Db2 11.1 for Linux, UNIX, and Windows

Partition and Clustering Guide

Db2 11.1 for Linux, UNIX, and Windows

Partition and Clustering Guide

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.

Contents

Figures

Tables

Chapter 1. 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:

(artname: sout.gif) Best practices: Managing data growth

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.

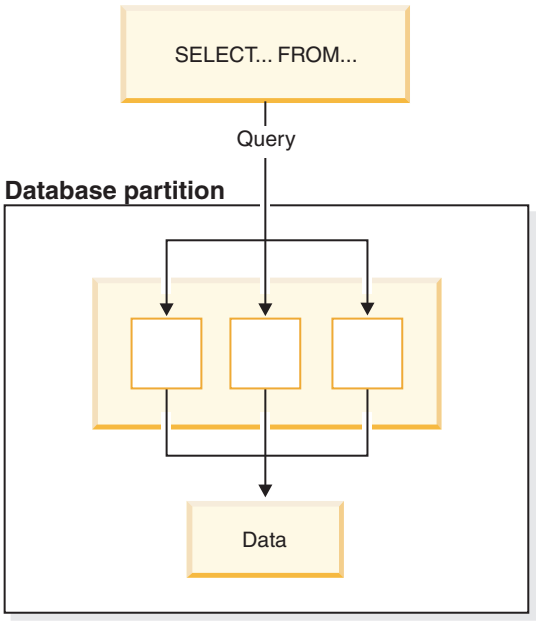
Intraquery parallelism refers to the simultaneous processing of parts of a single query, using either intrapartition parallelism, interpartition parallelism, or both.

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 1 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.



(artname: 00000226.gif)
Figure 1. Intrapartition parallelism

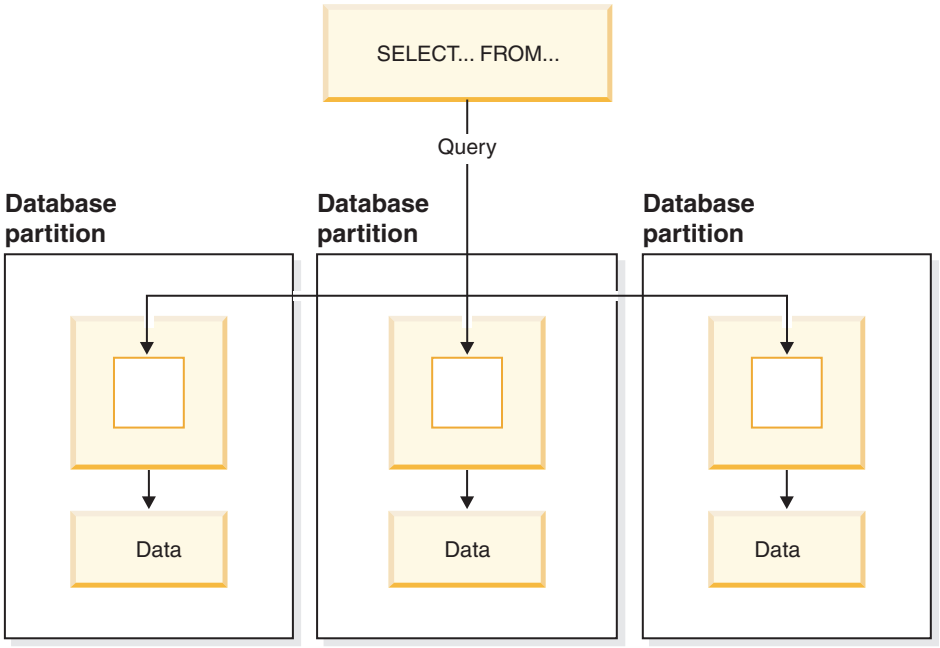
Interpartition parallelism

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 2 on page 4 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.

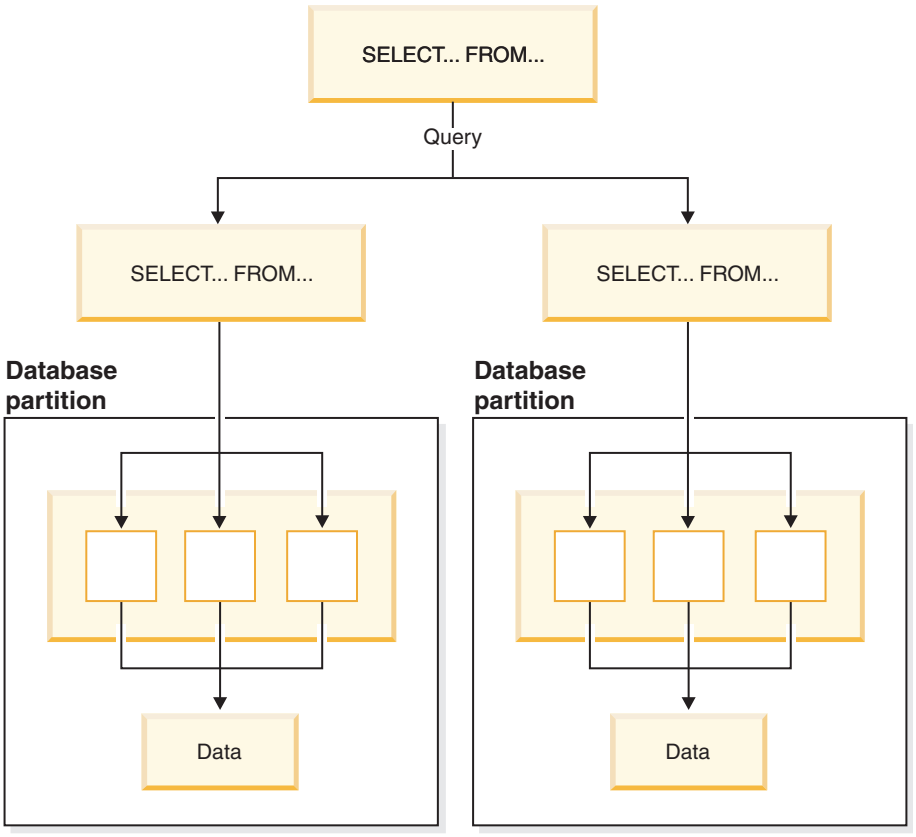
The degree of parallelism is largely determined by the number of database partitions you create and how you define your database partition groups.



(artname: 00000225.gif)
Figure 2. 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.



(artname: 00000159.gif)
Figure 3. 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.

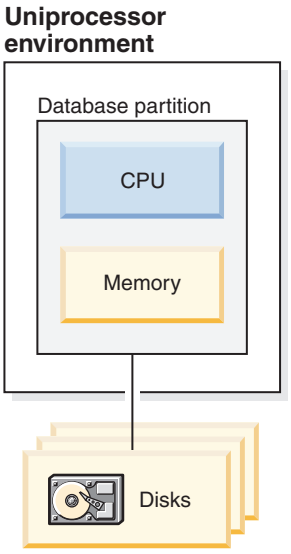
Database partition and processor environments

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 4). 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.



(artname: 00000236.gif)
Figure 4. Single database partition on a single processor

Capacity and scalability

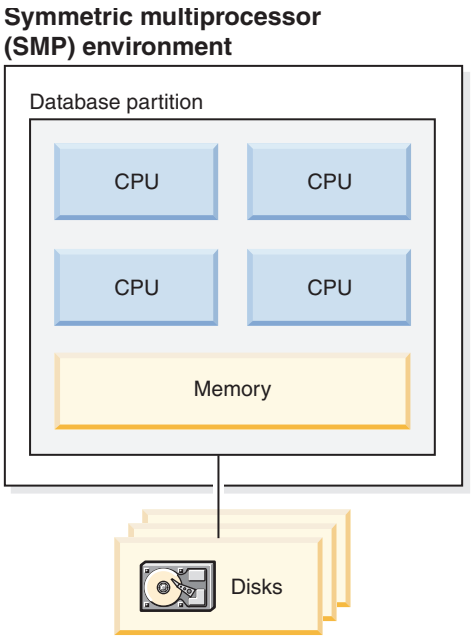
In this environment you can add more disks. Having one or more I/O servers for each disk allows for more than one I/O operation to take place at the same time.

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 5), 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.



