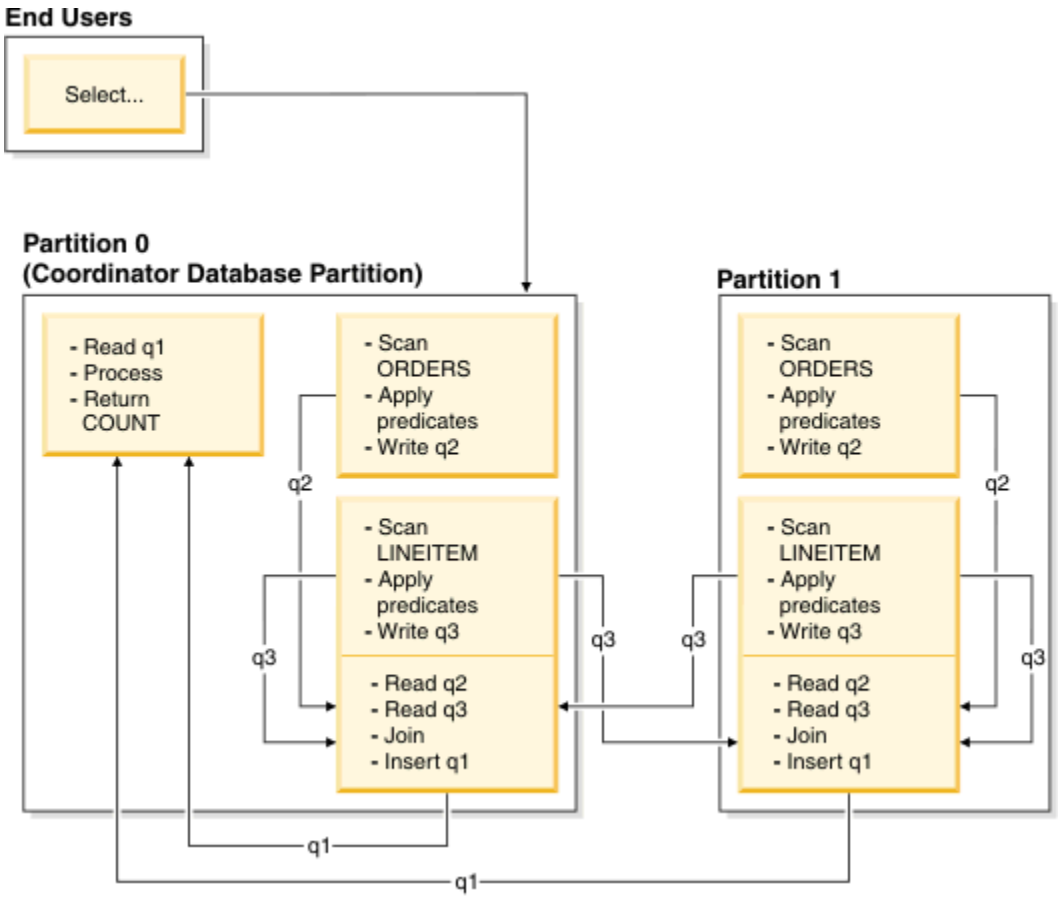


Figure 56. Directed Inner-Table and Outer-Table Join Example

Neither table is partitioned on the ORDERKEY column. Both tables are hashed and sent to new database partitions, where they are joined. Both table queue q2 and q3 are directed. In this example, the join predicate is assumed to be: `orders.orderkey = lineitem.orderkey`.

Broadcast inner-table joins

In the broadcast inner-table join strategy, the inner table is broadcast to all the database partitions of the outer table. Figure 57 on page 279 provides an example.

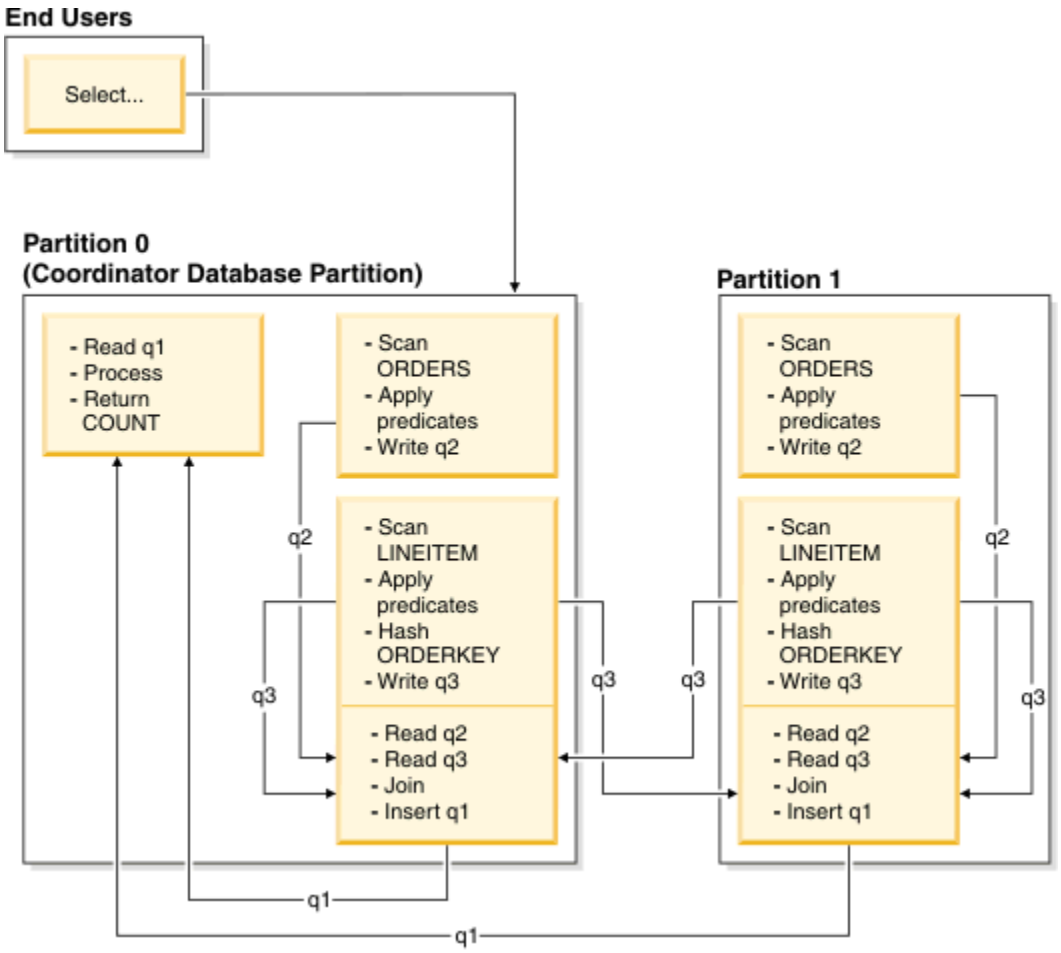


The LINEITEM table is sent to all database partitions that have the ORDERS table. Table queue q3 is broadcast to all database partitions of the outer table.

Figure 57. Broadcast Inner-Table Join Example

Directed inner-table joins

In the directed inner-table join strategy, each row of the inner table is sent to one database partition of the outer table, based on the splitting attributes of the outer table. The join occurs on this database partition. Figure 58 on page 280 provides an example.



The ORDERS table is partitioned on the ORDERKEY column. The LINEITEM table is partitioned on a different column. The LINEITEM table is hashed and sent to the correct database partition of the ORDERS table. In this example, the join predicate is assumed to be: `orders.orderkey = lineitem.orderkey`.

Figure 58. Directed Inner-Table Join Example

Replicated materialized query tables in partitioned database environments

Replicated materialized query tables (MQTs) improve the performance of frequently executed joins in a partitioned database environment by allowing the database to manage precomputed values of the table data.

Note that a replicated MQT in this context pertains to intra-database replication. Inter-database replication is concerned with subscriptions, control tables, and data that is located in different databases and on different operating systems.

In the following example:

- The SALES table is in a multi-partition table space named REGIONTABLESPACE, and is split on the REGION column.
- The EMPLOYEE and DEPARTMENT tables are in a single-partition database partition group.

Create a replicated MQT based on information in the EMPLOYEE table.

```
create table r_employee as (
  select empno, firstnme, midinit, lastname, workdept
    from employee
  )
  data initially deferred refresh immediate
  in regiontablespace
  replicated
```

Update the content of the replicated MQT:

```
refresh table r_employee
```

After using the REFRESH statement, you should invoke the runstats utility against the replicated table, as you would against any other table.

The following query calculates sales by employee, the total for the department, and the grand total:

```
select d.mgrno, e.empno, sum(s.sales)
  from department as d, employee as e, sales as s
  where
    s.sales_person = e.lastname and
    e.workdept = d.deptno
  group by rollup(d.mgrno, e.empno)
  order by d.mgrno, e.empno
```

Instead of using the EMPLOYEE table, which resides on only one database partition, the database manager uses R_EMPLOYEE, the MQT that is replicated on each of the database partitions on which the SALES table is stored. The performance enhancement occurs because the employee information does not have to be moved across the network to each database partition when performing the join.

Replicated materialized query tables in collocated joins

Replicated MQTs can also assist in the collocation of joins. For example, if a star schema contains a large fact table that is spread across twenty database partitions, the joins between the fact table and the dimension tables are most efficient if these tables are collocated. If all of the tables are in the same database partition group, at most one dimension table is partitioned correctly for a collocated join. The other dimension tables cannot be used in a collocated join, because the join columns in the fact table do not correspond to the distribution key for the fact table.

Consider a table named FACT (C1, C2, C3, ...), split on C1; a table named DIM1 (C1, dim1a, dim1b, ...), split on C1; a table named DIM2 (C2, dim2a, dim2b, ...), split on C2; and so on. In this case, the join between FACT and DIM1 is perfect, because the predicate `dim1.c1 = fact.c1` is collocated. Both of these tables are split on column C1.

However, the join involving DIM2 and the predicate `dim2.c2 = fact.c2` cannot be collocated, because FACT is split on column C1, not on column C2. In this case, you could replicate DIM2 in the database partition group of the fact table so that the join occurs locally on each database partition.

When you create a replicated MQT, the source table can be a single-partition table or a multi-partition table in a database partition group. In most cases, the replicated table is small and can be placed in a single-partition database partition group. You can limit the data that is to be replicated by specifying only a subset of the columns from the table, or by restricting the number of qualifying rows through predicates.

A replicated MQT can also be created in a multi-partition database partition group, so that copies of the source table are created on all of the database partitions. Joins between a large fact table and the dimension tables are more likely to occur locally in this environment, than if you broadcast the source table to all database partitions.

Indexes on replicated tables are not created automatically. You can create indexes that are different from those on the source table. However, to prevent constraints violations that are not present in the source table, you cannot create unique indexes or define constraints on replicated tables, even if the same constraints occur on the source table.

Replicated tables can be referenced directly in a query, but you cannot use the DBPARTITIONNUM scalar function with a replicated table to see the table data on a particular database partition.

Use the Db2 explain facility to determine whether a replicated MQT was used by the access plan for a query. Whether or not the access plan that is chosen by the optimizer uses a replicated MQT depends on the data that is to be joined. A replicated MQT might not be used if the optimizer determines that it would be cheaper to broadcast the original source table to the other database partitions in the database partition group.

Data redistribution

Data redistribution is a database administration operation that can be performed to primarily move data within a partitioned database environment when partitions are added or removed. The goal of this operation is typically to balance the usage of storage space, improve database system performance, or satisfy other system requirements.

Data redistribution can be performed by using one of the following interfaces:

- **REDISTRIBUTE DATABASE PARTITION GROUP** command
- ADMIN_CMD built-in procedure
- STEPWISE_REDISTRIBUTE_DBPG built-in procedure
- `sqludrdt` API

Data redistribution within a partitioned database is done for one of the following reasons:

- To rebalance data whenever a new database partition is added to the database environment or an existing database partition is removed.
- To introduce user-specific data distribution across partitions.
- To secure sensitive data by isolating it within a particular partition.

Data redistribution is performed by connecting to a database at the catalog database partition and beginning a data redistribution operation for a specific partition group by using one of the supported interfaces. Data redistribution relies on the existence of distribution key definitions for the tables within the partition group. The distribution key value for a row of data within the table is used to determine on which partition the row of data will be stored. A distribution key is generated automatically when a table is created in a multi-partition database partition group. A distribution key can also be explicitly defined by using the CREATE TABLE or ALTER TABLE statements. By default during data redistribution, for each table within a specified database partition group, table data is divided and redistributed evenly among the database partitions. Other distributions, such as a skewed distribution, can be achieved by specifying an input distribution map which defines how the data is to be distributed. Distribution maps can be generated during a data redistribution operation for future use or can be created manually.

Comparison of logged, recoverable redistribution and minimally logged, not roll-forward recoverable redistribution

When performing data redistribution by using either the **REDISTRIBUTE DATABASE PARTITION GROUP** command or the ADMIN_CMD built-in procedure, you can choose between two methods of data redistribution: logged, recoverable redistribution and minimally logged, not roll-forward recoverable redistribution. The latter method is specified by using the **NOT ROLLFORWARD RECOVERABLE** command parameter.

Data redistribution in capacity growth scenarios, during load balancing, or during performance tuning can require precious maintenance window time, a considerable amount of planning time, as well as log space and extra container space that can be expensive. Your choice of redistribution methods depends on whether you prioritize recoverability or speed:

- When the logged, recoverable redistribution method is used, extensive logging of all row movement is performed such that the database can be recovered in the event of any interruptions, errors, or other business need.
- The not roll-forward recoverable redistribution method offers better performance because data is moved in bulk and log records are no longer required for insert and delete operations.

The latter method is particularly beneficial if, in the past, large active log space and storage requirements forced you to break a single data redistribution operation into multiple smaller redistribution tasks, which might have resulted in even more time required to complete the end-to-end data redistribution operation.

The not roll-forward recoverable redistribution method is the best practice in most situations because the data redistribution takes less time, is less error prone, and consumes fewer system resources. As a result,

the total cost of performing data redistribution is reduced, which frees up time and resources for other business operations.

Minimally logged, not roll-forward recoverable redistribution

When the **REDISTRIBUTE DATABASE PARTITION GROUP** command is issued and the **NOT ROLLFORWARD RECOVERABLE** parameter is specified, a minimal logging strategy is used that minimizes the writing of log records for each moved row. This type of logging is important for the usability of the redistribute operation since an approach that fully logs all data movement could, for large systems, require an impractical amount of active and permanent log space and would generally have poorer performance characteristics.

There are also features and optional parameters that are only available when you choose the not roll-forward recoverable redistribution method. For example, by default this method of redistribution quiesces the database and performs a precheck to ensure that prerequisites are met. You can also optionally specify to rebuild indexes and collect table statistics as part of the redistribution operation. The combination and automation of these otherwise manual tasks makes them less error prone, faster, and more efficient, while providing you with more control over the operations.

The not roll-forward recoverable redistribution method automatically reorganizes the tables, which can free up disk space. This table reorganization comes at no additional performance cost to the redistribute operation. For tables with clustering indexes, the reorganization does not attempt to maintain clustering. If perfect clustering is desired, it will be necessary to perform a **REORG TABLE** command on tables with a clustering index after data redistribution completes. For multi-dimensional-clustered (MDC) tables, the reorganization maintains the clustering of the table and frees unused blocks for reuse; however the total size of the table after redistribution appears unchanged.

Note: It is critical that you back up each affected table space or the entire database when the redistribute operation is complete because rolling forward through this type of redistribute operation results in all tables that were redistributed being marked invalid. Such tables can only be dropped, which means there is no way to recover the data in these tables. This is why, for recoverable databases, the **REDISTRIBUTE DATABASE PARTITION GROUP** utility when issued with the **NOT ROLLFORWARD RECOVERABLE** option puts all table spaces it touches into the BACKUP PENDING state. This state forces you to back up all redistributed table spaces at the end of a successful redistribute operation. With a backup taken after the redistribution operation, you should not have a need to roll-forward through the redistribute operation itself.

There is one important consequence of the lack of roll-forward recoverability: If you choose to allow updates to be made against tables in the database (even tables outside the database partition group being redistributed) while the redistribute operation is running, including the period at the end of redistribute where the table spaces touched by redistribute are being backed up, such updates can be lost in the event of a serious failure, for example, a database container is destroyed. The reason that such updates can be lost is that the redistribute operation is not roll-forward recoverable. If it is necessary to restore the database from a backup taken before the redistribution operation, then it will not be possible to roll-forward through the logs in order to replay the updates that were made during the redistribution operation without also rolling forward through the redistribution which, as was described previously, leaves the redistributed tables in the UNAVAILABLE state. Thus, the only thing that can be done in this situation is to restore the database from the backup taken before the redistribution without rolling forward. Then the redistribute operation can be performed again. Unfortunately, all the updates that occurred during the original redistribute operation are lost.

The importance of this point cannot be overemphasized. In order to be certain that there will be no lost updates during a redistribution operation, one of the following must be true:

- You must avoid making updates during the operation of the **REDISTRIBUTE DATABASE PARTITION GROUP** command, including the period after the command finishes where the affected table spaces are being backed up.
- The redistribution operation is performed with the **QUIESCE DATABASE** command parameter set to YES. You must still ensure that any applications or users that are allowed to access the quiesced database are not making updates.

- Updates that are applied during the redistribute operation come from a repeatable source, meaning that they can be applied again at any time. For example, if the source of updates is data that is stored in a file and the updates are applied during batch processing, then clearly even in the event of a failure requiring a database restore, the updates would not be lost since they could simply be applied again at any time.

With respect to allowing updates to the database during the redistribution operation, you must decide whether such updates are appropriate or not based on whether the updates can be repeated after a database restore, if necessary.

Note: Not every failure during operation of the **REDISTRIBUTE DATABASE PARTITION GROUP** command results in this problem. In fact, most do not. The **REDISTRIBUTE DATABASE PARTITION GROUP** command is fully restartable, meaning that if the utility fails in the middle of its work, it can be easily continued or aborted with the **CONTINUE** or **ABORT** options. The failures mentioned previously are failures that require the user to restore from the backup taken before the redistribute operation.