(artname: 00000217.gif)
Figure 5. 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

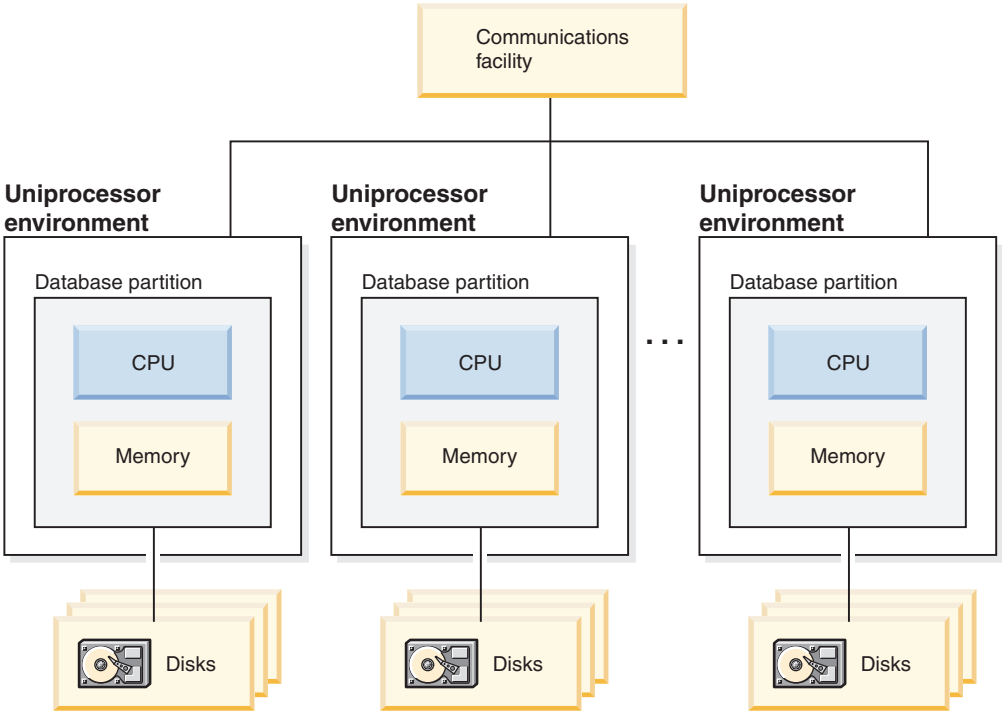
You can divide a database into multiple database partitions, each on its own machine. Multiple machines with multiple database partitions can be grouped together. This section describes the following database partition configurations:

- 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 6). 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.



(artname: 00000194.gif)
Figure 6. Massively parallel processing (MPP) environment

Capacity and scalability

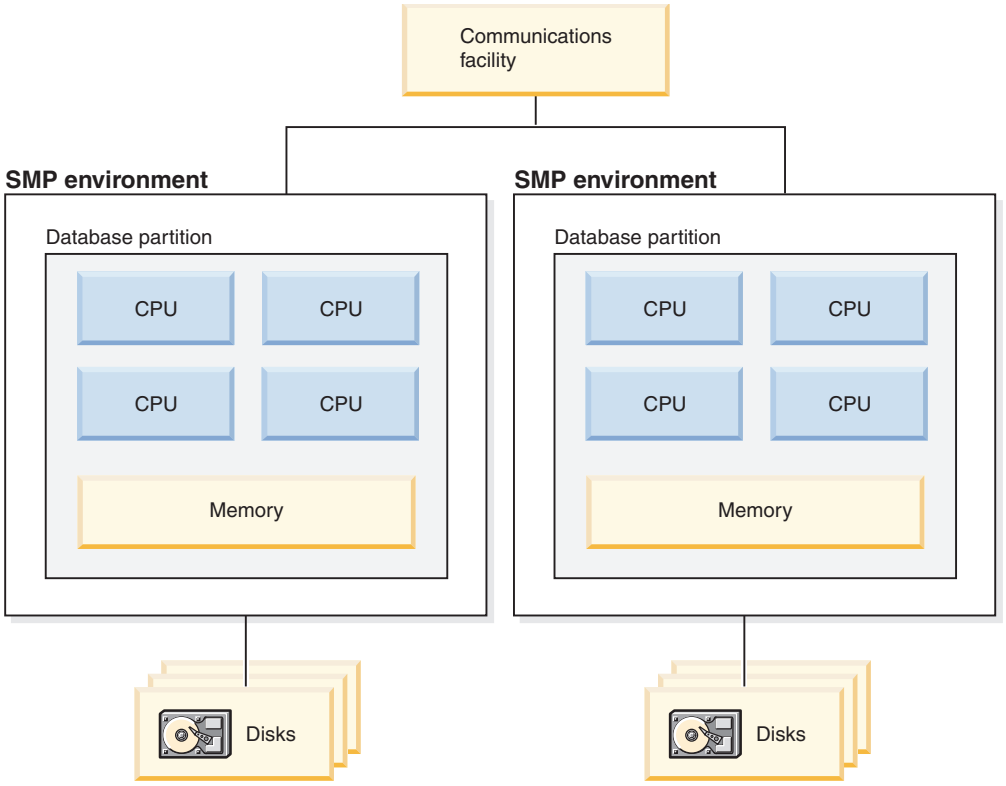
In this environment you can add more database partitions to your configuration. On some platforms the maximum number is 512 database partitions. However, there might be practical limits on managing a high number of machines and instances.

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 7).

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.



(artname: 00000218.gif)
Figure 7. Several symmetric multiprocessor (SMP) environments in a cluster

Capacity and scalability

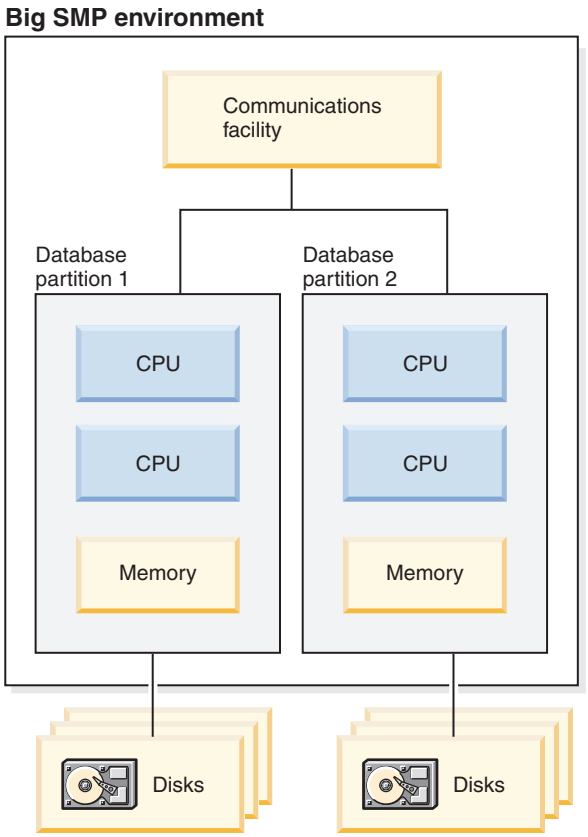
In this environment you can add more database partitions, and you can add more processors to existing database partitions.

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.

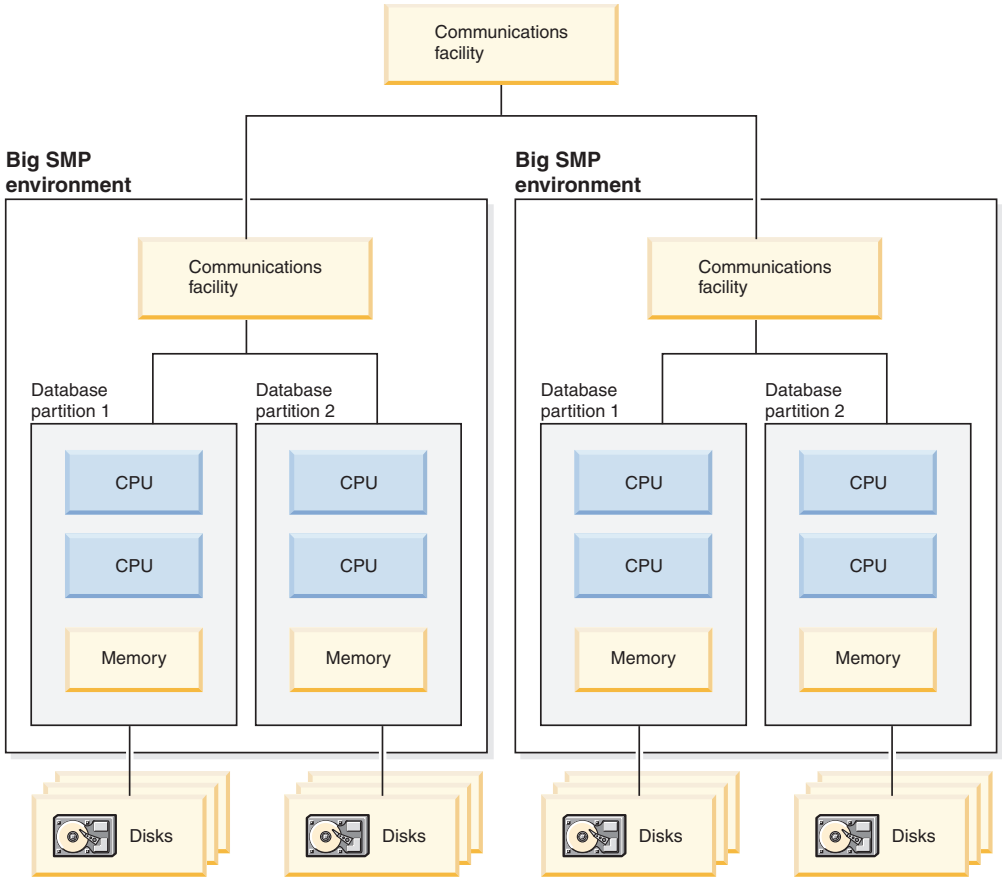
Logical database partitions provide scalability. Multiple database managers running on multiple logical partitions can make fuller use of available resources than a single database manager can. Figure 8 on page 10 illustrates the fact that you can gain more scalability on an SMP machine by adding more database partitions; this

is particularly true for machines with many processors. By distributing the database, you can administer and recover each database partition separately.



(artname: 00000189.gif)
Figure 8. Partitioned database with symmetric multiprocessor environment

Figure 9 on page 11 illustrates that you can multiply the configuration shown in Figure 8 to increase processing power.



(artname: 00000219.gif)
Figure 9. 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 1. 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
Multiple Database Partitions, One Processor (MPP)	Yes	No [1]	Yes

Table 1. 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, 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. Column-organized tables in a Partitioned database environment

Using column-organized tables in a massively parallel processing (MPP) environment may improve performance or reduce cost, depending on the nature of your workload.

There are no general guidelines to determine whether a particular workload should use column-organized tables. This should be examined on a case by case basis. In general, the same guidelines for performance in non-massively parallel processing (MPP) environments apply in MPP environments. In certain scenarios, vector processing of column data in a partitioned database environment provides significant improvements to storage, query performance, and ease of use through simplified design and tuning.

If you already have column-organized tables, but not in a massively parallel processing (MPP) environment, consider moving to an MPP environment. The following are some of the benefits of column-organized tables in an MPP environment:

- Column-organized table storage capacity can be increased. The maximum column-organized table size in a non-MPP environment is 64TB of compressed data . A partitioned database environment distributes the table across additional database members. Each new member allows the maximum table size to increase.
- The columnar workload's resources such as processors, memory and disks are increased by having multiple servers in parallel. Consider this factor of MPP only when the maximum resources of your non-MPP environment have already been applied. Consider whether or not the target workload will benefit from MPP parallelism, what other options exist to achieve performance objectives, and the cost and operational considerations of each option.

You must decide whether or not column-organized tables in an MPP environment will meet the performance goals of your target workload and be cost efficient. For example, MPP environments may increase operational costs since there will be multiple database partitions to monitor and maintain.

If you would like to create a new MPP environment with column-organized tables, ensure the prerequisites for column-organized tables in non-partitioned databases are met before doing so. If you plan to introduce column-organized tables into an existing MPP environment there are three approaches you can take:

- Replacing all existing tables
- Replacing some tables
- Creating new and independent tables

Prerequisities for adopting column-organized tables in partitioned databases

If you would like to introduce column-organized tables into a partitioned database, the database must meet the prerequisites for column-organized tables in non-partitioned databases.

If you are creating a new massively parallel processing (MPP) environment and plan to populate it with column-organized tables, follow the general system sizing and configuration recommendations for column-organized tables. Set **DB2_WORKLOAD** to ANALYTICS before you create the database. This automatically configures the database for columnar workloads.

If you want to introduce column organized tables into an existing MPP environment, follow the recommendations mentioned above. However, run the AUTOCONFIGURE utility with the APPLY NONE option to view the recommended configuration changes when running column-organized tables.

There are three approaches you can take to introducing column-organized tables into your partitioned database environments. These are the replacement, hybrid, or independent methods:

- Replacement
  - Replace all traditional row-organized tables with column-organized tables.
- Hybrid
  - Convert some of the row-organized tables to column-organized tables, or add new column-organized tables to an existing row table schema.
- Independent
  - Introduce a new, independent columnar workload with no interaction with any existing workload or row table schema.

Replacing all row-organized tables with column-organized tables in a partitioned database environment

Use the **db2convert** command to convert your row-organized tables into column-organized tables.

Before you begin

Switching to column-organized tables will affect the performance and use of resources of the system. Review the best practices for column-organized tables for recommendations on the memory, cores, and storage to provide for your desired dataset size.

About this task

There are several ways to convert row-organized tables to column-organized tables. One way is to use the **db2convert** command. If the database is not recoverable, simply run the **db2convert** command. If it is recoverable, you must run the command, make a backup of the database, and then rerun the command to complete the conversion.

Procedure

To replace all row-organized tables with column-organized tables in a non-recoverable database:

- Run the **db2convert** command on the desired database:

```
db2convert –d <name_of_database>
```

To replace all row-organized tables with column-organized tables in a recoverable database:

- Complete the first part of the conversion by running **db2convert** with the **-stopBeforeSwap** parameter, backup the database where you plan to store the converted tables, and then complete the conversion by running the command again:

```
db2convert -d <name_of_database> –stopBeforeSwap;

BACKUP DB <name_of_database> ONLINE TO <destination> ;
BACKUP DB <name_of_database> TABLESPACE <target_data_table_space> ONLINE TO <destination>;

db2convert -d <name_of_database> –continue;
```

  Where:

  – **–stopBeforeSwap** specifies that **db2convert** stops before it performs the SWAP phase of the ADMIN_MOVE_TABLE procedure and prompts you to complete an online backup operation before continuing.
  – **-continue** completes the conversion process.

Adopting a hybrid environment

Use the **db2convert** command to retain some row-organized tables, but replace the rest with column-organized tables.

Before you begin

- Ensure you implement best practices for column-organized table configuration
- To minimize the impact of the configuration changes on the row-table applications, set the **DEGREE** value for the row-table applications to 1 and the column-table applications to ANY. Use the MAXIMUM_DEGREE attribute to achieve this.

About this task

One of the ways to convert row-organized to column-organized tables is the **db2convert** command. If the database is recoverable, you must backup the database before completing the conversion, as in the replacement method.

Procedure

To replace only some row-organized tables with column-organized tables in a non-recoverable database:

Run the db2convert command on the database schema and tables you would like to be column-organized:

```
db2convert -d <name_of_database> -z <schema> –t <table_name>
```

Where:

- **-z** specifies the schema name of one or more tables to convert.
- **–t** specifies the unqualified name of the table to convert.

Creating an independent column-organized, partitioned environment

Introduce a columnar workload to co-exist with, but not interact with, existing row applications by loading a new dataset into the database.

Before you begin

- Review the best practices for column-organized table configuration

- To minimize the impact of the configuration changes on the row-table applications, set the **DEGREE** value for the row-table applications to 1 and the column-table applications to ANY. Use the MAXIMUM_DEGREE attribute to achieve this.

About this task

Create new column-organized tables in the database, then load the desired information into them.

Procedure

To introduce independent columnar workloads to the database:

1. Create a table by running the **CREATE TABLE** command with the ORGANIZE BY COLUMN option:

```
CREATE TABLE <new_table_name> (
 <column_1> <datatype>,
 <column_2> <datatype>,
 <column_3> <datatype>)
 ORGANIZE BY COLUMN;
```

2. Load data into the table using the **INGEST**, **INSERT**, **IMPORT**, or **LOAD** command:

```
LOAD FROM <file_with_data> of <file_with_data's_directory> REPLACE INTO <new_table_name>
```

Chapter 3. 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.