Logged, recoverable redistribution

The original and default version of the **REDISTRIBUTE DATABASE PARTITION GROUP** command, this method redistributes data by using standard SQL inserts and deletes. Extensive logging of all row movement is performed such that the database is recoverable by restoring it using the **RESTORE DATABASE** command then rolling forward through all changes using the **ROLLFORWARD DATABASE** command.

After the data redistribution, the source table contains empty spaces because rows were deleted and sent to new database partitions. If you want to free the empty spaces, you must reorganize the tables. To reorganize the tables, you must use a separate operation, after the redistribution is complete. To improve performance of this method, drop the indexes and re-create them after the redistribution is complete.

Prerequisites for data redistribution

Before data redistribution can be performed successfully for a set of tables within a database partition group, certain prerequisites must be met.

The following is a list of mandatory prerequisites:

- Authorization to perform data redistribution from the supported data redistribution interface of choice.
- A significant amount of time during a period of low system activity in which to perform the redistribution operation.
- All tables containing data to be redistributed as part of a data redistribution operation must be in a NORMAL state. For example, tables cannot be in LOAD PENDING state or other inaccessible load table states. To check the states of tables, establish a connection to each partition in the database partition group and issue the **LOAD QUERY** command. The output of this command contains information about the state of the table. The documentation of the **LOAD QUERY** command explains the meaning of each of the table states and how to move tables from one state to another.
- All tables within the database partition being redistributed must have been defined with a distribution key. If a new database partition is added to a single-partition system, data redistribution cannot be performed until all of the tables within the partitions have a distribution key. For tables that were created using the CREATE TABLE statement and have definitions that do not contain a distribution key, you must alter the table by using the ALTER TABLE statement to add a distribution key before redistributing the data.
- Replicated materialized query tables contained in a database partition group must be dropped before you redistribute the data. Store a copy of the materialized query table definitions so that they can be recreated after data redistribution completes.
- If a non-uniform redistribution is desired a distribution map must be created as a target distribution map to be used a parameter to the redistribute interface.
- A backup of the database must be created by using the **BACKUP DATABASE** command. This backup is not a mandatory prerequisite however it is strongly recommended that it be done.
- A connection must be established to the database from the catalog database partition.

- Adequate space must be available to rebuild all indexes either during or after the data redistribution. The **INDEXING MODE** command parameter affects when the indexes are rebuilt.
- When the **NOT ROLLFORWARD RECOVERABLE** command parameter is specified, adequate space should be available for writing status information to control files used by IBM Service for problem determination. The control files are generated in the following paths and should be manually deleted when the data redistribution operation is complete:
  - On Linux and UNIX operating systems: **diagpath**/redist/*db_name*/*db_partitiongroup_name*/*timestamp*/
  - On Windows operating systems: **diagpath**\redist*db_name**db_partitiongroup_name**timestamp*\

  You can calculate the space requirements in bytes for the control files by using the following formula:

```
(number of pages for all tables in the database partition group) * 64 bytes
+ number of LOB values in the database partition group) * 600 bytes
```

  To estimate *number of LOB values in the database partition group*, add the number of LOB columns in your tables and multiply it by the number of rows in the largest table.
- When the **NOT ROLLFORWARD RECOVERABLE** command parameter is **not** specified, adequate log file space must be available to contain the log entries associated with the INSERT and DELETE operations performed during data redistribution otherwise data redistribution will be interrupted or fail.

The **util_heap_sz** database configuration parameter is critical to the processing of data movement between database partitions - allocate as much memory as possible to **util_heap_sz** for the duration of the redistribution operation. Sufficient **sortheap** is also required if indexes are being rebuilt as part of the redistribution operation. Increase the value of **util_heap_sz** and **sortheap** database configuration parameter, as necessary, to improve redistribution performance.

Restrictions on data redistribution

Restrictions on data redistribution are important to note before proceeding with data redistribution or when troubleshooting problems related to data redistribution.

The following restrictions apply to data redistribution:

- Data redistribution on partitions where tables do not have partitioning key definitions is restricted.
- When data redistribution is in progress:
  - Starting another redistribution operation on the same database partition group is restricted.
  - Dropping the database partition group is restricted.
  - Altering the database partition group is restricted.
  - Executing an ALTER TABLE statement on any table in the database partition group is restricted.
  - Creating new indexes in the table undergoing data redistribution is restricted.
  - Dropping indexes defined on the table undergoing data redistribution is restricted.
  - Querying data in the table undergoing data redistribution is restricted.
  - Updating the table undergoing data redistribution is restricted.
- Updating tables in a database undergoing a data redistribution that was started using the **REDISTRIBUTE DATABASE PARTITION GROUP** command where the **NOT ROLLFORWARD RECOVERABLE** command parameter was specified is restricted. Although the updates can be made, if data redistribution is interrupted the changes made to the data might be lost and so this practice is strongly discouraged.
- When the **REDISTRIBUTE DATABASE PARTITION GROUP** command is issued and the **NOT ROLLFORWARD RECOVERABLE** command parameter is specified:
  - Data distribution changes that occur during the redistribution are not roll-forward recoverable.

  - If the database is recoverable, the table space is put into the BACKUP PENDING state after accessing the first table within the table space. To remove the table from this state, you must take a backup of the table space changes when the redistribution operation completes.
  - During data redistribution, the data in the tables in the database partition group being redistributed cannot be updated - the data is read-only. Tables that are actively being redistributed are inaccessible.
- For typed (hierarchy) tables, if the **REDISTRIBUTE DATABASE PARTITION GROUP** command is used and the **TABLE** parameter is specified with the value ONLY, then the table name is restricted to being the name of the root table only. Sub-table names cannot be specified.
- Data redistribution is supported for the movement of data between database partitions. For partitioned tables, however, movement of data between ranges of a data partitioned table is restricted unless both of the following are true:
  - The partitioned table has an access mode of FULL ACCESS in the SYSTABLES.ACCESS_MODE catalog table.
  - The partitioned table does not have any partitions currently being attached or detached.
- For replicated materialized query tables, if the data in a database partition group contains replicated materialized query tables, you must drop these tables before you redistribute the data. After data is redistributed, you can recreate the materialized query tables.
- For database partitions that contain multi-dimensional-clustered tables (MDCs) use of the **REDISTRIBUTE DATABASE PARTITION GROUP** command is restricted and will not proceed successfully if there are any multi-dimensional-clustered tables in the database partition group that contain rolled out blocks that are pending cleanup. These MDC tables must be cleaned up before data redistribution can be resumed or restarted.
- Dropping tables that are currently marked in the Db2 catalog views as being in the state "Redistribute in Progress" is restricted. To drop a table in this state, first run the **REDISTRIBUTE DATABASE PARTITION GROUP** command with the **ABORT** or **CONTINUE** parameters and an appropriate table list so that redistribution of the table is either completed or aborted.

Determining if data redistribution is needed

Determining the current data distribution for a database partition group or table can be helpful in determining if data redistribution is required. Details about the current data distribution can also be used to create a custom distribution map that specifies how to distribute data.

About this task

If a new database partition is added to a database partition group, or an existing database partition is dropped from a database partition group, perform data redistribution to balance data among all the database partitions.

If no database partitions have been added or dropped from a database partition group, then data redistribution is usually only indicated when there is an unequal distribution of data among the database partitions of the database partition group. Note that in some cases an unequal distribution of data can be desirable. For example, if some database partitions reside on a powerful machine, then it might be beneficial for those database partitions to contain larger volumes of data than other partitions.

Procedure

To determine if data redistribution is needed:

1. Get information about the current distribution of data among database partitions in the database partition group.

   Run the following query on the largest table (alternatively, a representative table) in the database partition group:

```
SELECT DBPARTITIONNUM(column_name), COUNT(*) FROM table_name
      GROUP BY DBPARTITIONNUM(column_name)
      ORDER BY DBPARTITIONNUM(column_name) DESC
```

Here, *column_name* is the name of the distribution key for table *table_name*.

The output of this query shows how many records from *table_name* reside on each database partition. If the distribution of data among database partitions is not as desired, then proceed to the next step.

2. Get information about the distribution of data across hash partitions.

   Run the following query with the same *column_name* and *table_name* that were used in the previous step:

```
SELECT HASHEDVALUE(column_name), COUNT(*) FROM table_name
      GROUP BY HASHEDVALUE(column_name)
      ORDER BY HASHEDVALUE(column_name) DESC
```

   The output of this query can easily be used to construct the distribution file needed when the **USING DISTFILE** parameter in the **REDISTRIBUTE DATABASE PARTITION GROUP** command is specified. Refer to the **REDISTRIBUTE DATABASE PARTITION GROUP** command reference for a description of the format of the distribution file.

3. Optional: If the data requires redistribution, you can plan to do this operation during a system maintenance opportunity.

   When the **USING DISTFILE** parameter is specified, the **REDISTRIBUTE DATABASE PARTITION GROUP** command uses the information in the file to generate a new partition map for the database partition group. This operation results in a uniform distribution of data among database partitions.

   If a uniform distribution is not desired, you can construct your own target partition map for the redistribution operation. The target partition map can be specified by using the **USING TARGETMAP** parameter in the **REDISTRIBUTE DATABASE PARTITION GROUP** command.

Results

After doing this investigation, you will know if your data is uniformly distributed or not or if data redistribution is required.

Redistributing data across database partitions by using the REDISTRIBUTE DATABASE PARTITION GROUP command

The **REDISTRIBUTE DATABASE PARTITION GROUP** command is the recommended interface for performing data redistribution.

Procedure

To redistribute data across database partitions in a database partition group:

1. Optional: Perform a backup of the database.

   See the **BACKUP DATABASE** command.

   It is strongly recommended that you create a backup copy of the database before you perform a data redistribution that is not roll-forward recoverable.

2. Connect to the database partition that contains the system catalog tables.

   See the CONNECT statement.

3. Issue the **REDISTRIBUTE DATABASE PARTITION GROUP** command.

   Note: In previous versions of the Db2 database product, this command used the **NODEGROUP** keyword instead of the **DATABASE PARTITION GROUP** keywords.

   Specify the following arguments:

   database partition group name
   You must specify the database partition group within which data is to be redistributed.

UNIFORM
OPTIONAL: Specifies that data is to be evenly distributed. **UNIFORM** is the default when no distribution-type is specified, so if no other distribution type has been specified, it is valid to omit this option.

USING DISTFILE ***distfile-name***
OPTIONAL: Specifies that a customized distribution is desired and the file path name of a distribution file that contains data that defines the desired data skew. The contents of this file is used to generate a target distribution map.

USING TARGETMAP ***targetmap-name***
OPTIONAL: Specifies that a target data redistribution map is to be used and the name of file that contains the target redistribution map.

For details, see the **REDISTRIBUTE DATABASE PARTITION GROUP** command-line utility information.

4. Allow the command to run uninterrupted.

   When the command completes, perform the following actions if the data redistribution proceeded successfully:

   - Take a backup of all table spaces in the database partition group that are in the BACKUP PENDING state. Alternatively, a full database backup can be performed.

     Note: Table spaces are only put into the BACKUP PENDING state if the database is recoverable and the **NOT ROLLFORWARD RECOVERABLE** command parameter is used in the **REDISTRIBUTE DATABASE PARTITION GROUP** command.
   - Recreate any replicated materialized query tables dropped before redistribution.
   - Execute the **RUNSTATS** command if the following conditions are met:
     – The **STATISTICS NONE** command parameter was specified in the **REDISTRIBUTE DATABASE PARTITION GROUP** command, or the **NOT ROLLFORWARD RECOVERABLE** command parameter was omitted. Both of these conditions mean that the statistics were not collected during data redistribution.
     – There are tables in the database partition group possessing a statistics profile.

     The **RUNSTATS** command collects data distribution statistics for the SQL compiler and optimizer to use when choosing data access plans for queries.
   - If the **NOT ROLLFORWARD RECOVERABLE** command parameter was specified, delete the control files located in the following paths :
     – On Linux and UNIX operating systems: **diagpath**/redist/*db_name*/*db_partitiongroup_name*/*timestamp*/
     – On Windows operating systems: **diagpath**\redist*db_name**db_partitiongroup_name**timestamp*\

Results

Data redistribution is complete and information about the data redistribution process is available in the redistribution log file. Information about the distribution map that was used can be found in the Db2 explain tables.

Redistributing data in a database partition group

To create an effective redistribution plan for your database partition group and redistribute your data, issue the **REDISTRIBUTE DATABASE PARTITION GROUP** command or call the sqludrdt API.

Before you begin

To work with database partition groups, you must have SYSADM, SYSCTRL, or DBADM authority.

Procedure

- To redistribute data in a database partition group:
  - Issue a **REDISTRIBUTE DATABASE PARTITION GROUP** command in the command line processor (CLP).
  - Issue the **REDISTRIBUTE DATABASE PARTITION GROUP** command by using the ADMIN_CMD procedure.
  - Call the `sqludrdt` API

Log space requirements for data redistribution

To successfully perform a data redistribution operation, adequate log file space must be allocated to ensure that data redistribution is not interrupted. Log space requirements are less of a concern when you specify the **NOT ROLLFORWARD RECOVERABLE** command parameter, since there is minimal logging during that type of data redistribution.

The quantity of log file space required depends on multiple factors including which options of the **REDISTRIBUTE DATABASE PARTITION GROUP** command are used.

When the redistribution is performed from any supported interface where the data redistribution is roll-forward recoverable:

- The log must be large enough to accommodate the INSERT and DELETE operations at each database partition where data is being redistributed. The heaviest logging requirements will be either on the database partition that will lose the most data, or on the database partition that will gain the most data.
- If you are moving to a larger number of database partitions, use the ratio of current database partitions to the new number of database partitions to estimate the number of INSERT and DELETE operations. For example, consider redistributing data that is uniformly distributed before redistribution. If you are moving from four to five database partitions, approximately twenty percent of the four original database partitions will move to the new database partition. This means that twenty percent of the DELETE operations will occur on each of the four original database partitions, and all of the INSERT operations will occur on the new database partition.
- Consider a nonuniform distribution of the data, such as the case in which the distribution key contains many NULL values. In this case, all rows that contain a NULL value in the distribution key move from one database partition under the old distribution scheme and to a different database partition under the new distribution scheme. As a result, the amount of log space required on those two database partitions increases, perhaps well beyond the amount calculated by assuming uniform distribution.
- The redistribution of each table is a single transaction. For this reason, when you estimate log space, you multiply the percentage of change, such as twenty percent, by the size of the largest table. Consider, however, that the largest table might be uniformly distributed but the second largest table, for example, might have one or more inflated database partitions. In such a case, consider using the non-uniformly distributed table instead of the largest one.

Note: After you estimate the maximum amount of data to be inserted and deleted at a database partition, double that estimate to determine the peak size of the active log. If this estimate is greater than the active log limit of 1024 GB, then the data redistribution must be done in steps. For example, use the STEPWISE_REDISTRIBUTE_DBPG procedure with a number of steps proportional to how much the estimate is greater than active log limit. You might also set the **logsecond** database configuration parameter to -1 to avoid most log space problems.

When the redistribution is performed from any supported interface where the data redistribution is not roll-forward recoverable:

- Log records are not created when rows are moved as part of data redistribution. This behavior significantly reduces log file space requirements; however, when this option is used with database roll-forward recovery, the redistribute operation log record cannot be rolled forward, and any tables processed as part of the roll-forward operation remain in UNAVAILABLE state.
- If the database partition group undergoing data redistribution contains tables with long-field (LF) or large-object (LOB) data in the tables, the number of log records generated during data redistribution will

be higher, because a log record is created for each row of data. In this case, expect the log space requirement per database partition to be roughly one third of the amount of data moving on that partition (that is, data being sent, received, or both).

Redistribution event log files

During data redistribution event logging is performed. Event information is logged to event log files which can later be used to perform error recovery.

When data redistribution is performed, information about each table that is processed is logged in a pair of event log files. The event log files are named *database-name*.*database-partition-group-name*.*timestamp*.log and *database-name*.*database-partition-group-name*.*timestamp*.

The log files are located as follows:

- The *homeinst*/sqllib/redist directory on Linux and UNIX operating systems
- The *db2instprof*\instance\redist directory on Windows operating systems, where *db2instprof* is the value of the **DB2INSTPROF** registry variable

The following is an example of the event log file names:

```
SAMPLE.IBMDEFAULTGROUP.2012012620240204
SAMPLE.IBMDEFAULTGROUP.2012012620240204.log
```

These files are for a redistribution operation on a database named SAMPLE with a database partition group named IBMDEFAULTGROUP. The files were created on January 26, 2012 at 8:24 PM local time.

The three main uses of the event log files are as follows:

- To provide general information about the redistribute operation, such as the old and new distribution maps.
- Provide users with information that helps them determine which tables have been redistributed so far by the utility.
- To provide information about each table that has been redistributed, including the indexing mode being used for the table, an indication of whether the table was successfully redistributed or not, and the starting and ending times for the redistribution operation on the table.

Redistributing database partition groups using the STEPWISE_REDISTRIBUTE_DBPG procedure

Data redistribution can be performed using built-in procedures.

Procedure

To redistribute a database partition group using the STEPWISE_REDISTRIBUTE_DBPG procedure:

1. Analyze the database partition group regarding log space availability and data skew using the ANALYZE_LOG_SPACE procedure.

   The ANALYZE_LOG_SPACE procedure returns a result set (an open cursor) of the log space analysis results, containing fields for each of the database partitions of the given database partition group.
2. Create a data distribution file for a given table using the GENERATE_DISTFILE procedure.

   The GENERATE_DISTFILE procedure generates a data distribution file for the given table and saves it using the provided file name.
3. Create and report the content of a stepwise redistribution plan for the database partition group using the STEPWISE_REDISTRIBUTE_DBPG procedure.
4. Create a data distribution file for a given table using the GET_SWRD_SETTINGS and SET_SWRD_SETTINGS procedures.

   The GET_SWRD_SETTINGS procedure reads the existing redistribute registry records for the given database partition group.