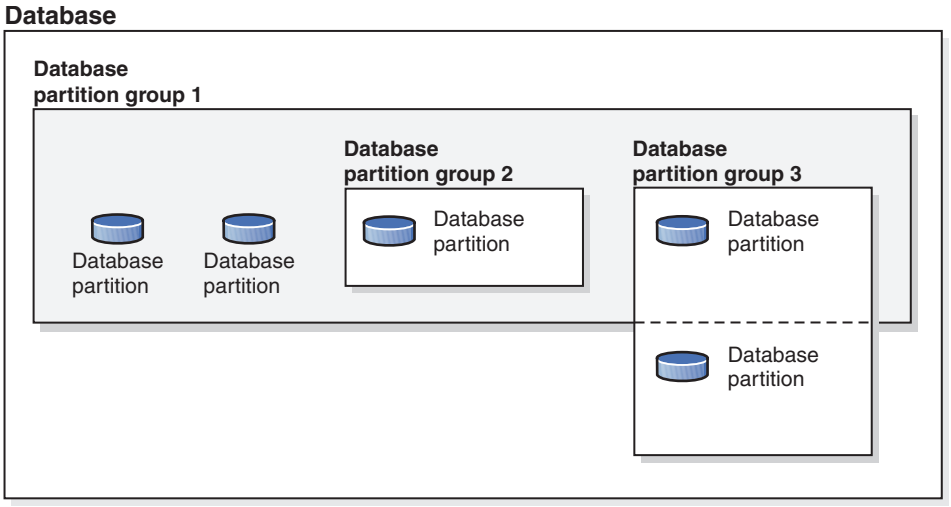
Chapter 4. Database partition groups

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 10 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.



(artname: 00000372.gif)
Figure 10. 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:

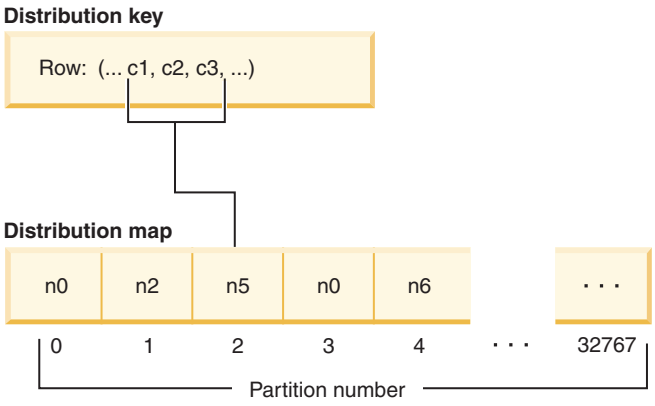
```
0 1 2 3 0 1 2 ...
```

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 11 on page 22 shows how the row with the distribution key value (c1, c2, c3) is mapped to number 2, which, in turn, references database partition n5.



(artname: 00000201.gif)
Figure 11. 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.

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.

Chapter 5. 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**.

Adding database partition servers to an instance (Windows)

On Windows, use the **db2ncrt** command to add a database partition server to an instance.

About this task

Note: Do not use the **db2ncrt** command if the instance already contains databases. Instead, use the **START DBM ADD DBPARTITIONNUM** command. This ensures that the database is correctly added to the new database partition server. **DO NOT EDIT** the db2nodes.cfg file, since changing the file might cause inconsistencies in the partitioned database environment.

The command has the following required parameters:

```
db2ncrt /n:partition_number
        /u:username,password
        /p:logical_port
```

/n:partition_number
The unique database partition number to identify the database partition server. The number can be from 1 to 999 in ascending sequence.

/u:username,password
The logon account name and password of the Db2 service.

/p:logical_port
The logical port number used for the database partition server if the logical port is not zero (0). If not specified, the logical port number assigned is 0.

The logical port parameter is only optional when you create the first database partition on a computer. If you create a logical database partition, you must specify this parameter and select a logical port number that is not in use. There are several restrictions:

- On every computer there must be a database partition server with a logical port 0.
- The port number cannot exceed the port range reserved for FCM communications in the services file in %SystemRoot%\system32\drivers\etc directory. For example, if you reserve a range of four ports for the current

instance, then the maximum port number would be 3 (ports 1, 2, and 3; port 0 is for the default logical database partition). The port range is defined when **db2icrt** is used with the /r:base_port, end_port parameter.

There are also several optional parameters:

/g:network_name
Specifies the network name for the database partition server. If you do not specify this parameter, Db2 uses the first IP address it detects on your system.

Use this parameter if you have multiple IP addresses on a computer and you want to specify a specific IP address for the database partition server. You can enter the network_name parameter using the network name or IP address.

/h:host_name
The TCP/IP host name that is used by FCM for internal communications if the host name is not the local host name. This parameter is required if you add the database partition server on a remote computer.

/i:instance_name
The instance name; the default is the current instance.

/m:computer_name
The computer name of the Windows workstation on which the database partition resides; the default name is the computer name of the local computer.

/o:instance_owning_computer
The computer name of the computer that is the instance-owning computer; the default is the local computer. This parameter is required when the **db2ncrt** command is invoked on any computer that is not the instance-owning computer.

For example, if you want to add a new database partition server to the instance TESTMPP (so that you are running multiple logical database partitions) on the instance-owning computer MYMACHIN, and you want this new database partition to be known as database partition 2 using logical port 1, enter:

```
db2ncrt /n:2 /p:1 /u:my_id,my_pword /i:TESTMPP
   /M:TEST /o:MYMACHIN
```

Setting up multiple logical partitions

There are several situations in which it is advantageous to have several database partition servers running on the same computer.

This means that the configuration can contain more database partitions than computers. In these cases, the computer is said to be running *multiple logical partitions* if they participate in the same instance. If they participate in different instances, this computer is not hosting multiple logical partitions.

With multiple logical partition support, you can choose from three types of configurations:

- A standard configuration, where each computer has only one database partition server
- A multiple logical partition configuration, where a computer has more than one database partition server

- A configuration where several logical partitions run on each of several computers

Configurations that use multiple logical partitions are useful when the system runs queries on a computer that has symmetric multiprocessor (SMP) architecture. The ability to configure multiple logical partitions on a computer is also useful if a computer fails. If a computer fails (causing the database partition server or servers on it to fail), you can restart the database partition server (or servers) on another computer using the **START DBM DBPARTITIONNUM** command. This ensures that user data remains available.

Another benefit is that multiple logical partitions can use SMP hardware configurations. In addition, because database partitions are smaller, you can obtain better performance when performing such tasks as backing up and restoring database partitions and table spaces, and creating indexes.

Configuring multiple logical partitions

There are two methods of configuring multiple logical partitions.

About this task

- Configure the logical partitions (database partitions) in the `db2nodes.cfg` file. You can then start all the logical and remote partitions with the **db2start** command or its associated API.

  Note: For Windows, you must use **db2ncrt** to add a database partition if there is no database in the system; or, **db2start addnode** command if there is one or more databases. Within Windows, the `db2nodes.cfg` file should never be manually edited.
- Restart a logical partition on another processor on which other logical partitions are already running. This allows you to override the hostname and port number specified for the logical partition in `db2nodes.cfg`.

To configure a logical database partition in `db2nodes.cfg`, you must make an entry in the file to allocate a logical port number for the database partition. Following is the syntax you should use:

```
nodenumber hostname logical-port netname
```

Note: For Windows, you must use **db2ncrt** to add a database partition if there is no database in the system; or, **db2start addnode** command if there is one or more databases. Within Windows, the `db2nodes.cfg` file should never be manually edited.

The format for the `db2nodes.cfg` file on Windows is different when compared to the same file on UNIX. On Windows, the column format is:

```
nodenumber hostname computername logical_port netname
```

Use the fully-qualified name for the hostname. The `/etc/hosts` file also should use the fully-qualified name. If the fully-qualified name is not used in the `db2nodes.cfg` file and in the`/etc/hosts` file, you might receive error message `SQL30082N RC=3`.

You must ensure that you define enough ports in the services file of the `etc` directory for FCM communications.

Enabling inter-partition query parallelism

Inter-partition parallelism occurs automatically based on the number of database partitions and the distribution of data across these database partitions.

About this task

You must modify configuration parameters to take advantage of parallelism within a database partition or within a non-partitioned database. For example, intra-partition parallelism can be used to take advantage of the multiple processors on a symmetric multi-processor (SMP) machine.

Procedure

- To enable parallelism when loading data:

  The load utility automatically makes use of parallelism, or you can use the following parameters on the **LOAD** command:

  – **CPU_PARALLELISM**

  – **DISK_PARALLELISM**

  In a partitioned database environment, inter-partition parallelism for data loading occurs automatically when the target table is defined on multiple database partitions. Inter-partition parallelism for data loading can be overridden by specifying **OUTPUT_DBPARTNUMS**. The load utility also intelligently enables database partitioning parallelism depending on the size of the target database partitions. **MAX_NUM_PART_AGENTS** can be used to control the maximum degree of parallelism selected by the load utility. Database partitioning parallelism can be overridden by specifying **PARTITIONING_DBPARTNUMS** when **ANYORDER** is also specified.
- To enable parallelism when creating an index:
  – The table must be large enough to benefit from parallelism
  – Multiple processors must be enabled on an SMP computer.
- To enable I/O parallelism when backing up a database or table space:
  – Use more than one target media.
  – Configure table spaces for parallel I/O by defining multiple containers, or use a single container with multiple disks, and the appropriate use of the **DB2_PARALLEL_IO** registry variable. If you want to take advantage of parallel I/O, you must consider the implications of what must be done before you define any containers. This cannot be done whenever you see a need; it must be planned for before you reach the point where you need to backup your database or table space.
  – Use the **PARALLELISM** parameter on the **BACKUP** command to specify the degree of parallelism.
  – Use the **WITH** num-buffers **BUFFERS** parameter on the **BACKUP** command to ensure that enough buffers are available to accommodate the degree of parallelism. The number of buffers should equal the number of target media you have plus the degree of parallelism selected plus a few extra.

    Also, use a backup buffer size that is:

    - As large as feasible. 4 MB or 8 MB (1024 or 2048 pages) is a good rule of thumb.
    - At least as large as the largest (extent size * number of containers) product of the table spaces being backed up.
- To enable I/O parallelism when restoring a database or table space:

– Use more than one source media.
– Configure table spaces for parallel I/O. You must decide to use this option before you define your containers. This cannot be done whenever you see a need; it must be planned for before you reach the point where you need to restore your database or table space.
– Use the **PARALLELISM** parameter on the **RESTORE** command to specify the degree of parallelism.
– Use the **WITH** num-buffers **BUFFERS** parameter on the **RESTORE** command to ensure that enough buffers are available to accommodate the degree of parallelism. The number of buffers should equal the number of target media you have plus the degree of parallelism selected plus a few extra.

  Also, use a restore buffer size that is:

  - As large as feasible. 4 MB or 8 MB (1024 or 2048 pages) is a good rule of thumb.
  - At least as large as the largest (extent size * number of containers) product of the table spaces being restored.
  - The same as, or an even multiple of, the backup buffer size.

Enabling intrapartition parallelism for queries

To enable intrapartition query parallelism, modify one or more database or database manager configuration parameters, precompile or bind options, or a special register. Alternatively, use the MAXIMUM DEGREE option on the CREATE or ALTER WORKLOAD statement, or the ADMIN_SET_INTRA_PARALLEL procedure to enable or disable intrapartition parallelism at the transaction level.

Before you begin

Use the following controls to specify what degree of intrapartition parallelism the optimizer is to use:

- CURRENT DEGREE special register (for dynamic SQL)
- DEGREE bind option (for static SQL)
- **dft_degree** database configuration parameter (provides the default value for the previous two parameters)

Use the following controls to limit the degree of intrapartition parallelism at run time. The runtime settings override the optimizer settings.

- **max_querydegree** database manager configuration parameter
- SET RUNTIME DEGREE command
- MAXIMUM DEGREE workload option

Use 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 statement (or ALTER WORKLOAD statement) option that specifies the maximum runtime degree of parallelism for a workload.

For example, suppose that `bank_trans` is a packaged application that mainly executes short OLTP transactions, and `bank_report` is another packaged application that runs complex queries to generate a business intelligence (BI) report. Neither application can be modified, and both are bound with degree 4 to the database. While `bank_trans` is running, it is assigned to workload `trans`, which disables intrapartition parallelism. This OLTP application will run without any performance degradation associated with intrapartition parallelism overhead. While `bank_report` is running, it is assigned to workload `bi`, which enables intrapartition parallelism and specifies a maximum runtime degree of 8. Because the compilation degree for the package is 4, the static SQL statements in this application run with only a degree of 4. If this BI application contains dynamic SQL statements, and the CURRENT DEGREE special register is set to 16, these statements run with a degree of 8.

```
connect to sample;

create workload trans
  applname('bank_trans')
  maximum degree 1
  enable;

create workload bi
  applname('bank_report')
  maximum degree 8
  enable;

connect reset;
```

ADMIN_SET_INTRA_PARALLEL

Procedure that enables or disables intrapartition parallelism for a database application. Although the procedure is called in the current transaction, it takes effect starting with the next transaction. For example, assume that the following code is part of the `demoapp` application, which uses the ADMIN_SET_INTRA_PARALLEL procedure with both static and dynamic SQL statements:

```
EXEC SQL CONNECT TO prod;

// Disable intrapartition parallelism:
EXEC SQL CALL SYSPROC.ADMIN_SET_INTRA_PARALLEL('NO');
// Commit so that the effect of this call
// starts in the next statement:
EXEC SQL COMMIT;

// All statements in the next two transactions run
// without intrapartition parallelism:
strcpy(stmt, "SELECT deptname FROM org");
EXEC SQL PREPARE rstmt FROM :stmt;
EXEC SQL DECLARE c1 CURSOR FOR rstmt;
EXEC SQL OPEN c1;
EXEC SQL FETCH c1 INTO :deptname;
EXEC SQL CLOSE c1;
...
// New section for this static statement:
EXEC SQL SELECT COUNT(*) INTO :numRecords FROM org;
...
EXEC SQL COMMIT;
```

```
// Enable intrapartition parallelism:
EXEC SQL CALL SYSPROC.ADMIN_SET_INTRA_PARALLEL('YES');
// Commit so that the effect of this call
// starts in the next statement:
EXEC SQL COMMIT;

strcpy(stmt, "SET CURRENT DEGREE='4'");
// Set the degree of parallelism to 4:
EXEC SQL EXECUTE IMMEDIATE :stmt;

// All dynamic statements in the next two transactions
// run with intrapartition parallelism and degree 4:
strcpy(stmt, "SELECT deptname FROM org");
EXEC SQL PREPARE rstmt FROM :stmt;
EXEC SQL DECLARE c2 CURSOR FOR rstmt;
EXEC SQL OPEN c2;
EXEC SQL FETCH c2 INTO :deptname;
EXEC SQL CLOSE c2;
...
// All static statements in the next two transactions
// run with intrapartition parallelism and degree 2:
EXEC SQL SELECT COUNT(*) INTO :numRecords FROM org;
...
EXEC SQL COMMIT;
```

The degree of intrapartition parallelism for dynamic SQL statements is specified through the CURRENT DEGREE special register, and for static SQL statements, it is specified through the DEGREE bind option. The following commands are used to prepare and bind the `demoapp` application:

```
connect to prod;
prep demoapp.sqc bindfile;
bind demoapp.bnd degree 2;
...
```

Chapter 6. Adding database partitions in partitioned database environments

You can add database partitions to the partitioned database system either when it is running, or when it is stopped. Because adding a new server can be time consuming, you might want to do it when the database manager is already running.

Use the **ADD DBPARTITIONNUM** command to add a database partition to a system. This command can be invoked in the following ways:

- As an option on the **START DBM** command
- With the **ADD DBPARTITIONNUM** command
- With the sqleaddn API
- With the sqlepstart API

If your system is stopped, use the **START DBM** command. If it is running, you can use any of the other choices.