The SET_SWRD_SETTINGS procedure creates or makes changes to the redistribute registry. If the registry does not exist, it creates it and add records into it. If the registry already exists, it uses *overwriteSpec* to identify which of the field values need to be overwritten. The *overwriteSpec* field enables this function to take NULL inputs for the fields that do not need to be updated.

5. Redistribute the database partition group according to the plan using the STEPWISE_REDISTRIBUTE_DBPG procedure.

   The STEPWISE_REDISTRIBUTE_DBPG procedure redistributes part of the database partition group according to the input and the setting file.

Example

The following is an example of a CLP script on AIX:

```
# ------------------------------------------------------------------------------------
# Set the database you wish to connect to
# ------------------------------------------------------------------------------------
dbName="SAMPLE"

# ------------------------------------------------------------------------------------
# Set the target database partition group name
# ------------------------------------------------------------------------------------
dbpgName="IBMDEFAULTGROUP"

# ------------------------------------------------------------------------------------
# Specify the table name and schema
# ------------------------------------------------------------------------------------
tbSchema="$USER"
tbName="STAFF"

# ------------------------------------------------------------------------------------
# Specify the name of the data distribution file
# ------------------------------------------------------------------------------------
distFile="$HOME/sqllib/function/$dbName.IBMDEFAULTGROUP_swrdData.dst"

export DB2INSTANCE=$USER
export DB2COMM=TCPIP

# ------------------------------------------------------------------------------------
# Invoke call statements in clp
# ------------------------------------------------------------------------------------
db2start
db2 -v "connect to $dbName"

# ------------------------------------------------------------------------------------
# Analyzing the effect of adding a database partition without applying the changes - a 'what if'
# hypothetical analysis
#
# - In the following case, the hypothesis is adding database partition 40, 50 and 60 to the
#   database partition group, and for database partitions 10,20,30,40,50,60, using a respective
#   target ratio of 1:2:1:2:1:2.
#
# NOTE: in this example only partitions 10, 20 and 30 actually exist in the database
#       partition group
# ------------------------------------------------------------------------------------
db2 -v "call sysproc.analyze_log_space('$dbpgName', '$tbSchema', '$tbName', 2, ' ',
'A', '40,50,60', '10,20,30,40,50,60', '1,2,1,2,1,2')"

# ------------------------------------------------------------------------------------
# Analyzing the effect of dropping a database partition without applying the changes
#
# - In the following case, the hypothesis is dropping database partition 30 from the database
#   partition group, and redistributing the data in database partitions 10 and 20 using a
#   respective target ratio of 1 : 1
#
# NOTE: In this example all database partitions 10, 20 and 30 should exist in the database
#       partition group
# ------------------------------------------------------------------------------------
db2 -v "call sysproc.analyze_log_space('$dbpgName', '$tbSchema', '$tbName', 2, ' ',
'D', '30', '10,20','1,1')"

# ------------------------------------------------------------------------------------
# Generate a data distribution file to be used by the redistribute process
# ------------------------------------------------------------------------------------
db2 -v "call sysproc.generate_distfile('$tbSchema', '$tbName', '$distFile')"

# ------------------------------------------------------------------------------------
```

```
# Write a step wise redistribution plan into a registry
#
# Setting the 10th parameter to 1, may cause a currently running step wise redistribute
# stored procedure to complete the current step and stop, until this parameter is reset
# to 0, and the redistribute stored procedure is called again.
# ------------------------------------------------------------------------------------
db2 -v "call sysproc.set_swrd_settings('$dbpgName', 255, 0, ' ', '$distFile', 1000,
12, 2, 1, 0, '10,20,30', '50,50,50')"

# ------------------------------------------------------------------------------------
# Report the content of the step wise redistribution plan for the given database
# partition group.
# ------------------------------------------------------------------------------------
db2 -v "call sysproc.get_swrd_settings('$dbpgName', 255, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)"

# ------------------------------------------------------------------------------------
# Redistribute the database partition group "dbpgName" according to the redistribution
# plan stored in the registry by set_swrd_settings. It starting with step 3 and
# redistributes the data until 2 steps in the redistribution plan are completed.
# ------------------------------------------------------------------------------------
db2 -v "call sysproc.stepwise_redistribute_dbpg('$dbpgName', 3, 2)"
```

Configuring self-tuning memory

Self-tuning memory in partitioned database environments

When using the self-tuning memory feature in partitioned database environments, there are a few factors that determine whether the feature will tune the system appropriately.

When self-tuning memory is enabled for partitioned databases, a single database partition is designated as the tuning partition, and all memory tuning decisions are based on the memory and workload characteristics of that database partition. After tuning decisions on that partition are made, the memory adjustments are distributed to the other database partitions to ensure that all database partitions maintain similar configurations.

The single tuning partition model assumes that the feature will be used only when all of the database partitions have similar memory requirements. Use the following guidelines when determining whether to enable self-tuning memory on your partitioned database.

Cases where self-tuning memory for partitioned databases is recommended

When all database partitions have similar memory requirements and are running on similar hardware, self-tuning memory can be enabled without any modifications. These types of environments share the following characteristics:

- All database partitions are on identical hardware, and there is an even distribution of multiple logical database partitions to multiple physical database partitions
- There is a perfect or near-perfect distribution of data
- Workloads are distributed evenly across database partitions, meaning that no database partition has higher memory requirements for one or more heaps than any of the others

In such an environment, if all database partitions are configured equally, self-tuning memory will properly configure the system.

Cases where self-tuning memory for partitioned databases is recommended with qualification

In cases where most of the database partitions in an environment have similar memory requirements and are running on similar hardware, it is possible to use self-tuning memory as long as some care is taken with the initial configuration. These systems might have one set of database partitions for data, and a much smaller set of coordinator partitions and catalog partitions. In such environments, it can be beneficial to configure the coordinator partitions and catalog partitions differently than the database partitions that contain data.

Self-tuning memory should be enabled on all of the database partitions that contain data, and one of these database partitions should be designated as the tuning partition. And because the coordinator and

catalog partitions might be configured differently, self-tuning memory should be disabled on those partitions. To disable self-tuning memory on the coordinator and catalog partitions, set the **self_tuning_mem** database configuration parameter on these partitions to OFF.

Cases where self-tuning memory for partitioned databases is not recommended

If the memory requirements of each database partition are different, or if different database partitions are running on significantly different hardware, it is good practice to disable the self-tuning memory feature. You can disable the feature by setting the **self_tuning_mem** database configuration parameter to OFF on all partitions.

Comparing the memory requirements of different database partitions

The best way to determine whether the memory requirements of different database partitions are sufficiently similar is to consult the snapshot monitor. If the following snapshot elements are similar on all database partitions (differing by no more than 20%), the memory requirements of the database partitions can be considered sufficiently similar.

Collect the following data by issuing the command: `get snapshot for database on <dbname>`

```
Locks held currently                        = 0
Lock waits                                  = 0
Time database waited on locks (ms)          = 0
Lock list memory in use (Bytes)             = 4968
Lock escalations                            = 0
Exclusive lock escalations                  = 0

Total Shared Sort heap allocated            = 0
Shared Sort heap high water mark            = 0
Post threshold sorts (shared memory)        = 0
Sort overflows                              = 0

Package cache lookups                       = 13
Package cache inserts                       = 1
Package cache overflows                     = 0
Package cache high water mark (Bytes)       = 655360

Number of hash joins                        = 0
Number of hash loops                        = 0
Number of hash join overflows               = 0
Number of small hash join overflows         = 0
Post threshold hash joins (shared memory)   = 0

Number of OLAP functions                    = 0
Number of OLAP function overflows           = 0
Active OLAP functions                       = 0
```

Collect the following data by issuing the command: `get snapshot for bufferpools on <dbname>`

```
Buffer pool data logical reads              = 0
Buffer pool data physical reads             = 0
Buffer pool index logical reads             = 0
Buffer pool index physical reads            = 0
Total buffer pool read time (milliseconds) = 0
Total buffer pool write time (milliseconds)= 0
```

Using self-tuning memory in partitioned database environments

When self-tuning memory is enabled in partitioned database environments, there is a single database partition (known as the *tuning partition*) that monitors the memory configuration and propagates any configuration changes to all other database partitions to maintain a consistent configuration across all the participating database partitions.

The tuning partition is selected on the basis of several characteristics, such as the number of database partitions in the partition group and the number of buffer pools.

- To determine which database partition is currently specified as the tuning partition, call the **ADMIN_CMD** procedure as follows:

```
CALL SYSPROC.ADMIN_CMD('get stmm tuning dbpartitionnum')
```

- To change the tuning partition, call the **ADMIN_CMD** procedure as follows:

```
CALL SYSPROC.ADMIN_CMD('update stmm tuning dbpartitionnum <partitionnum>')
```

  The tuning partition is updated asynchronously or at the next database startup. To have the memory tuner automatically select the tuning partition, enter -1 for the *partitionnum* value.

Starting the memory tuner in partitioned database environments

In a partitioned database environment, the memory tuner will start only if the database is activated by an explicit **ACTIVATE DATABASE** command, because self-tuning memory requires that all partitions be active.

Disabling self-tuning memory for a specific database partition

- To disable self-tuning memory for a subset of database partitions, set the **self_tuning_mem** database configuration parameter to OFF for those database partitions.
- To disable self-tuning memory for a subset of the memory consumers that are controlled by configuration parameters on a specific database partition, set the value of the relevant configuration parameter or the buffer pool size to MANUAL or to some specific value on that database partition. It is recommended that self-tuning memory configuration parameter values be consistent across all running partitions.
- To disable self-tuning memory for a particular buffer pool on a specific database partition, issue the ALTER BUFFERPOOL statement, specifying a size value and the partition on which self-tuning memory is to be disabled.

  An ALTER BUFFERPOOL statement that specifies the size of a buffer pool on a particular database partition will create an exception entry (or update an existing entry) for that buffer pool in the SYSCAT.BUFFERPOOLDBPARTITIONS catalog view. If an exception entry for a buffer pool exists, that buffer pool will not participate in self-tuning operations when the default buffer pool size is set to AUTOMATIC. To remove an exception entry so that a buffer pool can be enabled for self tuning:

  1. Disable self tuning for this buffer pool by issuing an ALTER BUFFERPOOL statement, setting the buffer pool size to a specific value.
  2. Issue another ALTER BUFFERPOOL statement to set the size of the buffer pool on this database partition to the size value specified in Step 1.
  3. Enable self tuning for this buffer pool by issuing another ALTER BUFFERPOOL statement, setting the buffer pool size to AUTOMATIC.

Enabling self-tuning memory in nonuniform environments

Ideally, data should be distributed evenly across all database partitions, and the workload that is run on each partition should have similar memory requirements. If the data distribution is skewed, so that one or more of your database partitions contain significantly more or less data than other database partitions, these anomalous database partitions should not be enabled for self tuning. The same is true if the memory requirements are skewed across the database partitions, which can happen, for example, if resource-intensive sorts are only performed on one partition, or if some database partitions are associated with different hardware and more available memory than others. Self tuning memory can still be enabled on some database partitions in this type of environment. To take advantage of self-tuning memory in environments with skew, identify a set of database partitions that have similar data and memory requirements and enable them for self tuning. Memory in the remaining partitions should be configured manually.

DB2 configuration parameters and variables

Configuring databases across multiple partitions

The database manager provides a single view of all database configuration elements across multiple partitions. This means that you can update or reset a database configuration across all database partitions without invoking the **db2_all** command against each database partition.

You can update a database configuration across partitions by issuing only one SQL statement or only one administration command from any partition on which the database resides. By default, the method of updating or resetting a database configuration is *on all database partitions*.

For backward compatibility of command scripts and applications, you have three options:

- Use the **db2set** command to set the **DB2_UPDDBCFG_SINGLE_DBPARTITION** registry variable to TRUE, as follows:

  ```
  DB2_UPDDBCFG_SINGLE_DBPARTITION=TRUE
  ```

  Note: Setting the registry variable does not apply to **UPDATE DATABASE CONFIGURATION** or **RESET DATABASE CONFIGURATION** requests that you make using the ADMIN_CMD procedure.
- Use the **DBPARTITIONNUM** parameter with either the **UPDATE DATABASE CONFIGURATION** or the **RESET DATABASE CONFIGURATION** command or with the ADMIN_CMD procedure. For example, to update the database configurations on all database partitions, call the ADMIN_CMD procedure as follows:

  ```
  CALL SYSPROC.ADMIN_CMD
    ('UPDATE DB CFG USING sortheap 1000')
  ```

  To update a single database partition, call the ADMIN_CMD procedure as follows:

  ```
  CALL SYSPROC.ADMIN_CMD
    ('UPDATE DB CFG DBPARTITIONNUM 10 USING sortheap 1000')
  ```
- Use the **DBPARTITIONNUM** parameter with the db2CfgSet API. The flags in the **db2Cfg** structure indicate whether the value for the database configuration is to be applied to a single database partition. If you set a flag, you must also provide the **DBPARTITIONNUM** value, for example:

  ```
  #define db2CfgSingleDbpartition           256
  ```

  If you do not set the db2CfgSingleDbpartition value, the value for the database configuration applies to all database partitions unless you set the **DB2_UPDDBCFG_SINGLE_DBPARTITION** registry variable to TRUE or you set *versionNumber* to anything that is less than the version number for Version 9.5, for the db2CfgSet API that sets the database manager or database configuration parameters.

When upgrading your databases to Version 9.7, existing database configuration parameters, as a general rule, retain their values after database upgrade. However, new parameters are added using their default values and some existing parameters are set to their new Version 9.7 default values. For more details about the changes to existing database configuration parameters, see "Db2 server behavior changes" in *Upgrading to Db2 Version 10.5*. Any subsequent update or reset database configuration requests for the upgraded databases will apply to all database partitions by default.

For existing update or reset command scripts, the same rules mentioned previously apply to all database partitions. You can modify your scripts to include the **DBPARTITIONNUM** option of the **UPDATE DATABASE CONFIGURATION** or **RESET DATABASE CONFIGURATION** command, or you can set the **DB2_UPDDBCFG_SINGLE_DBPARTITION** registry variable.

For existing applications that call the **db2CfgSet** API, you must use the instructions for Version 9.5 or later. If you want the pre-Version 9.5 behavior, you can set the **DB2_UPDDBCFG_SINGLE_DBPARTITION** registry variable or modify your applications to call the API with the Version 9.5 or later version number, including the new db2CfgSingleDbpartition flag and the new **dbpartitionnum** field to update or reset database configurations for a specific database partition.

Note: If you find that database configuration values are inconsistent, you can update or reset each database partition individually.

Partitioned database environment variables

You use partitioned database environment variables to control the default behavior of a partitioned database environment, including authorization, failover, and network behavior.

DB2_ASYNC_DPF_INDOUBT_RESOLUTION

- Operating system: All
- Default=ON, Values: ON or OFF
- When set to OFF, Db2 will make one attempt to resolve each indoubt transaction as part of crash recovery. If the indoubt transactions are not resolved, no further attempt will be made by Db2 until the next database restart. When set to ON, Db2 will continue to attempt to resolve indoubt transactions asynchronously until all are resolved.

DB2CHGPWD_EEE

- Operating system: Db2 ESE on AIX, Linux, and Windows
- Default=NULL, Values: YES or NO
- This variable specifies whether you allow other users to change passwords on AIX or Windows ESE systems. You must ensure that the passwords for all database partitions or nodes are maintained centrally using either a Windows domain controller on Windows, or LDAP on AIX. If not maintained centrally, passwords may not be consistent across all database partitions or nodes. This could result in a password being changed only at the database partition to which the user connects to make the change.

DB2_FCM_SETTINGS

- Operating system: Linux
- Default=YES, Values:
  - FCM_MAXIMIZE_SET_SIZE:[YES|TRUE|NO|FALSE]. The default value for FCM_MAXIMIZE_SET_SIZE is YES.
  - FCM_CFG_BASE_AS_FLOOR: [YES|TRUE|NO|FALSE]. The default value for FCM_CFG_BASE_AS_FLOOR is NO.
- You can set the **DB2_FCM_SETTINGS** registry variable with the FCM_MAXIMIZE_SET_SIZE token to preallocate a default 4 GB of space for the fast communication manager (FCM) buffer. The token must have a value of either YES or TRUE to enable this feature.

  You can use the **DB2_FCM_SETTINGS** registry variable with the FCM_CFG_BASE_AS_FLOOR option to set the base value as the floor for the *fcm_num_buffers* and *fcm_num_channels* database manager configuration parameters. When the FCM_CFG_BASE_AS_FLOOR option is set to YES or TRUE, and these parameters are set to AUTOMATIC and have an initial or starting value, Db2 will not tune them below this value.

DB2_FORCE_OFFLINE_ADD_PARTITION

- Operating system: All
- Default=FALSE, Values: FALSE or TRUE
- This variable allows you to specify that add database partition server operations are to be performed offline. The default setting of FALSE indicates that Db2 database partition servers can be added without taking the database offline. However, if you want the operation to be performed offline or if some limitation prevents you from adding database partition servers when the database is online, set **DB2_FORCE_OFFLINE_ADD_PARTITION** to TRUE. When this variable is set to TRUE, new Db2 database partition servers are added according to the Version 9.5 and earlier versions' behavior; that is, new database partition servers are not visible to the instance until it has been shut down and restarted.

DB2_NUM_FAILOVER_NODES

- Operating system: All
- Default=2, Values: 0 to the required number of database partitions
- Set **DB2_NUM_FAILOVER_NODES** to specify the number of additional database partitions that might need to be started on a machine in the event of failover.

  In a Db2 database high availability solution, if a database server fails, the database partitions on the failed machine can be restarted on another machine. The fast communication manager (FCM) uses **DB2_NUM_FAILOVER_NODES** to calculate how much memory to reserve on each machine to facilitate this failover.

  For example, consider the following configuration:

  – Machine A has two database partitions: 1 and 2.
  – Machine B has two database partitions: 3 and 4.
  – **DB2_NUM_FAILOVER_NODES** is set to 2 on both A and B.

  At START DBM, FCM will reserve enough memory on both A and B to manage up to four database partitions so that if one machine fails, the two database partitions on the failed machine can be restarted on the other machine. If machine A fails, database partitions 1 and 2 can be restarted on machine B. If machine B fails, database partitions 3 and 4 can be restarted on machine A.