When you use the **ADD DBPARTITIONNUM** command to add a new database partition to the system, all existing databases in the instance are expanded to the new database partition. You can also specify which containers to use for temporary table spaces for the databases. The containers can be:

- The same as those defined for the catalog partition for each database. (This is the default.)
- The same as those defined for another database partition.
- Not created at all. You must use the ALTER TABLESPACE statement to add temporary table space containers to each database before the database can be used.

Note: Any uncataloged database is not recognized when adding a new database partition. The uncataloged database will not be present on the new database partition. An attempt to connect to the database on the new database partition returns the error message SQL1013N.

You cannot use a database on the new database partition to contain data until one or more database partition groups are altered to include the new database partition.

You cannot change from a single-partition database to a multi-partition database by adding a database partition to your system. This is because the redistribution of data across database partitions requires a distribution key on each affected table. The distribution keys are automatically generated when a table is created in a multi-partition database. In a single-partition database, distribution keys can be explicitly created with the CREATE TABLE or ALTER TABLE SQL statements.

Note: If no databases are defined in the system and you are running Enterprise Server Edition on a UNIX operating system, edit the `db2nodes.cfg` file to add a new database partition definition; do not use any of the procedures described, as they apply only when a database exists.

Windows Considerations: If you are using Enterprise Server Edition on a Windows operating system and have no databases in the instance, use the **db2ncrt** command to scale the database system. If, however, you already have databases, use the **START DBM ADD DBPARTITIONNUM** command to ensure that a database partition is created for each existing database when you scale the system. On Windows operating systems, do not manually edit the database partition configuration file (`db2nodes.cfg`), because this can introduce inconsistencies to the file.

Adding an online database partition

You can add new database partitions that are online to a partitioned database environment while it is running and while applications are connected to databases.

Procedure

To add an online database partition to a running database manager using the command line:

1. On any existing database partition, run the **START DBM** command.

   On all platforms, specify the new database partition values for **DBPARTITIONNUM**, **ADD DBPARTITIONNUM**, **HOSTNAME**, **PORT**, and **NETNAME** parameters. On the Windows platform, you also specify the **COMPUTER**, **USER**, and **PASSWORD** parameters.

   You can also specify the source for any temporary table space container definitions that must be created with the databases. If you do not provide table space information, temporary table space container definitions are retrieved from the catalog partition for each database.

   For example, to add three new database partitions to an existing database, issue the following commands:

   ```
   START DBM DBPARTITIONNUM 3 ADD DBPARTITIONNUM HOSTNAME HOSTNAME3
    PORT PORT3;
   START DBM DBPARTITIONNUM 4 ADD DBPARTITIONNUM HOSTNAME HOSTNAME4
    PORT PORT4;
   START DBM DBPARTITIONNUM 5 ADD DBPARTITIONNUM HOSTNAME HOSTNAME5
    PORT PORT5;
   ```

2. Optional: Alter the database partition group to incorporate the new database partition. This action can also be an option when redistributing the data to the new database partition.
3. Optional: Redistribute data to the new database partition. This action is not really optional if you want to take advantage of the new database partitions. You can also include the alter database partition group option as part of the redistribution operation. Otherwise, altering the database partition group to incorporate the new database partitions must be done as a separate action before redistributing the data to the new database partition.
4. Optional: Back up all databases on the new database partition. Although optional, this would be helpful to have for the new database partition and for the other database partitions particularly if you redistributed the data across both the old and the new database partitions.

Restrictions when working online to add a database partition

The status of the new database partition following its addition to the instance depends on the status of the original database partition. Applications may or may not be aware of the new database partition following its addition to the instance if the application uses WITH HOLD cursors.

When adding a new database partition to a single-partition database instance:

- If the original database partition is up when the database partition is added, then the new database partition is down when the add database partition operation completes.
- If the original database partition is down when the database partition is added, then the new database partition is up when the add database partition operation completes.

Applications using WITH HOLD cursors that are started before the add database partition operation runs are not aware of the new database partition when the add database partition operation completes. If the WITH HOLD cursors are closed before the add database partition operation runs, then applications are aware of the new database partition when the add database partition operation completes

Adding a database partition offline (Linux and UNIX)

You can add new database partitions that are offline to a partitioned database system. The newly added database partition becomes available to all databases when the database manager is started again.

Before you begin

- Install the new server if it does not exist before you can create a database partition on it.
- Make the executables accessible using shared filesystem mounts or local copies.
- Synchronize operating system files with those on existing processors.
- Ensure that the `sqllib` directory is accessible as a shared file system.
- Ensure that the relevant operating system parameters (such as the maximum number of processes) are set to the appropriate values.
- Register the host name with the name server or in the `hosts` file in the `/etc` directory on all database partitions. The host name for the computer must be registered in `.rhosts` to run remote commands using **rsh** or **rah**.
- Set the default value of the **DB2_FORCE_OFFLINE_ADD_PARTITION** registry variable to `TRUE` to enforce that the added database partitions is offline.

Procedure

- To add a database partition to a stopped partitioned database server using the command line:
  1. Issue **STOP DBM** to stop all the database partitions.
  2. Run the **ADD DBPARTITIONNUM** command on the new server.

     A database partition is created locally for every database that exists in the system. The database parameters for the new database partitions are set to the default value, and each database partition remains empty until you move data to it. Update the database configuration parameter values to match those on the other database partitions.
  3. Run the **START DBM** command to start the database system. Note that the database partition configuration file (`db2nodes.cfg`) has already been updated by the database manager to include the new server during the installation of the new server.
  4. Update the configuration file on the new database partition as follows:
     a. On any existing database partition, run the **START DBM** command.

Specify the new database partition values for **DBPARTITIONNUM**, **ADD DBPARTITIONNUM**, **HOSTNAME**, **PORT**, and **NETNAME** parameters as well as the **COMPUTER**, **USER**, and **PASSWORD** parameters.

You can also specify the source for any temporary table space container definitions that must be created with the databases. If you do not provide table space information, temporary table space container definitions are retrieved from the catalog partition for each database.

For example, to add three new database partitions to an existing database, issue the following commands:

```
START DBM DBPARTITIONNUM 3 ADD DBPARTITIONNUM HOSTNAME HOSTNAME3
 PORT PORT3;
START DBM DBPARTITIONNUM 4 ADD DBPARTITIONNUM HOSTNAME HOSTNAME4
 PORT PORT4;
START DBM DBPARTITIONNUM 5 ADD DBPARTITIONNUM HOSTNAME HOSTNAME5
 PORT PORT5;
```

When the **START DBM** command is complete, the new server is stopped.

b. Stop the entire database manager by running the **STOP DBM** command.

When you stop all the database partitions in the system, the node configuration file is updated to include the new database partitions. The node configuration file is not updated with the new server information until **STOP DBM** is executed. This ensures that the **ADD DBPARTITIONNUM** command, which is called when you specify the **ADD DBPARTITIONNUM** parameter to the **START DBM** command, runs on the correct database partition. When the utility ends, the new server partitions are stopped.

5. Start the database manager by running the **START DBM** command.

   The newly added database partition is now started with the rest of the system.

   When all the database partitions in the system are running, you can run system-wide activities, such as creating or dropping a database.

   Note: You might have to issue the **START DBM** command twice for all database partition servers to access the new `db2nodes.cfg` file.

6. Optional: Alter the database partition group to incorporate the new database partition. This action might also be an option when redistributing the data to the new database partition.
7. Optional: Redistribute data to the new database partition. This action is not really optional if you want to take advantage of the new database partition. You can also include the alter database partition group option as part of the redistribution operation. Otherwise, altering the database partition group to incorporate the new database partition must be done as a separate action before redistributing the data to the new database partition.
8. Optional: Back up all databases on the new database partition. Although optional, this would be helpful to have for the new database partition and for the other database partitions particularly if you redistributed the data across both the old and the new database partitions.

- You can also update the configuration file manually, as follows:
  1. Edit the `db2nodes.cfg` file and add the new database partition to it.
  2. Issue the following command to start the new database partition: `START DBM DBPARTITIONNUM` *`partitionnum`*

     Specify the number you are assigning to the new database partition as the value of *partitionnum*.

3. If the new server is to be a logical partition (that is, it is not database partition 0), use **db2set** command to update the **DBPARTITIONNUM** registry variable. Specify the number of the database partition you are adding.
4. Run the **ADD DBPARTITIONNUM** command on the new database partition.