DB2_PARTITIONEDLOAD_DEFAULT

- Operating system: All supported ESE platforms
- Default=YES, Values: YES or NO
- The **DB2_PARTITIONEDLOAD_DEFAULT** registry variable lets users change the default behavior of the load utility in an ESE environment when no ESE-specific load options are specified. The default value is YES, which specifies that in an ESE environment if you do not specify ESE-specific load options, loading is attempted on all database partitions on which the target table is defined. When the value is NO, loading is attempted only on the database partition to which the load utility is currently connected.

  Note: This variable is deprecated and may be removed in a later release. The LOAD command has various options that can be used to achieve the same behavior. You can achieve the same results as the NO setting for this variable by specifying the following with the **LOAD** command: `PARTITIONED DB CONFIG MODE LOAD_ONLY OUTPUT_DBPARTNUMS x`, where x is the partition number of the partition into which you want to load data.

DB2PORTRANGE

- Operating system: Windows
- Values: nnnn:nnnn
- This value is set to the TCP/IP port range used by FCM so that any additional database partitions created on another machine will also have the same port range.

DB2_DEFAULT_TABLE_DISTRIBUTION

- Operating system: All
- Default=NULL
- Values: RANDOM - If explicit `DISTRIBUTE BY` clause is missing from **CREATE TABLE** statement, create table as `DISTRIBUTE BY RANDOM`. Setting is silently ignored for tables that can't be created as `DISTRIBUTE BY RANDOM`.

DB2CHGPWD_EEE

- Operating system: Db2 ESE on AIX, Linux, and Windows
- Default=NULL, Values: YES or NO
- This variable specifies whether you allow other users to change passwords on AIX or Windows ESE systems. You must ensure that the passwords for all database partitions or nodes are maintained

centrally using either a Windows domain controller on Windows, or LDAP on AIX. If not maintained centrally, passwords may not be consistent across all database partitions or nodes. This could result in a password being changed only at the database partition to which the user connects to make the change.

DB2_FCM_SETTINGS

- Operating system: Linux
- Default=YES, Values:
  - FCM_MAXIMIZE_SET_SIZE:[YES|TRUE|NO|FALSE]. The default value for FCM_MAXIMIZE_SET_SIZE is YES.
  - FCM_CFG_BASE_AS_FLOOR: [YES|TRUE|NO|FALSE]. The default value for FCM_CFG_BASE_AS_FLOOR is NO.
- You can set the **DB2_FCM_SETTINGS** registry variable with the FCM_MAXIMIZE_SET_SIZE token to preallocate a default 4 GB of space for the fast communication manager (FCM) buffer. The token must have a value of either YES or TRUE to enable this feature.

  You can use the **DB2_FCM_SETTINGS** registry variable with the FCM_CFG_BASE_AS_FLOOR option to set the base value as the floor for the *fcm_num_buffers* and *fcm_num_channels* database manager configuration parameters. When the FCM_CFG_BASE_AS_FLOOR option is set to YES or TRUE, and these parameters are set to AUTOMATIC and have an initial or starting value, Db2 will not tune them below this value.

DB2_FORCE_OFFLINE_ADD_PARTITION

- Operating system: All
- Default=FALSE, Values: FALSE or TRUE
- This variable allows you to specify that add database partition server operations are to be performed offline. The default setting of FALSE indicates that Db2 database partition servers can be added without taking the database offline. However, if you want the operation to be performed offline or if some limitation prevents you from adding database partition servers when the database is online, set **DB2_FORCE_OFFLINE_ADD_PARTITION** to TRUE. When this variable is set to TRUE, new Db2 database partition servers are added according to the Version 9.5 and earlier versions' behavior; that is, new database partition servers are not visible to the instance until it has been shut down and restarted.

DB2_NUM_FAILOVER_NODES

- Operating system: All
- Default=2, Values: 0 to the required number of database partitions
- Set **DB2_NUM_FAILOVER_NODES** to specify the number of additional database partitions that might need to be started on a machine in the event of failover.

  In a Db2 database high availability solution, if a database server fails, the database partitions on the failed machine can be restarted on another machine. The fast communication manager (FCM) uses **DB2_NUM_FAILOVER_NODES** to calculate how much memory to reserve on each machine to facilitate this failover.

  For example, consider the following configuration:

  - Machine A has two database partitions: 1 and 2.
  - Machine B has two database partitions: 3 and 4.
  - **DB2_NUM_FAILOVER_NODES** is set to 2 on both A and B.

  At START DBM, FCM will reserve enough memory on both A and B to manage up to four database partitions so that if one machine fails, the two database partitions on the failed machine can be restarted on the other machine. If machine A fails, database partitions 1 and 2 can be restarted on machine B. If machine B fails, database partitions 3 and 4 can be restarted on machine A.

DB2_PARTITIONEDLOAD_DEFAULT

- Operating system: All supported ESE platforms
- Default=YES, Values: YES or NO
- The **DB2_PARTITIONEDLOAD_DEFAULT** registry variable lets users change the default behavior of the load utility in an ESE environment when no ESE-specific load options are specified. The default value is YES, which specifies that in an ESE environment if you do not specify ESE-specific load options, loading is attempted on all database partitions on which the target table is defined. When the value is NO, loading is attempted only on the database partition to which the load utility is currently connected.

  Note: This variable is deprecated and may be removed in a later release. The LOAD command has various options that can be used to achieve the same behavior. You can achieve the same results as the NO setting for this variable by specifying the following with the **LOAD** command: `PARTITIONED DB CONFIG MODE LOAD_ONLY OUTPUT_DBPARTNUMS x`, where x is the partition number of the partition into which you want to load data.

DB2PORTRANGE

- Operating system: Windows
- Values: nnnn:nnnn
- This value is set to the TCP/IP port range used by FCM so that any additional database partitions created on another machine will also have the same port range.

DB2_DEFAULT_TABLE_DISTRIBUTION

- Operating system: All
- Default=NULL
- Values: RANDOM - If explicit `DISTRIBUTE BY` clause is missing from **CREATE TABLE** statement, create table as `DISTRIBUTE BY RANDOM`. Setting is silently ignored for tables that can't be created as `DISTRIBUTE BY RANDOM`.

Partitioned database environment configuration parameters

Communications

conn_elapse - Connection elapse time
This parameter specifies the number of seconds within which a network connection is to be established between Db2 members.

Configuration type
Database manager

Applies to
Db2 pureScale server (with more than one Db2 member)

Partitioned database server with local and remote clients

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]
`10` [`0-100`]

Unit of measure
Seconds

If the attempt to connect succeeds within the time specified by this parameter, communications are established. If it fails, another attempt is made to establish communications. If the connection is attempted the number of times specified by the **max_connretries** parameter and always times out, an error is issued.

fcm_num_buffers - Number of FCM buffers
You can use this parameter to specify the number of 4KB buffers that are used for internal communications, referred to as messages, both among and within database servers.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server or Db2 pureScale database server with local and remote clients

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]

32-bit platforms
AUTOMATIC [895 - 65300]

64-bit platforms
AUTOMATIC [895 - 524288]

- Database server with local and remote clients: 1024
- Database with local clients: 895
- Partitioned database server or Db2 pureScale database server with local and remote clients: 4096

Fast communication manager (FCM) buffers are used for both inter-member and intra-member communications by default.

Important: The default value of the **fcm_num_buffers** parameter is subject to change by the Db2 Configuration Advisor after initial database creation.

You can set both an initial value and the AUTOMATIC value for the **fcm_num_buffers** configuration parameter. When you set the parameter to AUTOMATIC, FCM monitors resource usage and can increase or decrease resources if they are not used within 30 minutes. The amount that resources are increased or decreased depends on the operating system. On Linux operating systems, the number of buffers can be increased only 25% more than the starting value. If the database manager attempts to start an instance and cannot allocate the specified number of buffers, it decreases the number until it can start the instance.

If you want to set the **fcm_num_buffers** parameter to both a specific value and AUTOMATIC, and you do not want the system controller thread to adjust resources lower than the specified value, set the FCM_CFG_BASE_AS_FLOOR option of the **DB2_FCM_SETTINGS** registry variable to YES or TRUE. The **DB2_FCM_SETTINGS** registry variable value is adjusted dynamically.

If you are using multiple logical nodes, one pool of **fcm_num_buffers** buffers is shared by all the logical nodes on the same machine. You can determine the size of the pool by multiplying the value of the **fcm_num_buffers** parameter by the number of logical nodes on the physical machine. Examine the value that you are using; consider how many FCM buffers are allocated on a machine or machines with multiple logical nodes. If you have multiple logical nodes on the same machine, you might have to increase the value of the **fcm_num_buffers** parameter. The number of users on the system, the number of database partition servers on the system, or the complexity of the applications can cause a system to run out of message buffers.

fcm_num_channels - Number of FCM channels
This parameter specifies the number of FCM channels for each database partition.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server orDb2 pureScale database server with local and remote clients
- Satellite database server with local clients

Parameter type
Configurable online

Propagation class
Immediate

Default [range]

UNIX 32-bit platforms
`Automatic`, with a starting value of 256, 512 or 2048 [128 - 120000]

UNIX 64-bit platforms
`Automatic`, with a starting value of 256, 512 or 2048 [128 - 524288]

Windows 32-bit
`Automatic`, with a starting value `10000` [128 - 120000]

Windows 64-bit
`Automatic`, with a starting value of 256, 512 or 2048 [128 - 524288]

The default starting values for different types of servers are as follows:

- For database server with local and remote clients, the starting value is 512.
- For database server with local clients, the starting value is 256.
- For partitioned database environment servers with local and remote clients, the starting value is 2048.

Fast communication manager (FCM) buffers are used for both inter-member and intra-member communications by default. To enable non-clustered database systems to use the FCM subsystem and the **fcm_num_channels** parameter, you had to set the **intra_parallel** parameter to YES

An FCM channel represents a logical communication end point between EDUs running in the Db2 engine. Both control flows (request and reply) and data flows (table queue data) rely on channels to transfer data between members.

When set to AUTOMATIC, FCM monitors channel usage, incrementally allocating and releasing resources as requirements change.

max_connretries - Node connection retries
This parameter specifies the maximum number of times an attempt will be made to establish a network connection between two Db2 members.

Configuration type
Database manager

Applies to
Partitioned database server with local and remote clients

Db2 pureScale server

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]
5 [0-100]

If the attempt to establish communication between two Db2 members fails (for example, the value specified by the **conn_elapse** parameter is reached), **max_connretries** specifies the number of

connection retries that can be made to a Db2 member. If the value specified for this parameter is exceeded, an error is returned.

max_time_diff - Maximum time difference between members
This parameter specifies the maximum time difference that is permitted between members in a Db2 pureScale environment that are listed in the node configuration file.

Configuration type
Database manager

Applies to
Members with local and remote clients

Parameter type
Configurable

Default [range]

In Db2 pureScale environments
1 [1 - 1 440]

Outside of Db2 pureScale environments
60 [1 - 1 440]

Unit of measure
Minutes

Each member has its own system clock. The time difference between two or more member system clocks is checked periodically. If the time difference between the system clocks is more than the amount specified by the **max_time_diff** parameter, warnings are logged in the db2diag log files.

In a Db2 pureScale environment, to ensure that members do not drift out of sync with each other, a Network Time Protocol (NTP) setup is required and periodically verified on each member. If chronyd or ntpd is not detected, warnings are logged in the db2diag log files.

The SQL1473N error message is returned in partitioned database environments where the system clock is compared to the virtual time stamp (VTS) saved in the SQLOGCTL.LFH log control file. If the time stamp in the .LFH log control file is less than the system time, the time in the database log is set to the VTS until the system clock matches the VTS.

Db2 database manager uses Coordinated Universal Time (UTC), so different time zones are not a consideration when you set the **max_time_diff** parameter. UTC is the same as Greenwich Mean Time.

start_stop_time - Start and stop timeout
This parameter specifies the time, in minutes, within which all database partition servers must respond to a **START DBM** or a **STOP DBM** command. It is also used as the timeout value during **ADD DBPARTITIONNUM** and **DROP DBPARTITIONNUM** operations.

Configuration type
Database manager

Applies to
Database server with local and remote clients

Parameter type
Configurable Online

Propagation class
Immediate

Default [range]
10 [1 - 1 440]

Unit of measure
Minutes

Member or nodes that do not respond to **db2start** or **db2stop** commands within the specified time will be killed and cleaned up automatically by **db2start** or **db2stop** in a multi member/node instance. The

diagnostic messages are logged into the **diagpath** defined in the database manager configuration or at its default value (for example, `sqllib/db2dump/ $m` on UNIX operating systems).

If a **db2start** or **db2stop** operation is not completed within the value specified by the **start_stop_time** database manager configuration parameter, the database members and partitions being stopped will be killed and cleaned up automatically. Environments with a low value for **start_stop_time** may experience this behavior. To resolve this behavior, increase the value of **start_stop_time**.

When adding a new database partition using one of the **db2start**, **START DATABASE MANAGER**, or **ADD DBPARTITIONNUM** commands, the add database partition operation must determine whether or not each database in the instance is enabled for automatic storage. This is done by communicating with the catalog partition for each database. If automatic storage is enabled, the storage path definitions are retrieved as part of that communication. Likewise, if system temporary table spaces are to be created with the database partition(s), the operation might have to communicate with another database partition server to retrieve the table space definitions for the database partition(s) that reside on that server. These factors should be considered when determining the value of the **start_stop_time** parameter.

To allow the force application operation more time to succeed, for example to rollback a large batch transaction, increase **start_stop_time**. If the timeout is configured to be too low or if the member/partition(s) is in a state that cannot be successfully forced, the member/partition(s) will be killed and possibly in need of crash recovery. **db2start** needs to be run before the crash recovery can occur and release any retained locks, then **db2stop** to stop the member/partition(s) cleanly.

Note: On UNIX operating systems, the **start_stop_time** configuration parameter on multi-member Db2 instances only includes the time required to stop any particular member locally; it does not include the time required to send the stop request to remote members through `rsh` or `ssh`.

Parallel processing

intra_parallel - Enable intrapartition parallelism
This parameter specifies whether or not database connections will use intrapartition query parallelism by default.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable

Default [range]
`NO (0)` [`SYSTEM (-1)`, `NO (0)`, `YES (1)`]

A value of `YES` enables intrapartition query parallelism. A value of `NO` disables intrapartition query parallelism.

A value of `SYSTEM` causes the parameter value to be set to `YES` or `NO` based on the hardware on which the database manager is running. If the number of logical CPUs on the system is > 1, when the value is set to `SYSTEM`, intrapartition query parallelism is enabled.

Note: The default value is subject to change by the Db2 Configuration Advisor after initial database creation.

Note:

- Parallel index creation does not use this configuration parameter.
- If you change this parameter value, packages might be rebound to the database, and some performance degradation might occur.

- The **intra_parallel** setting can be overridden in an application by a call to the ADMIN_SET_INTRA_PARALLEL procedure. Both the **intra_parallel** setting and the value set in an application by the ADMIN_SET_INTRA_PARALLEL procedure can be overridden in a workload by setting the MAXIMUM DEGREE attribute in a workload definition.

max_querydegree - Maximum query degree of parallelism
This parameter specifies the maximum degree of intrapartition parallelism that is used for any SQL statement executing on this instance of the database manager. An SQL statement will not use more than this number of parallel operations within a database partition when the statement is executed.

Configuration type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter type
Configurable Online

Propagation class
Statement boundary

Default [range]
-1 (ANY) [ANY, 1 - 32 767] (ANY means system-determined)

Note: The default value is subject to change by the Db2 Configuration Advisor after initial database creation.

The **intra_parallel** configuration parameter must be set to YES to enable the database partition to use intrapartition parallelism for SQL statements. The **intra_parallel** parameter is no longer required for parallel index creation.

The default value for this configuration parameter is -1. This value means that the system uses the degree of parallelism determined by the optimizer; otherwise, the user-specified value is used.

Note: The degree of parallelism for an SQL statement can be specified at statement compilation time using the CURRENT DEGREE special register or the **DEGREE** bind option.

The maximum query degree of parallelism for an active application can be modified using the **SET RUNTIME DEGREE** command. The actual runtime degree used is the lower of:

- **max_querydegree** configuration parameter
- Application runtime degree
- SQL statement compilation degree
- MAXIMUM DEGREE service class option
- MAXIMUM DEGREE workload option

This configuration parameter applies to queries only.

Chapter 5. Administrative APIs, commands, SQL statements

Administrative APIs

sqleaddn - Add a database partition to the partitioned database environment

Adds a database partition to a database partition server.

Scope

This API only affects the database partition server on which it is executed.

Authorization

One of the following authorities:

- SYSADM
- SYSCTRL

Required connection

None

API include file

```
sqlenv.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  sqleaddn (
        void * pAddNodeOptions,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgaddn (
        unsigned short addnOptionsLen,
        struct sqlca * pSqlca,
        void * pAddNodeOptions);
```

sqleaddn API parameters

pAddNodeOptions
Input. A pointer to the optional `sqle_addn_options` structure. This structure is used to specify the source database partition server, if any, of the system temporary table space definitions for all database partitions to be created. If not specified (that is, a NULL pointer is specified), the system temporary table space definitions will be the same as those for the catalog partition.

pSqlca
Output. A pointer to the `sqlca` structure.

sqlgaddn API-specific parameters

addnOptionsLen
Input. A 2-byte unsigned integer representing the length of the optional `sqle_addn_options` structure in bytes.

Usage notes

This API should only be used if a database partition server is added to an environment that has one database and that database is not cataloged at the time of the add partition operation. In this situation, because the database is not cataloged, the add partition operation does not recognize the database, and does not create a database partition for the database on the new database partition server. Any attempt to connect to the database partition on the new database partition server results in an error. The database must first be cataloged before the `sqleaddn` API can be used to create the database partition for the database on the new database partition server.

This API should not be used if the environment has more than one database and at least one of the databases is cataloged at the time of the add partition operation. In this situation, use the `sqlecran` API to create a database partition for each database that was not cataloged at the time of the add partition operation. Each uncataloged database must first be cataloged before the `sqlecran` API can be used to create the database partition for the database on the new database partition server.

Before adding a new database partition, ensure that there is sufficient storage for the containers that must be created.

The add node operation creates an empty database partition on the new database partition server for every database that exists in the instance. The configuration parameters for the new database partitions are set to the default value.

Note: Any uncataloged database is not recognized when adding a new database partition. The uncataloged database will not be present on the new database partition. An attempt to connect to the database on the new database partition returns the error message SQL1013N.

If an add node operation fails while creating a database partition locally, it enters a clean-up phase, in which it locally drops all databases that have been created. This means that the database partitions are removed only from the database partition server being added (that is, the local database partition server). Existing database partitions remain unaffected on all other database partition servers. If this fails, no further clean up is done, and an error is returned.

The database partitions on the new database partition server cannot be used to contain user data until after the ALTER DATABASE PARTITION GROUP statement has been used to add the database partition server to a database partition group.

This API will fail if a create database or a drop database operation is in progress. The API can be called again when the operation has completed.

The storage groups storage path definitions are retrieved when the `sqleaddn` API has to communicate with the catalog partition for each of the databases in the instance. Likewise, if system temporary table spaces are to be created with the database partitions, the `sqleaddn` API may have to communicate with another database partition server in the partitioned database environment in order to retrieve the table space definitions. The **start_stop_time** database manager configuration parameter is used to specify the time, in minutes, by which the other database partition server must respond with the automatic storage and table space definitions. If this time is exceeded, the API fails. Increase the value of **start_stop_time**, and call the API again.

REXX API syntax

This API can be called from REXX through the SQLDB2 interface.

sqlecran - Create a database on a database partition server

Creates a database only on the database partition server that calls the API.
This API is not intended for general use. For example, it should be used with `db2Restore` if the database partition at a database partition server was damaged and must be re-created. Improper use of this API can cause inconsistencies in the system, so it should only be used with caution.

Note: If this API is used to re-create a database partition that was dropped (because it was damaged), the database at this database partition server will be in the restore-pending state. After recreating the database partition, the database must immediately be restored on this database partition server.

Scope

This API only affects the database partition server on which it is called.

Authorization

One of the following authorities:

- SYSADM
- SYSCTRL

Required connection

Instance. To create a database at another database partition server, it is necessary to first attach to that database partition server. A database connection is temporarily established by this API during processing.

API include file

```
sqlenv.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  sqlecran (
        char * pDbName,
        void * pReserved,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgcran (
        unsigned short reservedLen,
        unsigned short dbNameLen,
        struct sqlca * pSqlca,
        void * pReserved,
        char * pDbName);
```

sqlecran API parameters

pDbName
Input. A string containing the name of the database to be created. Must not be NULL.

pReserved
Input. A spare pointer that is set to null or points to zero. Reserved for future use.

pSqlca
Output. A pointer to the sqlca structure.

sqlgcran API-specific parameters

reservedLen
Input. Reserved for the length of **pReserved**.

dbNameLen
Input. A 2-byte unsigned integer representing the length of the database name in bytes.

Usage notes

When the database is successfully created, it is placed in restore-pending state. The database must be restored on this database partition server before it can be used.

REXX API syntax

This API can be called from REXX through the SQLDB2 interface.

sqledpan - Drop a database on a database partition server

Drops a database at a specified database partition server. Can only be run in a partitioned database environment.

Scope

This API only affects the database partition server on which it is called.

Authorization

One of the following authorities:

- SYSADM
- SYSCTRL

Required connection

None. An instance attachment is established for the duration of the call.

API include file

```
sqlenv.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  sqledpan (
        char * pDbAlias,
        void * pReserved,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgdpan (
        unsigned short Reserved1,
        unsigned short DbAliasLen,
        struct sqlca * pSqlca,
        void * pReserved2,
        char * pDbAlias);
```

sqledpan API parameters

pDbAlias
Input. A string containing the alias of the database to be dropped. This name is used to reference the actual database name in the system database directory.

pReserved
Reserved. Should be NULL.

pSqlca
Output. A pointer to the sqlca structure.

sqlgdpan API-specific parameters

Reserved1
Reserved for future use.

DbAliasLen
Input. A 2-byte unsigned integer representing the length in bytes of the database alias.

pReserved2
A spare pointer that is set to null or points to zero. Reserved for future use.

Usage notes

Improper use of this API can cause inconsistencies in the system, so it should only be used with caution.

REXX API syntax

This API can be called from REXX through the SQLDB2 interface.

sqledrpn - Check whether a database partition server can be dropped

Verifies whether a database partition server is being used by a database. A message is returned, indicating whether the database partition server can be dropped.

Scope

This API only affects the database partition server on which it is issued.

Authorization

One of the following authorities:

- SYSADM
- SYSCTRL

API include file

```
sqlenv.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  sqledrpn (
        unsigned short Action,
        void * pReserved,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgdrpn (
        unsigned short Reserved1,
        struct sqlca * pSqlca,
        void * pReserved2,
        unsigned short Action);
```

sqledrpn API parameters

Action
The action requested. The valid value is: `SQL_DROPNODE_VERIFY`

pReserved
Reserved. Should be `NULL`.

pSqlca
Output. A pointer to the `sqlca` structure.

sqlgdrpn API-specific parameters

Reserved1
Reserved for the length of **pReserved2**.

pReserved2
A spare pointer that is set to NULL or points to 0. Reserved for future use.

Usage notes

If a message is returned, indicating that the database partition server is not in use, use the **db2stop** command with **DROP NODENUM** to remove the entry for the database partition server from the db2nodes.cfg file, which removes the database partition server from the partitioned database environment.

If a message is returned, indicating that the database partition server is in use, the following actions should be taken:

1. The database partition server to be dropped will have a database partition on it for each database in the instance. If any of these database partitions contain data, redistribute the database partition groups that use these database partitions. Redistribute the database partition groups to move the data to database partitions that exist at database partition servers that are not being dropped.
2. After the database partition groups are redistributed, drop the database partition from every database partition group that uses it. To remove a database partition from a database partition group, you can use either the drop node option of the sqludrdt API or the ALTER DATABASE PARTITION GROUP statement.
3. Drop any event monitors that are defined on the database partition server.
4. Rerun sqledrpn to ensure that the database partition at the database partition server is no longer in use.

REXX API syntax

This API can be called from REXX through the SQLDB2 interface.

sqlugrpn - Get the database partition server number for a row

Beginning with Version 9.7, this API is deprecated. Use the db2GetRowPartNum (Get the database partition server number for a row) API to return the database partition number and database partition server number for a row. If you call the sqlugrpn API and the **DB2_PMAP_COMPATIBILITY** registry variable is set to OFF, the error message SQL2768N is returned.

Returns the database partition number and the database partition server number based on the distribution key values. An application can use this information to determine on which database partition server a specific row of a table is stored.

The partitioning data structure, sqlupi, is the input for this API. The structure can be returned by the sqlugtpi API. Another input is the character representations of the corresponding distribution key values. The output is a database partition number generated by the distribution strategy and the corresponding database partition server number from the distribution map. If the distribution map information is not provided, only the database partition number is returned. This can be useful when analyzing data distribution.

The database manager does not need to be running when this API is called.

Scope

This API must be invoked from a database partition server in the db2nodes.cfg file. This API should not be invoked from a client, since it could result in erroneous database partitioning information being returned due to differences in code page and endianess between the client and the server.

Authorization

None

API include file

```
sqlutil.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  sqlugrpn (
   unsigned short num_ptrs,
   unsigned char ** ptr_array,
   unsigned short * ptr_lens,
   unsigned short territory_ctrycode,
   unsigned short codepage,
   struct sqlupi * part_info,
   short * part_num,
   SQL_PDB_NODE_TYPE * node_num,
   unsigned short chklvl,
   struct sqlca * sqlca,
   short dataformat,
   void * pReserved1,
   void * pReserved2);

SQL_API_RC SQL_API_FN
  sqlggrpn (
   unsigned short num_ptrs,
   unsigned char ** ptr_array,
   unsigned short * ptr_lens,
   unsigned short territory_code,
   unsigned short codepage,
   struct sqlupi * part_info,
   short * part_num,
   SQL_PDB_NODE_TYPE * node_num,
   unsigned short chklvl,
   struct sqlca * sqlca,
   short dataformat,
   void * pReserved1,
   void * pReserved2);
```

sqlugrpn API parameters

num_ptrs
The number of pointers in **ptr_array**. The value must be the same as the one specified for the **part_info** parameter; that is, **part_info**->**sqld**.

ptr_array
An array of pointers that points to the character representations of the corresponding values of each part of the distribution key specified in **part_info**. If a null value is required, the corresponding pointer is set to null. For generated columns, this function does not generate values for the row. The user is responsible for providing a value that will lead to the correct partitioning of the row.

ptr_lens
An array of unsigned integers that contains the lengths of the character representations of the corresponding values of each part of the partitioning key specified in **part_info**.

territory_ctrycode
The country/region code of the target database. This value can also be obtained from the database configuration file using the **GET DATABASE CONFIGURATION** command.

codepage
The code page of the target database. This value can also be obtained from the database configuration file using the **GET DATABASE CONFIGURATION** command.

part_info
A pointer to the `sqlupi` structure.

part_num
A pointer to a 2-byte signed integer that is used to store the database partition number.

node_num
A pointer to an SQL_PDB_NODE_TYPE field used to store the node number. If the pointer is null, no node number is returned.

chklvl
An unsigned integer that specifies the level of checking that is done on input parameters. If the value specified is zero, no checking is done. If any non-zero value is specified, all input parameters are checked.

sqlca
Output. A pointer to the `sqlca` structure.

dataformat
Specifies the representation of distribution key values. Valid values are:

SQL_CHARSTRING_FORMAT
All distribution key values are represented by character strings. This is the default value.

SQL_IMPLIEDDECIMAL_FORMAT
The location of an implied decimal point is determined by the column definition. For example, if the column definition is DECIMAL(8,2), the value 12345 is processed as 123.45.

SQL_PACKEDDECIMAL_FORMAT
All decimal column distribution key values are in packed decimal format.

SQL_BINARYNUMERICS_FORMAT
All numeric distribution key values are in big-endian binary format.

pReserved1
Reserved for future use.

pReserved2
Reserved for future use.

Usage notes

Data types supported on the operating system are the same as those that can be defined as a distribution key.

Note: CHAR, VARCHAR, GRAPHIC, and VARGRAPHIC data types must be converted to the database code page before this API is called.

For numeric and datetime data types, the character representations must be at the code page of the corresponding system where the API is invoked.

If **node_num** is not null, the distribution map must be supplied; that is, **pmaplen** field in **part_info** parameter (**part_info**->**pmaplen**) is either 2 or 8192. Otherwise, SQLCODE -6038 is returned. The distribution key must be defined; that is, **sqld** field in **part_info** parameter (**part_info**->**sqld**) must be greater than zero. Otherwise, SQLCODE -2032 is returned.

If a null value is assigned to a non-nullable partitioning column, SQLCODE -6039 is returned.

All the leading blanks and trailing blanks of the input character string are stripped, except for the CHAR, VARCHAR, GRAPHIC, and VARGRAPHIC data types, where only trailing blanks are stripped.

Commands

REDISTRIBUTE DATABASE PARTITION GROUP

The **REDISTRIBUTE DATABASE PARTITION GROUP** command redistributes data across the partitions in a database partition group. This command affects all objects present in the database partition group and cannot be restricted to one object alone.

Scope

This command affects all database partitions in the database partition group.

Authorization

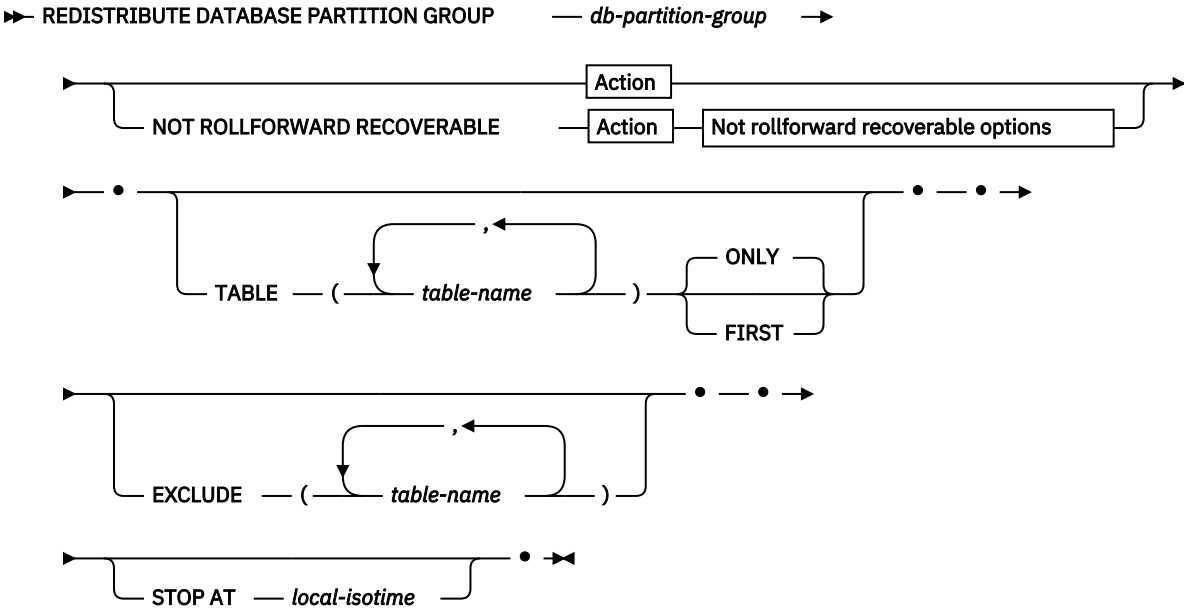
One of the following authorities is required:

- SYSADM
- SYSCTRL
- DBADM

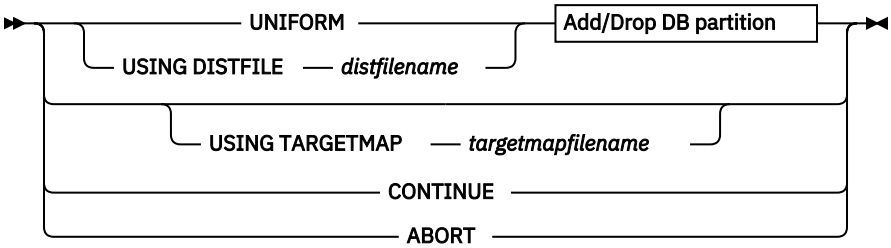
In addition, one of the following groups of authorizations is also required:

- DELETE, INSERT, and SELECT privileges on all tables in the database partition group being redistributed
- DATAACCESS authority

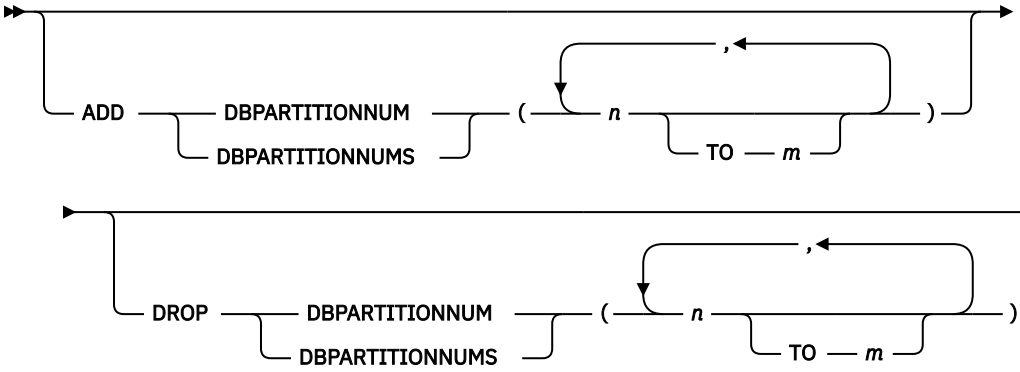
Command syntax



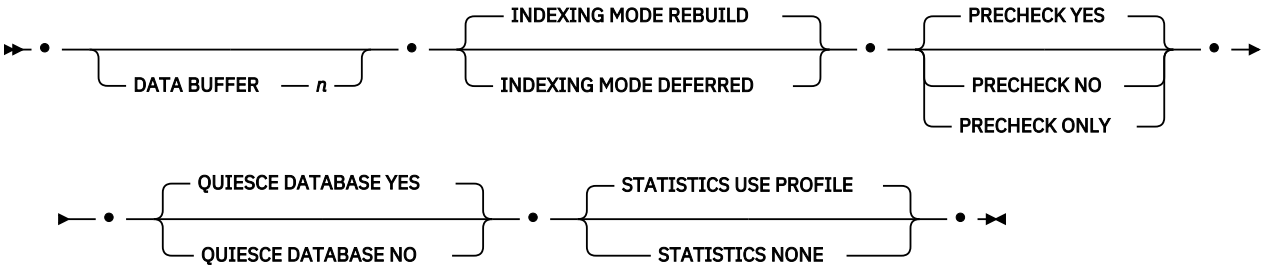
Action



Add/Drop DB partition



Not rollforward recoverable options



Command parameters

DATABASE PARTITION GROUP *db-partition-group*
The name of the database partition group. This one-part name identifies a database partition group described in the SYSCAT.DBPARTITIONGROUPS catalog table. The database partition group cannot currently be undergoing redistribution.

Note: Tables in the IBMCATGROUP and the IBMTEMPGROUP database partition groups cannot be redistributed.

NOT ROLLFORWARD RECOVERABLE
When this option is used, the **REDISTRIBUTE DATABASE PARTITION GROUP** command is not rollforward recoverable.

- Data is moved in bulk instead of by internal insert and delete operations. This reduces the number of times that a table must be scanned and accessed, which results in better performance.
- Log records are no longer required for each of the insert and delete operations. This means that you no longer need to manage large amounts of active log space and log archiving space in your system when performing data redistribution.
- When using the **REDISTRIBUTE DATABASE PARTITION GROUP** command with the **NOT ROLLFORWARD RECOVERABLE** option, the redistribute operation uses the **INDEXING MODE DEFERRED** option for tables that contain XML columns. If a table does not contain an XML column, the redistribute operation uses the indexing mode specified when issuing the command.