   This command creates a database partition locally for every database that exists in the system. The database parameters for the new database partitions are set to the default value, and each database partition remains empty until you move data to it. Update the database configuration parameter values to match those on the other database partitions.
5. When the **ADD DBPARTITIONNUM** command completes, issue the **START DBM** command to start the other database partitions in the system.

   Do not perform any system-wide activities, such as creating or dropping a database, until all database partitions are successfully started.

Adding a database partition offline (Windows)

You can add new database partitions to a partitioned database system while it is stopped. The newly added database partition becomes available to all databases when the database manager is started again.

Before you begin

- You must install the new server before you can create a database partition on it.
- Set the default value of the **DB2_FORCE_OFFLINE_ADD_PARTITION** registry variable to TRUE to enforce that any added database partitions is offline.

Procedure

To add a database partition to a stopped partitioned database server using the command line:

1. Issue **STOP DBM** to stop all the database partitions.
2. Run the **ADD DBPARTITIONNUM** command on the new server.

   A database partition is created locally for every database that already exists in the system. The database parameters for the new database partitions are set to the default value, and each database partition remains empty until you move data to it. Update the database configuration parameter values to match those on the other database partitions.
3. Run the **START DBM** command to start the database system. Note that the database partition configuration file has already been updated by the database manager to include the new server during the installation of the new server.
4. Update the configuration file on the new database partition as follows:
   a. On any existing database partitions, run the **START DBM** command.

      Specify the new database partition values for **DBPARTITIONNUM**, **ADD DBPARTITIONNUM**, **HOSTNAME**, **PORT**, and **NETNAME** parameters as well as the **COMPUTER**, **USER**, and **PASSWORD** parameters.

      You can also specify the source for any temporary table space container definitions that need to be created with the databases. If you do not provide table space information, temporary table space container definitions are retrieved from the catalog partition for each database.

      For example, to add three new database partitions to an existing database, issue the following commands:

      ```
      START DBM DBPARTITIONNUM 3 ADD DBPARTITIONNUM HOSTNAME HOSTNAME3
       PORT PORT3;
      ```

```
START DBM DBPARTITIONNUM 4 ADD DBPARTITIONNUM HOSTNAME HOSTNAME4
 PORT PORT4;
START DBM DBPARTITIONNUM 5 ADD DBPARTITIONNUM HOSTNAME HOSTNAME5
 PORT PORT5;
```

When the **START DBM** command is complete, the new server is stopped.

b. Stop the database manager by running the **STOP DBM** command.

When you stop all the database partitions in the system, the node configuration file is updated to include the new database partitions. The node configuration file is not updated with the new server information until **STOP DBM** is executed. This ensures that the **ADD DBPARTITIONNUM** command, which is called when you specify the **ADD DBPARTITIONNUM** parameter to the **START DBM** command, runs on the correct database partitions. When the utility ends, the new server partitions are stopped.

5. Start the database manager by running the **START DBM** command.

   The newly added database partitions are now started with the rest of the system.

   When all the database partitions in the system are running, you can run system-wide activities, such as creating or dropping a database.

   Note: You might have to issue the **START DBM** command twice for all database partition servers to access the new `db2nodes.cfg` file.
6. Optional: Alter the database partition group to incorporate the new database partition. This action could also be an option when redistributing the data to the new database partition.
7. Optional: Redistribute data to the new database partition. This action is not really optional if you want to take advantage of the new database partition. You can also include the alter database partition group option as part of the redistribution operation. Otherwise, altering the database partition group to incorporate the new database partition must be done as a separate action before redistributing the data to the new database partition.
8. Optional: Back up all databases on the new database partition. Although optional, this would be helpful to have for the new database partition and for the other database partitions particularly if you have redistributed the data across both the old and the new database partitions.

Error recovery when adding database partitions

Adding database partitions does not fail as a result of nonexistent buffer pools, because the database manager creates system buffer pools to provide default automatic support for all buffer pool page sizes.

However, if one of these system buffer pools is used, performance might be seriously affected, because these buffer pools are very small. If a system buffer pool is used, a message is written to the administration notification log. System buffer pools are used in database partition addition scenarios in the following circumstances:

- You add database partitions to a partitioned database environment that has one or more system temporary table spaces with a page size that is different from the default of 4 KB. When a database partition is created, only the IBMDEFAULTDP buffer pool exists, and this buffer pool has a page size of 4 KB.

  Consider the following examples:

  1. You use the **START DBM** command to add a database partition to the current multi-partition database:

```
START DBM DBPARTITIONNUM 2 ADD DBPARTITIONNUM HOSTNAME newhost PORT 2
```

2. You use the **ADD DBPARTITIONNUM** command after you manually update the `db2nodes.cfg` file with the new database partition description.

One way to prevent these problems is to specify the WITHOUT TABLESPACES clause on the **ADD DBPARTITIONNUM** or the **START DBM** commands. After doing this, use the CREATE BUFFERPOOL statement to create the buffer pools using the appropriate SIZE and PAGESIZE values, and associate the system temporary table spaces to the buffer pool using the ALTER TABLESPACE statement.

- You add database partitions to an existing database partition group that has one or more table spaces with a page size that is different from the default page size, which is 4 KB. This occurs because the non-default page-size buffer pools created on the new database partition have not been activated for the table spaces.

  Note: In previous versions, this command used the NODEGROUP keyword instead of the DATABASE PARTITION GROUP keywords.

  Consider the following example:

  – You use the ALTER DATABASE PARTITION GROUP statement to add a database partition to a database partition group, as follows:

  ```
  START DBM
  CONNECT TO mpp1
  ALTER DATABASE PARTITION GROUP ng1 ADD DBPARTITIONNUM (2)
  ```

  One way to prevent this problem is to create buffer pools for each page size and then to reconnect to the database before issuing the following ALTER DATABASE PARTITION GROUP statement:

  ```
  START DBM
  CONNECT TO mpp1
  CREATE BUFFERPOOL bp1 SIZE 1000 PAGESIZE 8192
  CONNECT RESET
  CONNECT TO mpp1
  ALTER DATABASE PARTITION GROUP ng1 ADD DBPARTITIONNUM (2)
  ```

Note: If the database partition group has table spaces with the default page size, message SQL1759W is returned.

Chapter 7. 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.

Chapter 8. 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.

Chapter 9. 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.

Listing database partition servers in an instance (Windows)

On Windows, use the **db2nlist** command to obtain a list of database partition servers that participate in an instance.

About this task

The command is used as follows:

```
db2nlist
```

When using this command as shown, the default instance is the current instance (set by the **DB2INSTANCE** environment variable). To specify a particular instance, you can specify the instance using:

```
db2nlist /i:instName
```

where *instName* is the particular instance name you want.

You can also optionally request the status of each database partition server by using:

```
db2nlist /s
```

The status of each database partition server might be one of: starting, running, stopping, or stopped.

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

If you are running multiple logical database partition servers on one computer, your db2nodes.cfg file contains multiple entries for that computer.

About this task

In this situation, the **rah** command needs to know whether you want the command to be executed only once on each computer or once for each logical database partition listed in the db2nodes.cfg file. Use the **rah** command to specify computers. Use the **db2_all** command to specify logical database partitions.

Note: On Linux and UNIX operating systems, if you specify computers, **rah** normally eliminates duplicates from the computer list, with the following exception: if you specify logical database partitions, **db2_all** prepends the following assignment to your command:

```
export DB2NODE=nnn  (for Korn shell syntax)
```

where *nnn* is the database partition number taken from the corresponding line in the db2nodes.cfg file, so that the command is routed to the desired database partition server.

When specifying logical database partitions, you can restrict the list to include all logical database partitions except one, or specify only one using the <<-*nnn*< and <<+*nnn*< prefix sequences. You might want to do this if you want to run a command to catalog the database partition first, and when that has completed, run the same command at all other database partition servers, possibly in parallel. This is usually required when running the **RESTART DATABASE** command. You need to know the database partition number of the catalog partition to do this.

If you execute **RESTART DATABASE** using the **rah** command, duplicate entries are eliminated from the list of computers. However if you specify the " prefix, then duplicates are not eliminated, because it is assumed that use of the " prefix implies sending to each database partition server, rather than to each computer.

Specifying the list of machines in a partitioned database environment

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

About this task

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

Procedure

To override the list of computers in db2nodes.cfg:

- Specify a path name to the file that contains the list of computers by exporting (on Linux and UNIX operating systems) or setting (on Windows) the environment variable **RAHOSTFILE**.