When this option is *not* used, extensive logging of all row movement is performed such that the database can be recovered later in the event of any interruptions, errors, or other business need.

This option is not supported for column-organized tables.

UNIFORM
Specifies that the data is uniformly distributed across hash partitions (that is, every hash partition is assumed to have the same number of rows), but the same number of hash partitions do not map to each database partition. After redistribution, all database partitions in the database partition group have approximately the same number of hash partitions.

USING DISTFILE *distfilename*
If the distribution of distribution key values is skewed, use this option to achieve a uniform redistribution of data across the database partitions of a database partition group.

Use the *distfilename* to indicate the current distribution of data across the 32 768 hash partitions.

Use row counts, byte volumes, or any other measure to indicate the amount of data represented by each hash partition. The utility reads the integer value associated with a partition as the weight of that partition. When a *distfilename* is specified, the utility generates a target distribution map that it uses to redistribute the data across the database partitions in the database partition group as uniformly as possible. After the redistribution, the weight of each database partition in the database partition group is approximately the same (the weight of a database partition is the sum of the weights of all hash partitions that map to that database partition).

For example, the input distribution file might contain entries as follows:

```
10223
1345
112000
0
100
...
```

In the example, hash partition 2 has a weight of 112000, and partition 3 (with a weight of 0) has no data mapping to it at all.

The *distfilename* should contain 32 768 positive integer values in character format. The sum of the values should be less than or equal to 4 294 967 295.

USING TARGETMAP *targetmapfilename*
The file specified in *targetmapfilename* is used as the target distribution map. Data redistribution is done according to this file.

The *targetmapfilename* should contain 32 768 integers, each representing a valid database partition number. The number on any row maps a hash value to a database partition. This means that if row *X* contains value *Y*, then every record with HASHEDVALUE() of *X* is to be located on database partition *Y*.

If a database partition, included in the target map, is not in the database partition group, an error is returned. Issue ALTER DATABASE PARTITION GROUP ADD DBPARTITIONNUM statement before running **REDISTRIBUTE DATABASE PARTITION GROUP** command.

If a database partition, excluded from the target map, *is* in the database partition group, that database partition will not be included in the partitioning. Such a database partition can be dropped using ALTER DATABASE PARTITION GROUP DROP DBPARTITIONNUM statement either before or after the **REDISTRIBUTE DATABASE PARTITION GROUP** command.

CONTINUE
Continues a previously failed or stopped **REDISTRIBUTE DATABASE PARTITION GROUP** operation. If none occurred, an error is returned.

ABORT
Aborts a previously failed or stopped **REDISTRIBUTE DATABASE PARTITION GROUP** operation. If none occurred, an error is returned.

ADD

DBPARTITIONNUM *n*

TO *m*

n or *n* **TO** *m* specifies a list or lists of database partition numbers which are to be added into the database partition group. Any specified partition must not already be defined in the database partition group (SQLSTATE 42728). This is equivalent to executing the ALTER DATABASE PARTITION GROUP statement with ADD DBPARTITIONNUM clause specified.

DBPARTITIONNUMS *n*

TO *m*

n or *n* **TO** *m* specifies a list or lists of database partition numbers which are to be added into the database partition group. Any specified partition must not already be defined in the database partition group (SQLSTATE 42728). This is equivalent to executing the ALTER DATABASE PARTITION GROUP statement with ADD DBPARTITIONNUM clause specified.

Note:

1. When a database partition is added using this option, containers for table spaces are based on the containers of the corresponding table space on the lowest numbered existing partition in the database partition group. If this would result in a naming conflict among containers, which could happen if the new partitions are on the same physical machine as existing containers, this option should not be used. Instead, the ALTER DATABASE PARTITION GROUP statement should be used with the WITHOUT TABLESPACES option before issuing the **REDISTRIBUTE DATABASE PARTITION GROUP** command. Table space containers can then be created manually specifying appropriate names.
2. Data redistribution might create table spaces for all new database partitions if the **ADD DBPARTITIONNUMS** parameter is specified.

DROP

DBPARTITIONNUM ***n***

TO ***m***

n or *n* **TO** *m* specifies a list or lists of database partition numbers which are to be dropped from the database partition group. Any specified partition must already be defined in the database partition group (SQLSTATE 42729). This is equivalent to executing the ALTER DATABASE PARTITION GROUP statement with the DROP DBPARTITIONNUM clause specified.

DBPARTITIONNUMS ***n***

TO ***m***

n or *n* **TO** *m* specifies a list or lists of database partition numbers which are to be dropped from the database partition group. Any specified partition must already be defined in the database partition group (SQLSTATE 42729). This is equivalent to executing the ALTER DATABASE PARTITION GROUP statement with the DROP DBPARTITIONNUM clause specified.

TABLE ***tablename***

Specifies a table order for redistribution processing.

ONLY

If the table order is followed by the **ONLY** keyword (which is the default), then, only the specified tables will be redistributed. The remaining tables can be later processed by **REDISTRIBUTE CONTINUE** commands. This is the default.

FIRST

If the table order is followed by the **FIRST** keyword, then, the specified tables will be redistributed with the given order and the remaining tables in the database partition group will be redistributed with random order.

EXCLUDE ***tablename***

Specifies tables to omit from redistribution processing. For example, you can temporarily omit a table until you can configure it to meet the requirements for data redistribution. The omitted tables can be later processed by **REDISTRIBUTE CONTINUE** commands.

STOP AT ***local-isotime***

When this option is specified, before beginning data redistribution for each table, the *local-isotime* is compared with the current local timestamp. If the specified *local-isotime* is equal to or earlier than the current local timestamp, the utility stops with a warning message. Data redistribution processing of tables in progress at the stop time will complete without interruption. No new data redistribution processing of tables begins. The unprocessed tables can be redistributed using the **CONTINUE** option. This *local-isotime* value is specified as a time stamp, a 7-part character string that identifies a combined date and time. The format is *yyyy-mm-dd-hh.mm.ss.nnnnnn* (year, month, day, hour, minutes, seconds, microseconds) expressed in local time.

DATA BUFFER *n*
Specifies the number of 4 KB pages to use as buffered space for transferring data within the utility. This command parameter can be used only when the **NOT ROLLFORWARD RECOVERABLE** parameter is also specified.

If the value specified is lower than the minimum supported value, the minimum value is used and no warning is returned. If a **DATA BUFFER** value is not specified, an intelligent default is calculated by the utility at runtime at the beginning of processing each table. Specifically, the default is to use 50% of the memory available in the utility heap at the time redistribution of the table begins and to take into account various table properties as well.

This memory is allocated directly from the utility heap, whose size can be modified through the **util_heap_sz** database configuration parameter. The value of the **DATA BUFFER** parameter of the **REDISTRIBUTE DATABASE PARTITION GROUP** command can temporarily exceed **util_heap_sz** if more memory is available in the system.

INDEXING MODE
Specifies how indexes are maintained during redistribution. This command parameter can be used only when the **NOT ROLLFORWARD RECOVERABLE** parameter is also specified.

Valid values are:

REBUILD
Indexes will be rebuilt from scratch. Indexes do not have to be valid to use this option. As a result of using this option, index pages will be clustered together on disk.

DEFERRED
Redistribute will not attempt to maintain any indexes. Indexes will be marked as needing a refresh. The first access to such indexes might force a rebuild, or indexes might be rebuilt when the database is restarted.

Note: For non-MDC and non-ITC tables, if there are invalid indexes on the tables, the **REDISTRIBUTE DATABASE PARTITION GROUP** command automatically rebuilds them if you do not specify **INDEXING MODE DEFERRED**. For an MDC or ITC table, even if you specify **INDEXING MODE DEFERRED**, a composite index that is invalid is rebuilt before table redistribution begins because the utility needs the composite index to process an MDC or ITC table.

PRECHECK
Verifies that the database partition group can be redistributed. This command parameter can be used only when the **NOT ROLLFORWARD RECOVERABLE** parameter is also specified.

YES
This is the default value. The redistribution operation begins only if the verification completes successfully. If the verification fails, the command terminates and returns an error message related to the first check that failed.

NO
The redistribution operation begins immediately; no verification occurs.

ONLY
The command terminates after performing the verification; no redistribution occurs. By default it will not quiesce the database. If the **QUIESCE DATABASE** command parameter was set to YES or defaulted to a value of YES, the database remains quiesced. To restore connectivity to the database, perform the redistribution operation or issue **UNQUIESCE DATABASE** command.

QUIESCE DATABASE
Specifies to force all users off the database and put it into a quiesced mode. This command parameter can be used only when the **NOT ROLLFORWARD RECOVERABLE** parameter is also specified.

YES
This is the default value. Only users with SYSADM, SYSMAINT, or SYSCTRL authority or users who have been granted QUIESCE_CONNECT authority will be able to access the database or its objects. Once the redistribution completes successfully, the database is unquiesced.

NO
The redistribution operation does not quiesce the database; no users are forced off the database.

STATISTICS
Specifies that the utility should collect statistics for the tables that have a statistics profile. This command parameter can be used only when the **NOT ROLLFORWARD RECOVERABLE** parameter is also specified.

Specifying this option is more efficient than separately issuing the **RUNSTATS** command after the data redistribution is completed.

USE PROFILE
Statistics will be collected for the tables with a statistics profile. For tables without a statistics profile, nothing will be done. This is the default.

NONE
Statistics will not be collected for tables.

Examples

Redistribute database partition group DBPG_1 by providing the current data distribution through a data distribution file, `distfile_for_dbpg_1`. Move the data onto two new database partitions, 6 and 7.

Redistribute database partition group DBPG_2 such that:

- The redistribution is not rollforward recoverable;
- Data is uniformly distributed across hash partitions;
- Indexes are rebuilt from scratch;
- Statistics are not collected;
- 180,000 4 KB pages are used as buffered space for transferring the data.

This redistribution operation also quiesces the database and performs a precheck due to the default values for the **QUIESCE DATABASE** and **PRECHECK** command parameters.

Usage notes

- Before starting a redistribute operation, ensure that the tables are in normal state and not in "load pending" state or "reorg pending" state. Table states can be checked by using the **LOAD QUERY** command.
- When the **NOT ROLLFORWARD RECOVERABLE** option is specified and the database is a recoverable database, the first time the utility accesses a table space, it is put into the BACKUP PENDING state. All the tables in that table space will become read-only until the table space is backed-up, which can only be done when all tables in the table space have finished being redistributed.
- When a redistribution operation is running, it produces an event log file containing general information about the redistribution operation and information such as the starting and ending time of each table processed. This event log file is written to:
  - The home`inst/sqllib/redist` directory on Linux and UNIX operating systems, using the following format for subdirectories and file name: *`database-name.database-partition-group-name.timestamp`*`.log`.
  - The **DB2INSTPROF**`\`*`instance`*`\redist` directory on Windows operating systems (where **DB2INSTPROF** is the value of the **DB2INSTPROF** registry variable), using the following format for subdirectories and file name: *`database-name.database-partition-group-name.timestamp`*`.log`.
  - The time stamp value is the time when the command was issued.
- This utility performs intermittent COMMITs during processing.
- All packages having a dependency on a table that has undergone redistribution are invalidated. It is recommended to explicitly rebind such packages after the redistribute database partition group operation has completed. Explicit rebinding eliminates the initial delay in the execution of the first SQL request for the invalid package. The redistribute message file contains a list of all the tables that have undergone redistribution.

- By default, the redistribute utility will update the statistics for those tables that have a statistics profile. For the tables without a statistics profile, it is recommended that you separately update the table and index statistics for these tables by calling the db2Runstats API or by issuing the **RUNSTATS** command after the redistribute operation has completed.
- Database partition groups containing replicated materialized query tables or tables defined with DATA CAPTURE CHANGES cannot be redistributed.
- Redistribution is not allowed if there are user temporary table spaces with existing declared temporary tables or created temporary tables in the database partition group.
- Options such as **INDEXING MODE** are ignored on tables, on which they do not apply, without warning. For example, **INDEXING MODE** will be ignored on tables without indexes.
- The **REDISTRIBUTE DATABASE PARTITION GROUP** command might fail (SQLSTATE 55071) if an add database partition server request is either pending or in progress. This command might also fail (SQLSTATE 55077) if a new database partition server is added online to the instance and not all applications are aware of the new database partition server.
- The REDISTRIBUTE DATABASE PARTITION GROUP command is not allowed if there is an outstanding asynchronous background process to create a column compression dictionary (SQL6056N).

Compatibilities

Tables containing XML columns that use the Db2 Version 9.5 or earlier XML record format cannot be redistributed. Use the ADMIN_MOVE_TABLE stored procedure to migrate the table to the new format.

db2nchg - Change database partition server configuration

Modifies database partition server configuration. This includes moving the database partition server from one machine to another; changing the TCP/IP host name of the machine; and selecting a different logical port number or a different network name for the database partition server.

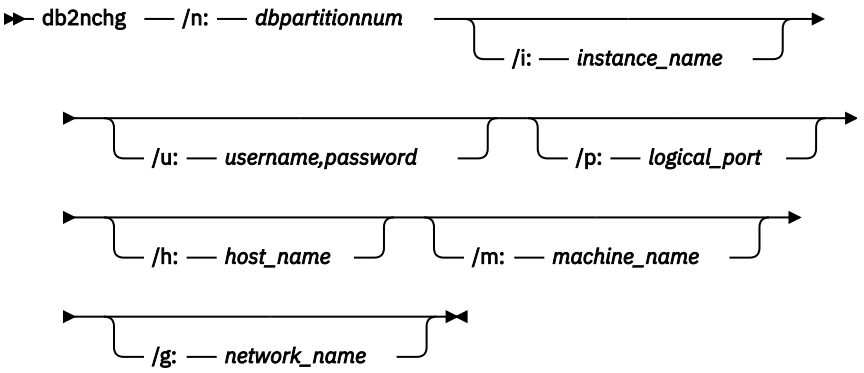
This command can only be used if the database partition server is stopped.

This command is available on Windows operating systems only.

Authorization

Local Administrator

Command syntax



Command parameters

/n:*dbpartitionnum*
Specifies the database partition number of the database partition server's configuration that is to be changed.

/i:*instance_name*
Specifies the instance in which this database partition server participates. If a parameter is not specified, the default is the current instance.

/u:*username,password*
Specifies the user name and password. If a parameter is not specified, the existing user name and password will apply.

/p:*logical_port*
Specifies the logical port for the database partition server. This parameter must be specified to move the database partition server to a different machine. If a parameter is not specified, the logical port number will remain unchanged.

/h:*host_name*
Specifies TCP/IP host name used by FCM for internal communications. If this parameter is not specified, the host name will remain the same.

/m:*machine_name*
Specifies the machine where the database partition server will reside. The database partition server can only be moved if there are no existing databases in the instance.

/g:*network_name*
Changes the network name for the database partition server. This parameter can be used to apply a specific IP address to the database partition server when there are multiple IP addresses on a machine. The network name or the IP address can be entered.

Examples

To change the logical port assigned to database partition 2, which participates in the instance TESTMPP, to logical port 3, enter the following command:

```
db2nchg /n:2 /i:TESTMPP /p:3
```

db2ncrt - Add database partition server to an instance

Adds a database partition server to an instance.

This command is available on Windows operating systems only.

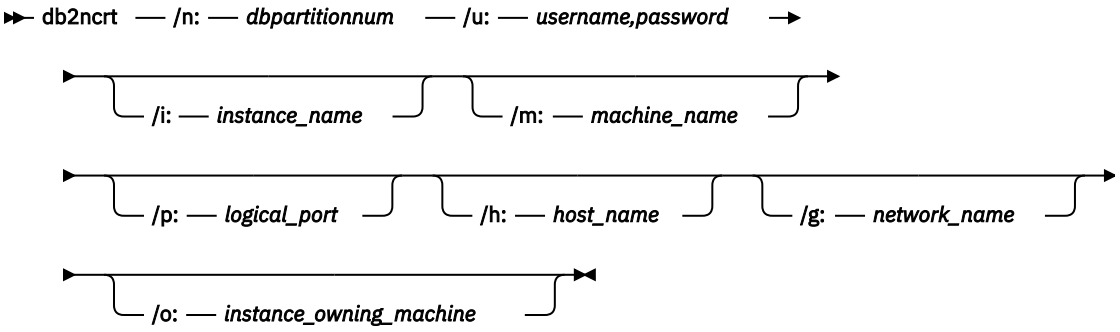
Scope

If a database partition server is added to a computer where an instance already exists, a database partition server is added as a logical database partition server to the computer. If a database partition server is added to a computer where an instance does not exist, the instance is added and the computer becomes a new physical database partition server. This command should not be used if there are databases in an instance. Instead, the **START DATABASE MANAGER** command should be issued with the **ADD DBPARTITIONNUM** option. This ensures that the database is correctly added to the new database partition server. It is also possible to add a database partition server to an instance in which a database has been created. The `db2nodes.cfg` file should not be edited since changing the file might cause inconsistencies in the partitioned database environment.

Authorization

`Local Administrator` authority on the computer where the new database partition server is added.

Command syntax



Command parameters

/n:*dbpartitionnum*
A unique database partition number which identifies the database partition server. The number entered can range from 1 to 999.

/u:*username,password*
Specifies the logon account name and password for Db2.

/i:*instance_name*
Specifies the instance name. If a parameter is not specified, the default is the current instance.

/m:*machine_name*
Specifies the computer name of the Windows workstation on which the database partition server resides. This parameter is required if a database partition server is added on a remote computer.

/p:*logical_port*
Specifies the logical port number used for the database partition server. If this parameter is not specified, the logical port number assigned will be 0. When creating a logical database partition server, this parameter must be specified and a logical port number that is not in use must be selected. Note the following restrictions:

- Every computer must have a database partition server that has a logical port 0.
- The port number cannot exceed the port range reserved for FCM communications in the `x:\winnt\system32\drivers\etc\` directory. For example, if a range of 4 ports is reserved for the current instance, then the maximum port number is 3. Port 0 is used for the default logical database partition server.

/h:*host_name*
Specifies the TCP/IP host name that is used by FCM for internal communications. This parameter is required when the database partition server is being added on a remote computer.

/g:*network_name*
Specifies the network name for the database partition server. If a parameter is not specified, the first IP address detected on the system will be used. This parameter can be used to apply a specific IP address to the database partition server when there are multiple IP addresses on a computer. The network name or the IP address can be entered.

/o:*instance_owning_machine*
Specifies the computer name of the instance-owning computer. The default is the local computer. This parameter is required when the **db2ncrt** command is invoked on any computer that is not the instance-owning computer.

Usage notes

db2ncrt command can be executed only on a partitioned database instance.

Examples

To add a new database partition server to the instance TESTMPP on the instance-owning computer SHAYER, where the new database partition server is known as database partition 2 and uses logical port 1, enter the following command:

```
db2ncrt /n:2 /u:QBPAULZ\paulz,g1reeky /i:TESTMPP /m:TEST /p:1 /o:SHAYER /h:TEST
```

db2ndrop - Drop database partition server from an instance

Drops a database partition server from an instance that has no databases. If a database partition server is dropped, its database partition number can be reused for a new database partition server.

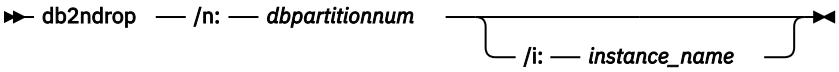
This command can only be used if the database partition server is stopped.

This command is available on Windows operating systems only.

Authorization

`Local Administrator` authority on the machine where the database partition server is being dropped.

Command syntax



Command parameters

/n:*dbpartitionnum*
A unique database partition number which identifies the database partition server.

/i:*instance_name*
Specifies the instance name. If a parameter is not specified, the default is the current instance.

Examples

```
db2ndrop /n:2 /i=KMASCI
```

Usage notes

If the instance-owning database partition server (dbpartitionnum 0) is dropped from the instance, the instance becomes unusable. To drop the instance, use the **db2idrop** command.

This command should not be used if there are databases in this instance. Instead, the **db2stop drop dbpartitionnum** command should be used. This ensures that the database partition server is correctly removed from the partition database environment. It is also possible to drop a database partition server in an instance where a database exists. The `db2nodes.cfg` file should not be edited since changing the file might cause inconsistencies in the partitioned database environment.

To drop a database partition server that is assigned to the logical port 0 from a machine that is running multiple logical database partition servers, all other database partition servers assigned to the other logical ports must be dropped first. Each database partition server must have a database partition server assigned to logical port 0.

SQL language elements

Data types

Database partition-compatible data types

Database partition compatibility is defined between the base data types of corresponding columns of distribution keys. Database partition-compatible data types have the property that two variables, one of each type, with the same value, are mapped to the same distribution map index by the same database partitioning function.

Table 42 on page 324 shows the compatibility of data types in database partitions.

Database partition compatibility has the following characteristics:

- Internal formats are used for DATE, TIME, and TIMESTAMP. They are not compatible with each other, and none are compatible with character or graphic data types.
- Partition compatibility is not affected by the nullability of a column.
- Partition compatibility is affected by collation. Locale-sensitive UCA-based collations require an exact match in collation, except that the strength (S) attribute of the collation is ignored. All other collations are considered equivalent for the purposes of determining partition compatibility.
- Character columns defined with FOR BIT DATA are only compatible with character columns without FOR BIT DATA when a collation other than a locale-sensitive UCA-based collation is used.
- Null values of compatible data types are treated identically. Different results might be produced for null values of non-compatible data types.
- Base data type of the UDT is used to analyze database partition compatibility.
- Timestamps of the same value in the distribution key are treated identically, even if their timestamp precisions differ.
- Decimals of the same value in the distribution key are treated identically, even if their scale and precision differ.
- Trailing blanks in character strings (CHAR, VARCHAR, GRAPHIC or VARGRAPHIC) are ignored by the system-provided hashing function.
- When a locale-sensitive UCA-based collation is used, CHAR, VARCHAR, GRAPHIC, and VARGRAPHIC are compatible data types. When other collations are used, CHAR and VARCHAR are compatible types and GRAPHIC and VARGRAPHIC are compatible types, but CHAR and VARCHAR are not compatible types with GRAPHIC and VARGRAPHIC. CHAR or VARCHAR of different lengths are compatible data types.
- DECFLOAT values that are equal are treated identically even if their precision differs. DECFLOAT values that are numerically equal are treated identically even if they have a different number of significant digits.
- Data types that are not supported as part of a distribution key are not applicable for database partition compatibility. Examples of such data types are:
  - BLOB
  - CLOB
  - DBCLOB
  - XML
  - A distinct type based on BLOB, CLOB, DBCLOB, or XML
  - A structured type

Table 42. Database Partition Compatibilities

Operands	Binary Integer	Decimal Number	Floating-point	Decimal Floating-point	Character String	Graphic String	Binary String	Date	Time	Time-stamp	Distinct Type	Boolean
Binary Integer	Yes	No	No	No	No	No	No	No	No	No	1	No
Decimal Number	No	Yes	No	No	No	No	No	No	No	No	1	No
Floating-point	No	No	Yes	No	No	No	No	No	No	No	1	No
Decimal Floating-point	No	No	No	Yes	No	No	No	No	No	No	1	No
Character String	No	No	No	No	Yes[2]	2, 3	No	No	No	No	1	No
Graphic String	No	No	No	No	2, 3	Yes[2]	No	No	No	No	1	No
Binary String							Yes					No
Date	No	No	No	No	No	No	No	Yes	No	No	1	No
Time	No	No	No	No	No	No	No	No	Yes	No	1	No
Timestamp	No	No	No	No	No	No	No	No	No	Yes	1	No
Distinct Type	1	1	1	1	1	1	1	1	1	1	1	1
Boolean	No	No	No	No	No	No	No	No	No	No	1	Yes

Note:

1
A distinct type value is database partition compatible with the source data type of the distinct type or with any other distinct type with the same source data type. The source data type of the distinct type must be a data type that is supported as part of a distribution key. A user-defined distinct type (UDT) value is database partition compatible with the source type of the UDT or any other UDT with a database partition compatible source type. A distinct type cannot be based on BLOB, CLOB, DBCLOB, or XML.

2
Character and graphic string types are compatible when they have compatible collations.

3
Character and graphic string types are compatible when a locale-sensitive UCA-based collation is in effect. Otherwise, they are not compatible types.

Special registers

CURRENT MEMBER
The CURRENT MEMBER special register specifies an INTEGER value that identifies the coordinator member for the statement.

For statements issued from an application, the coordinator is the member to which the application connects. For statements issued from a routine, the coordinator is the member from which the routine is invoked.

When used in an SQL statement inside a routine, CURRENT MEMBER is never inherited from the invoking statement.

CURRENT MEMBER returns 0 if the database instance is not defined to support database partitioning or the IBM Db2 pureScale Feature. The database instance is not defined for such support if there is no db2nodes.cfg file. For a partitioned database or a Db2 pureScale environment, the db2nodes.cfg file exists and contains database partition and member definitions.

CURRENT MEMBER can be changed through the CONNECT statement, but only under certain conditions.

For compatibility with previous versions of Db2 and with other database products, NODE can be specified in place of MEMBER.

Examples

Example 1: Set the host variable APPL_NODE (integer) to the number of the member to which the application is connected.

```
VALUES CURRENT MEMBER
  INTO :APPL_NODE
```

Example 2: The following command is issued on member 0 and on a 4 member system in a partitioned database environment. This query will retrieve the currently connected database member number.

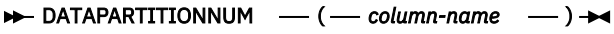
```
VALUES CURRENT MEMBER

1
-----------
          0
```

SQL functions

DATAPARTITIONNUM

The DATAPARTITIONNUM function returns the sequence number (SYSDATAPARTITIONS.SEQNO) of the data partition in which the row resides.



The schema is SYSIBM.

column-name
The qualified or unqualified name of any column in the table. Because row-level information is returned, the result is the same regardless of which column is specified. The column can have any data type.

If the column is a column of a view, the expression for the column in the view must reference a column of the underlying base table, and the view must be deletable. A nested or common table expression follows the same rules as a view.

Result

The data type of the result is INTEGER and is never null.

Data partitions are sorted by range, and sequence numbers start at 0. For example, the DATAPARTITIONNUM function returns 0 for a row that resides in the data partition with the lowest range.

Notes

- This function cannot be used as a source function when creating a user-defined function. Because the function accepts any data type as an argument, it is not necessary to create additional signatures to support user-defined distinct types.
- The DATAPARTITIONNUM function cannot be used within check constraints or in the definition of generated columns (SQLSTATE 42881). The DATAPARTITIONNUM function cannot be used in a materialized query table (MQT) definition (SQLSTATE 428EC).
- The DATAPARTITIONNUM function cannot be used as part of an expression-based key in a CREATE INDEX statement.

Examples

- *Example 1:* Retrieve the sequence number of the data partition in which the row for EMPLOYEE.EMPNO resides.

```
SELECT DATAPARTITIONNUM (EMPNO)
FROM EMPLOYEE
```

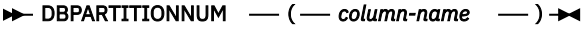
- *Example 2:* To convert a sequence number that is returned by DATAPARTITIONNUM (for example, 0) to a data partition name that can be used in other SQL statements (such as ALTER TABLE...DETACH PARTITION), you can query the SYSCAT.DATAPARTITIONS catalog view. Include the SEQNO obtained from DATAPARTITIONNUM in the WHERE clause, as shown in the following example.

```
SELECT DATAPARTITIONNAME
FROM SYSCAT.DATAPARTITIONS
WHERE TABNAME = 'EMPLOYEE' AND SEQNO = 0
```

  results in the value 'PART0'.

DBPARTITIONNUM

The DBPARTITIONNUM function returns the database partition number for a row. For example, if used in a SELECT clause, it returns the database partition number for each row in the result set.



The schema is SYSIBM.

column-name
The qualified or unqualified name of any column in the table. Because row-level information is returned, the result is the same regardless of which column is specified. The column can have any data type.

If the column is a column of a view, the expression for the column in the view must reference a column of the underlying base table, and the view must be deletable. A nested or common table expression follows the same rules as a view.

The specific row (and table) for which the database partition number is returned by the DBPARTITIONNUM function is determined from the context of the SQL statement that uses the function.

The database partition number returned on transition variables and tables is derived from the current transition values of the distribution key columns. For example, in a before insert trigger, the function returns the projected database partition number, given the current values of the new transition variables. However, the values of the distribution key columns might be modified by a subsequent before insert trigger. Thus, the final database partition number of the row when it is inserted into the database might differ from the projected value.

Result

The data type of the result is INTEGER and is never null. If there is no `db2nodes.cfg` file, the result is 0.

Notes

- The DBPARTITIONNUM function cannot be used on replicated tables, within check constraints, or in the definition of generated columns (SQLSTATE 42881).
- The DBPARTITIONNUM function cannot be used as a source function when creating a user-defined function. Because it accepts any data type as an argument, it is not necessary to create additional signatures to support user-defined distinct types.
- The DBPARTITIONNUM function cannot be used as part of an expression-based key in a CREATE INDEX statement.
- ***Syntax alternatives:*** For compatibility with previous versions of Db2 products, the function name NODENUMBER is a synonym for DBPARTITIONNUM.

Examples

- *Example 1*: Count the number of instances in which the row for a given employee in the EMPLOYEE table is on a different database partition than the description of the employee's department in the DEPARTMENT table.

```
SELECT COUNT(*) FROM DEPARTMENT D, EMPLOYEE E
  WHERE D.DEPTNO=E.WORKDEPT
  AND DBPARTITIONNUM(E.LASTNAME) <> DBPARTITIONNUM(D.DEPTNO)
```

- *Example 2*: Join the EMPLOYEE and DEPARTMENT tables so that the rows of the two tables are on the same database partition.

```
SELECT * FROM DEPARTMENT D, EMPLOYEE E
  WHERE DBPARTITIONNUM(E.LASTNAME) = DBPARTITIONNUM(D.DEPTNO)
```

- *Example 3*: Using a before trigger on the EMPLOYEE table, log the employee number and the projected database partition number of any new row in the EMPLOYEE table in a table named EMPINSERTLOG1.

```
CREATE TRIGGER EMPINSLOGTRIG1
BEFORE INSERT ON EMPLOYEE
REFERENCING NEW AW NEWTABLE
FOR EACH ROW
INSERT INTO EMPINSERTLOG1
VALUES(NEWTABLE.EMPNO, DBPARTITIONNUM
(NEWTABLE.EMPNO))
```

SQL statements

ALTER DATABASE PARTITION GROUP

The ALTER DATABASE PARTITION GROUP statement is used to add one or more database partitions to a database partition group, or drop one or more database partitions from a database partition group.

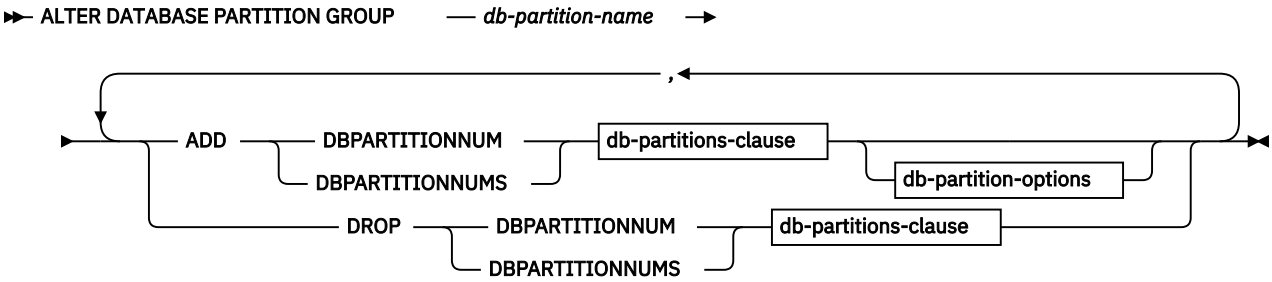
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

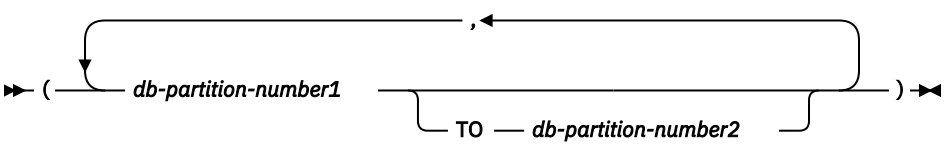
Authorization

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

Syntax



db-partitions-clause



db-partition-options

Description

db-partition-name
Names the database partition group. This is a one-part name. It is an SQL identifier (either ordinary or delimited). It must be a database partition group described in the catalog. IBMCATGROUP and IBMTEMPGROUP cannot be specified (SQLSTATE 42832).

ADD DBPARTITIONNUM
Specifies the specific database partition or partitions to add to the database partition group. DBPARTITIONNUMS is a synonym for DBPARTITIONNUM. Any specified database partition must not already be defined in the database partition group (SQLSTATE 42728).

DROP DBPARTITIONNUM
Specifies the specific database partition or partitions to drop from the database partition group. DBPARTITIONNUMS is a synonym for DBPARTITIONNUM. Any specified database partition must already be defined in the database partition group (SQLSTATE 42729).

db-partitions-clause
Specifies the database partition or partitions to be added or dropped.

db-partition-number1
Specify a specific database partition number.

TO *db-partition-number2*
Specify a range of database partition numbers. The value of *db-partition-number2* must be greater than or equal to the value of *db-partition-number1* (SQLSTATE 428A9).

db-partition-options

LIKE DBPARTITIONNUM *db-partition-number*
Specifies that the containers for the existing table spaces in the database partition group will be the same as the containers on the specified *db-partition-number*. The specified database partition must be a partition that existed in the database partition group before this statement, and that is not included in a DROP DBPARTITIONNUM clause of the same statement.

For table spaces that are defined to use automatic storage (that is, table spaces that were created with the MANAGED BY AUTOMATIC STORAGE clause of the CREATE TABLESPACE statement, or for which no MANAGED BY clause was specified at all), the containers will not necessarily match those from the specified partition. Instead, containers will automatically be assigned by the database manager based on the storage paths that are associated with the database, and this might or might not result in the same containers being used. The size of each table space is based on the initial size that was specified when the table space was created, and might not match the current size of the table space on the specified partition.

WITHOUT TABLESPACES
Specifies that the containers for existing table spaces in the database partition group are not created on the newly added database partition or partitions. The ALTER TABLESPACE statement using the *db-partitions-clause* or the MANAGED BY AUTOMATIC STORAGE clause must be used to define containers for use with the table spaces that are defined on this database partition group. If this option is not specified, the default containers are specified on newly added database partitions for each table space defined on the database partition group.

This option is ignored for table spaces that are defined to use automatic storage (that is, table spaces that were created with the MANAGED BY AUTOMATIC STORAGE clause of the CREATE TABLESPACE statement, or for which no MANAGED BY clause was specified at all). There is no way to defer container creation for these table spaces. Containers will automatically be assigned by the database manager based on the storage paths that are associated with the database. The

size of each table space will be based on the initial size that was specified when the table space was created.

Rules

- Each database partition specified by number must be defined in the `db2nodes.cfg` file (SQLSTATE 42729).
- Each *db-partition-number* listed in the *db-partitions-clause* must be for a unique database partition (SQLSTATE 42728).
- A valid database partition number is between 0 and 999 inclusive (SQLSTATE 42729).
- A database partition cannot appear in both the ADD and DROP clauses (SQLSTATE 42728).
- There must be at least one database partition remaining in the database partition group. The last database partition cannot be dropped from a database partition group (SQLSTATE 428C0).
- If neither the LIKE DBPARTITIONNUM clause nor the WITHOUT TABLESPACES clause is specified when adding a database partition, the default is to use the lowest database partition number of the existing database partitions in the database partition group (say it is *2*) and proceed as if LIKE DBPARTITIONNUM *2* had been specified. For an existing database partition to be used as the default, it must have containers defined for all the table spaces in the database partition group (column IN_USE of SYSCAT.DBPARTITIONGROUPDEF is not 'T').
- The ALTER DATABASE PARTITION GROUP statement might fail (SQLSTATE 55071) if an add database partition server request is either pending or in progress. This statement might also fail (SQLSTATE 55077) if a new database partition server is added online to the instance and not all applications are aware of the new database partition server.

Notes

- When a database partition is added to a database partition group, a catalog entry is made for the database partition (see SYSCAT.DBPARTITIONGROUPDEF). The distribution map is changed immediately to include the new database partition, along with an indicator (IN_USE) that the database partition is in the distribution map if either:
  - no table spaces are defined in the database partition group or
  - no tables are defined in the table spaces defined in the database partition group and the WITHOUT TABLESPACES clause was not specified.

  The distribution map is not changed and the indicator (IN_USE) is set to indicate that the database partition is not included in the distribution map if either:
  - Tables exist in table spaces in the database partition group or
  - Table spaces exist in the database partition group and the WITHOUT TABLESPACES clause was specified (unless all of the table spaces are defined to use automatic storage, in which case the WITHOUT TABLESPACES clause is ignored)

  To change the distribution map, the REDISTRIBUTE DATABASE PARTITION GROUP command must be used. This redistributes any data, changes the distribution map, and changes the indicator. Table space containers need to be added before attempting to redistribute data if the WITHOUT TABLESPACES clause was specified.
- When a database partition is dropped from a database partition group, the catalog entry for the database partition (see SYSCAT.DBPARTITIONGROUPDEF) is updated. If there are no tables defined in the table spaces defined in the database partition group, the distribution map is changed immediately to exclude the dropped database partition and the entry for the database partition in the database partition group is dropped. If tables exist, the distribution map is not changed and the indicator (IN_USE) is set to indicate that the database partition is waiting to be dropped. The REDISTRIBUTE DATABASE PARTITION GROUP command must be used to redistribute the data and drop the entry for the database partition from the database partition group.

- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with previous versions of Db2 and with other database products. These alternatives are non-standard and should not be used.
  - NODE can be specified in place of DBPARTITIONNUM
  - NODES can be specified in place of DBPARTITIONNUMS
  - NODEGROUP can be specified in place of DATABASE PARTITION GROUP

Example

Assume that you have a six-partition database that has the following database partitions: 0, 1, 2, 5, 7, and 8. Two database partitions (3 and 6) are added to the system.

- *Example 1*: Assume that you want to add database partitions 3 and 6 to a database partition group called MAXGROUP, and have table space containers like those on database partition 2. The statement is as follows:

```
ALTER DATABASE PARTITION GROUP MAXGROUP
  ADD DBPARTITIONNUMS (3,6)LIKE DBPARTITIONNUM 2
```

- *Example 2*: Assume that you want to drop database partition 1 and add database partition 6 to database partition group MEDGROUP. You will define the table space containers separately for database partition 6 using ALTER TABLESPACE. The statement is as follows:

```
ALTER DATABASE PARTITION GROUP MEDGROUP
  ADD DBPARTITIONNUM(6)WITHOUT TABLESPACES
  DROP DBPARTITIONNUM(1)
```

CREATE DATABASE PARTITION GROUP

The CREATE DATABASE PARTITION GROUP statement defines a new database partition group within the database, assigns database partitions to the database partition group, and records the database partition group definition in the system catalog.

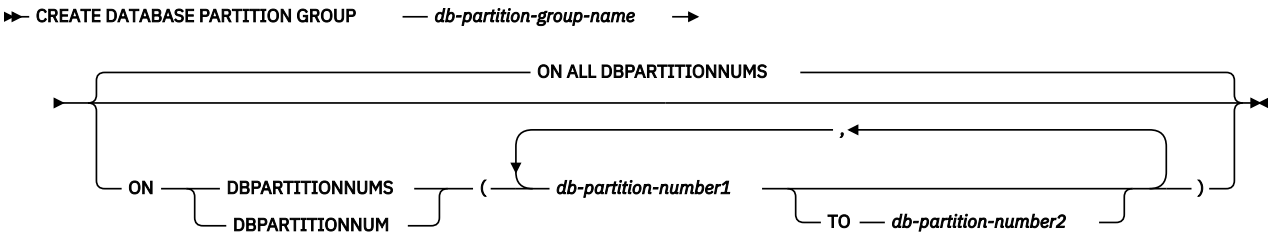
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include SYSCTRL or SYSADM authority.

Syntax



Description

db-partition-group-name
Names the database partition group. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The *db-partition-group-name* must not identify a database partition group that already

exists in the catalog (SQLSTATE 42710). The *db-partition-group-name* must not begin with the characters 'SYS' or 'IBM' (SQLSTATE 42939).

ON ALL DBPARTITIONNUMS
Specifies that the database partition group is defined over all database partitions defined to the database (db2nodes.cfg file) at the time the database partition group is created.

If a database partition is added to the database system, the ALTER DATABASE PARTITION GROUP statement should be issued to include this new database partition in a database partition group (including IBMDEFAULTGROUP). Furthermore, the REDISTRIBUTE DATABASE PARTITION GROUP command must be issued to move data to the database partition.

ON DBPARTITIONNUMS
Specifies the database partitions that are in the database partition group. DBPARTITIONNUM is a synonym for DBPARTITIONNUMS.

db-partition-number1
Specify a database partition number. (A *node-name* of the form NODE*nnnnn* can be specified for compatibility with the previous version.)

TO *db-partition-number2*
Specify a range of database partition numbers. The value of *db-partition-number2* must be greater than or equal to the value of *db-partition-number1* (SQLSTATE 428A9). All database partitions between and including the specified database partition numbers are included in the database partition group.

Rules

- Each database partition specified by number must be defined in the db2nodes.cfg file (SQLSTATE 42729).
- Each *db-partition-number* listed in the ON DBPARTITIONNUMS clause can appear only once (SQLSTATE 42728).
- A valid *db-partition-number* is between 0 and 999 inclusive (SQLSTATE 42729).
- The CREATE DATABASE PARTITION GROUP statement might fail (SQLSTATE 55071) if an add database partition server request is either pending or in progress. This statement might also fail (SQLSTATE 55077) if a new database partition server is added online to the instance and not all applications are aware of the new database partition server.

Notes

- This statement creates a distribution map for the database partition group. A distribution map identifier (PMAP_ID) is generated for each distribution map. This information is recorded in the catalog and can be retrieved from SYSCAT.DBPARTITIONGROUPS and SYSCAT.PARTITIONMAPS. Each entry in the distribution map specifies the target database partition on which all rows that are hashed reside. For a single-partition database partition group, the corresponding distribution map has only one entry. For a multiple partition database partition group, the corresponding distribution map has 32768 entries, where the database partition numbers are assigned to the map entries in a round-robin fashion, by default.
- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with previous versions of Db2 and with other database products. These alternatives are non-standard and should not be used.
  - NODE can be specified in place of DBPARTITIONNUM
  - NODES can be specified in place of DBPARTITIONNUMS
  - NODEGROUP can be specified in place of DATABASE PARTITION GROUP

Examples

The following examples are based on a partitioned database with six database partitions defined as 0, 1, 2, 5, 7, and 8.

- *Example 1*: Assume that you want to create a database partition group called MAXGROUP on all six database partitions. The statement is as follows:

```
CREATE DATABASE PARTITION GROUP MAXGROUP ON ALL DBPARTITIONNUMS
```

- *Example 2*: Assume that you want to create a database partition group called MEDGROUP on database partitions 0, 1, 2, 5, and 8. The statement is as follows:

```
CREATE DATABASE PARTITION GROUP MEDGROUP
  ON DBPARTITIONNUMS( 0 TO 2, 5, 8)
```

- *Example 3*: Assume that you want to create a single-partition database partition group MINGROUP on database partition 7. The statement is as follows:

```
CREATE DATABASE PARTITION GROUP MINGROUP
  ON DBPARTITIONNUM (7)
```

Supported administrative SQL routines and views

ADMIN_CMD stored procedure and associated administrative SQL routines

GET STMM TUNING command using the ADMIN_CMD procedure

The **GET STMM TUNING** command reads the catalog tables to report the user preferred self tuning memory manager (STMM) tuning member number and current STMM tuning member number.

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities or privilege:

- DBADM
- SECADM
- SQLADM
- ACCESSCTRL
- DATAACCESS
- SELECT on SYSIBM.SYSTUNINGINFO

Required connection

Database

Command syntax

Example

```
CALL SYSPROC.ADMIN_CMD( 'get stmm tuning member' )
```

The following is an example of output from this query.

```
Result set 1
  --------------

  USER_PREFERRED_NUMBER CURRENT_NUMBER
  --------------------- --------------
                      2              2

  1 record(s) selected.
```

```
Return Status = 0
```

Usage notes

- The user preferred self tuning memory manager (STMM) tuning member number (USER_PREFERRED_NUMBER) is set by the user and specifies the member on which the user wants to run the memory tuner. While the database is running, the tuning member is applied a few times an hour. As a result, it is possible that the CURRENT_NUMBER and USER_PREFERRED_NUMBER returned are not in sync after an update of the user preferred STMM member. To resolve this, either wait for the CURRENT_NUMBER to be updated asynchronously, or stop and start the database to force the update of CURRENT_NUMBER.

Compatibilities

For compatibility with previous versions:

- **DBPARTITIONNUM** can be substituted for **MEMBER**, except when the **DB2_ENFORCE_MEMBER_SYNTAX** registry variable is set to ON.

Result set information

Command execution status is returned in the SQLCA resulting from the CALL statement. If execution is successful, the command returns additional information in the following result set:

Table 43. Result set returned by the GET STMM TUNING command

Column name	Data type	Description
USER_PREFERRED_NUMBER	INTEGER	User preferred self tuning memory manager (STMM) tuning member number. In a partitioned database environment, a value of -1 indicates that the default member is used.
CURRENT_NUMBER	INTEGER	Current STMM tuning member number. A value of -1 indicates that the default member is used.

UPDATE STMM TUNING command by using the ADMIN_CMD procedure

The **UPDATE STMM TUNING** command updates the user preferred database member number on which the self-tuning memory manager (STMM) tuner is created.

Authorization

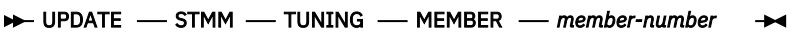
The privileges that are held by the authorization ID of the statement must include at least one of the following authorities:

- DBADM
- DATAACCESS
- SQLADM

Required connection

Database

Command syntax



Command parameter

member-number
The value of *member-number* is an integer.

In a partitioned database environment:

- If a valid member number is specified, the database server runs the STMM memory tuner on that member.
- If -1 or a nonexistent member number is specified, the database server selects an appropriate member on which to run the STMM memory tuner.

In a Db2 pureScale environment:

- If a valid member number is specified, database server runs the STMM memory tuner on that member.
- If -2 is specified, then the database server enables STMM tuners to run and tune independently on each member.
- If -1 or a nonexistent member number is specified, the database server selects an appropriate member on which to run the STMM memory tuner.

Example

In a partitioned database environment, update the user preferred self-tuning memory manager (STMM) tuning database partition to member 3.

```
CALL SYSPROC.ADMIN_CMD('update stmm tuning member 3')
```

Usage notes

- The STMM tuning process periodically checks for a change in the user preferred STMM tuning member number value. The STMM tuning process moves to the user preferred STMM tuning member if *member-number* exists and is an active member. If the specified STMM tuning member does not exist, the database server selects an appropriate member. After this command changes the STMM tuning member number, an immediate change is made to the current STMM tuning member number.
- Command execution status is returned in the SQLCA resulting from the **CALL** statement.
- This command commits its changes in the **ADMIN_CMD** procedure.

Compatibilities

For compatibility with previous versions:

- **DBPARTITIONNUM** can be substituted for **MEMBER**, except when the **DB2_ENFORCE_MEMBER_SYNTAX** registry variable is set to ON.

Configuration administrative SQL routines and views

DB_PARTITIONS
The DB_PARTITIONS table function returns the contents of the db2nodes.cfg file in table format.

Note: This table function has been deprecated and replaced by the DB2_MEMBER and DB2_CF administrative views and DB2_GET_INSTANCE_INFO table function.

Syntax

The schema is SYSPROC.

Authorization

One of the following authorities is required to execute the routine:

- EXECUTE privilege on the routine
- DATAACCESS authority
- DBADM authority
- SQLADM authority

Default PUBLIC privilege

In a non-restrictive database, EXECUTE privilege is granted to PUBLIC when the function is automatically created.

Table function parameters

The function has no input parameters.

Examples

Retrieve information from a 4 member partitioned database instance.

```
SELECT * FROM TABLE(DB_PARTITIONS()) as T
```

The following is an example of the output from this query:

```
PARTITION_NUMBER HOST_NAME PORT_NUMBER SWITCHNAME
---------------- --------- ----------- -------
               0 so1                 0 so1-ib0
               1 so2                 0 so2-ib0
               2 so3                 0 so3-ib0
               3 so4                 0 so4-ib0

     4 record(s) selected.
```

In a Db2 pureScale environment, retrieve information from a 3 member and 1 cluster caching facility Db2 pureScale instance.

```
SELECT * FROM TABLE(DB_PARTITIONS()) as T
```

The following is an example of the output from this query:

```
PARTITION_NUMBER HOST_NAME  PORT_NUMBER SWITCHNAME
---------------- ---------- ----------- ----------
               0 so1                  0 so1-ib0
               0 so2                  0 so2-ib0
               0 so3                  0 so3-ib0

     3 record(s) selected.
```

Usage notes

For Db2 Enterprise Server Edition and in a partitioned database environment, the DB_PARTITIONS table function returns one row for each entry in the `db2nodes.cfg` file.

In a Db2 pureScale environment, the DB_PARTITIONS table function returns multiple rows, with the following information in the columns:

- The PARTITION_NUMBER column always contains 0.
- The remaining columns show information for the entry in the `db2nodes.cfg` file for the current member.

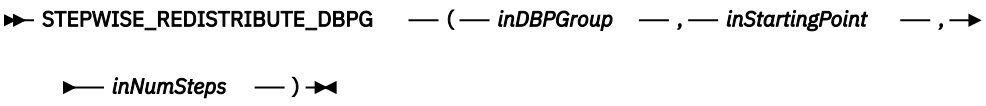
Information returned

Table 44. Information returned by the DB_PARTITIONS table function		
Column name	**Data type**	**Description**
PARTITION_NUMBER	SMALLINT	partition_number - Partition Number monitor element
HOST_NAME	VARCHAR(256)	host_name - Host name monitor element
PORT_NUMBER	SMALLINT	The port number for the database partition server.
SWITCH_NAME	VARCHAR(128)	The name of a high speed interconnect, or switch, for database partition communications.

Stepwise redistribute administrative SQL routines

STEPWISE_REDISTRIBUTE_DBPG procedure - Redistribute part of database partition group
The STEPWISE_REDISTRIBUTE_DBPG procedure redistributes part of the database partition group according to the input specified for the procedure, and the setting file created or updated by the SET_SWRD_SETTINGS procedure.

Syntax



The schema is SYSPROC.

Procedure parameters

inDBPGroup
An input argument of type VARCHAR (128) that specifies the name of the target database partition group.

inStartingPoint
An input argument of type SMALLINT that specifies the starting point to use. If the parameter is set to a positive integer and is not NULL, the STEPWISE_REDISTRIBUTE_DBPG procedure uses this value instead of using the *nextStep* value specified in the setting file. This is a useful option when you want to rerun the STEPWISE_REDISTRIBUTE_DBPG procedure from a particular step. If the parameter is set to NULL, the *nextStep* value is used.

inNumSteps
An input argument of type SMALLINT that specifies the number of steps to run. If the parameter is set to a positive integer and is not NULL, the STEPWISE_REDISTRIBUTE_DBPG procedure uses this value instead of using the *stageSize* value specified in the setting file. This is a useful option when you want to rerun the STEPWISE_REDISTRIBUTE_DBPG procedure with a different number of steps than what is specified in the settings. For example, if there are five steps in a scheduled stage, and the redistribution process failed at step 3, the STEPWISE_REDISTRIBUTE_DBPG procedure can be called to run the remaining three steps once the error condition has been corrected. If the parameter is set to NULL, the *stageSize* value is used. The value -2 can be used in this procedure to indicate that the number is unlimited.

Note: There is no parameter for specifying the equivalent of the **NOT ROLLFORWARD RECOVERABLE** option on the **REDISTRIBUTE DATABASE PARTITION GROUP** command. Logging is always performed for row data redistribution performed when the STEPWISE_REDISTRIBUTE_DBPG procedure is used.

Authorization

- EXECUTE privilege on the STEPWISE_REDISTRIBUTE_DBPG procedure
- SYSADM, SYSCTRL or DBADM

Default PUBLIC privilege

In a non-restrictive database, EXECUTE privilege is granted to PUBLIC when the procedure is automatically created.

Example

Redistribute the database partition group "IBMDEFAULTGROUP" according to the redistribution plan stored in the registry by the SET_SWRD_SETTINGS procedure. It is starting with step 3 and redistributes the data until 2 steps in the redistribution plan are completed.

```
CALL SYSPROC.STEPWISE_REDISTRIBUTE_DBPG('IBMDEFAULTGROUP', 3, 2)
```

For a full usage example of the stepwise redistribute procedures, refer to "Redistributing database partition groups using the STEPWISE_REDISTRIBUTE_DBPG procedure" in the *Partitioning and Clustering Guide*.

Usage notes

If the registry value for *processState* is updated to 1 using the SET_SWRD_SETTINGS procedure after the STEPWISE_REDISTRIBUTE_DBPG procedure execution is started, the process stops at the beginning to the next step and a warning message is returned.

As the SQL COMMIT statement is called by the redistribute process, running the redistribute process under a Type-2 connection is not supported.

Index

D

E

F

G

H

M

N

O

P

Q

R

S

T

U

V

W

X