- Specify the list explicitly, as a string of names separated by spaces, by exporting (on Linux and UNIX operating systems) or setting (on Windows) the environment variable **RAHOSTLIST**.

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

Changing the database configuration across multiple database partitions

When you have a database that is distributed across more than one database partition, the database configuration file should be the same on all database partitions.

About this task

Consistency is required since the SQL compiler compiles distributed SQL statements based on information in the database partition configuration file and creates an access plan to satisfy the needs of the SQL statement. Maintaining different configuration files on database partitions could lead to different access plans, depending on which database partition the statement is prepared. Use **db2_all** to maintain the configuration files across all database partitions.

Adding containers to SMS table spaces on database partitions

You can add a container to an SMS table space only on a database partition that currently has no containers.

Procedure

To add a container to an SMS table space using the command line, enter the following:

```
ALTER TABLESPACE name
   ADD ('path')
   ON DBPARTITIONNUM (database_partition_number)
```

The database partition specified by number, and every partition in the range of database partitions, must exist in the database partition group on which the table space is defined. A *database_partition_number* might only appear explicitly or within a range in exactly one *db-partitions-clause* for the statement.

Example

The following example shows how to add a new container to database partition number 3 of the database partition group used by table space “plans” on a UNIX operating system:

```
ALTER TABLESPACE plans
   ADD ('/dev/rhdisk0')
   ON DBPARTITIONNUM (3)
```

Using database partition expressions

In most cases, you must use the same storage paths for each partition in a partitioned database environment, and all of the storage paths must exist before you issue a statement. One exception is when you use database partition expressions within the storage path.

Doing this allows the database partition number to be reflected in the storage path such that the resulting path name is different on each partition.

You can specify a database partition expression for container string syntax when creating either SMS or DMS containers. You typically specify the database partition expression when using multiple logical database partitions in a partitioned database system. The expression ensures that container names are unique across database partition servers. If you specify an expression, the database partition number is part of the container name or, if you specify additional arguments, the result of the argument is part of the container name.

Important: The SMS table space type has been deprecated in Version 10.1 for user-defined permanent table spaces and might be removed in a future release. The SMS table space type is not deprecated for catalog and temporary table spaces. For more information, see "SMS permanent table spaces have been deprecated" at http://www.ibm.com/support/knowledgecenter/SSEPGG_10.1.0/com.ibm.db2.luw.wn.doc/doc/i0058748.html.

Important: Starting with Version 10.1 Fix Pack 1, the DMS table space type is deprecated for user-defined permanent table spaces and might be removed in a future release. The DMS table space type is not deprecated for catalog and temporary table spaces. For more information, see "DMS permanent table spaces have been deprecated" at http://www.ibm.com/support/knowledgecenter/SSEPGG_10.1.0/com.ibm.db2.luw.wn.doc/doc/i0060577.html.

Use the argument " $N" ([blank]$N) to indicate a database partition expression. You can use a database partition expression anywhere in the storage path name, and you can specify multiple database partition expressions. Terminate the database partition expression with a space character; whatever follows the space is appended to the storage path name after the database partition expression is evaluated. If there is no space character in the storage path name after the database partition expression, it is assumed that the rest of the string is part of the expression. If you specify a number before the N argument, ($[number]N), the partition number is formatted with leading zeros.

You must specify the argument by using one of the forms in the following table. Operators are evaluated from left to right. A percent sign (%) represents the modulus operator. The database partition number in the following examples is assumed to be 10.

Table 2. Database partition expressions

Syntax	Example	Value
[blank]$N	"$N"	10
[blank]$[number]N	"$4N"	0010
[blank]$N+[number]	"$N+100"	110
[blank]$N%[number]	"$N%5"	0
[blank]$N+[number] %[number]	"$N+1%5"	1
[blank]$N %[number]+[number]	"$N%4+2"	4

If you specified a storage path by using a database partition expression, you must use the same storage path string, including the database partition expression, to

drop the path. This path string is in the DB_STORAGE_PATH_WITH_DPE field of the ADMIN_GET_STORAGE_PATHS table function. This element is not shown if you did not include a database partition expression in the original path.

Examples

1. On a system with two database partitions:

```
CREATE TABLESPACE TS1 MANAGED BY DATABASE USING
     (device '/dev/rcont $N' 20000)
```

   The following containers are created:

```
     /dev/rcont0 - on database partition 0
     /dev/rcont1 - on database partition 1
```

2. On a system with three database partitions:

```
ALTER STOGROUP IBMSTOGROUP ADD '/DB2/path $N'
```

   The following paths are added:

```
/DB2/path0 - on database partition 0
/DB2/path1 - on database partition 1
/DB2/path2 - on database partition 2
```

3. On a system with four database partitions:

```
   CREATE TABLESPACE TS2 MANAGED BY DATABASE USING
     (file '/DB2/containers/TS2/container $N+100' 10000)
```

   The following containers are created:

```
     /DB2/containers/TS2/container100 - on database partition 0
     /DB2/containers/TS2/container101 - on database partition 1
     /DB2/containers/TS2/container102 - on database partition 2
     /DB2/containers/TS2/container103 - on database partition 3
```

4. On a system with two database partitions:

```
   CREATE TABLESPACE TS3 MANAGED BY SYSTEM USING
     ('/TS3/cont $N%2','/TS3/cont $N%2+2')
```

   The following containers are created:

```
     /TS3/cont0 - On database partition 0
     /TS3/cont2 - On database partition 0
     /TS3/cont1 - On database partition 1
     /TS3/cont3 - On database partition 1
```

5. If there are 10 database partitions, the containers use the following syntax:

```
     '/dbdir/node $N /cont1'
     '/ $N+1000 /file1'
     ' $N%10 /container'
     '/dir/ $N2000 /dmscont'
```

   The containers are created as:

```
     '/dbdir/node5/cont1'
     '/1005/file1'
     '5/container'
     '/dir/2000/dmscont'
```

Changing database partitions (Windows)

On Windows, use the **db2nchg** command to change database partitions.

About this task

- Move the database partition from one computer to another.
- Change the TCP/IP host name of the computer.

  If you are planning to use multiple network adapters, you must use this command to specify the TCP/IP address for the "netname" field in the `db2nodes.cfg` file.
- Use a different logical port number.
- Use a different name for the database partition server.

The command has the following required parameter:

```
db2nchg /n:node_number
```

The parameter **/n:** is the number of the database partition server that you want to change. This parameter is required.

Optional parameters include:

/i:*instance_name*
Specifies the instance that this database partition server participates in. If you do not specify this parameter, the default is the current instance.

/u:*username*,*password*
Changes the logon account name and password for the Db2 database service. If you do not specify this parameter, the logon account and password remain the same.

/p:*logical_port*
Changes the logical port for the database partition server. This parameter must be specified if you move the database partition server to a different computer. If you do not specify this parameter, the logical port number remains unchanged.

/h:*host_name*
Changes the TCP/IP host name used by FCM for internal communications. If you do not specify this parameter, the host name is unchanged.

/m:*computer_name*
Moves the database partition server to another computer. The database partition server can be moved only if there are no existing databases in the instance.

/g:*network_name*
Changes the network name for the database partition server.

Use this parameter if you have multiple IP addresses on a computer and you want to use a specific IP address for the database partition server. You can enter the *network_name* using the network name or the IP address.

For example, to change the logical port assigned to database partition 2, which participates in the instance TESTMPP, to use the logical port 3, enter the following command:

```
db2nchg /n:2 /i:TESTMPP /p:3
```

The Db2 database manager provides the capability of accessing Db2 database system registry variables at the instance level on a remote computer.

Currently, Db2 database system registry variables are stored in three different levels: computer or global level, instance level, and database partition level. The

registry variables stored at the instance level (including the database partition level) can be redirected to another computer by using **DB2REMOTEPREG**. When **DB2REMOTEPREG** is set, the Db2 database manager accesses the Db2 database system registry variables from the computer pointed to by **DB2REMOTEPREG**. The **db2set** command would appear as:

```
db2set DB2REMOTEPREG=remote_workstation
```

where *remote_workstation* is the remote workstation name.

Note:

- Care must be taken in setting this option since all Db2 database instance profiles and instance listings will be located on the specified remote computer name.
- If your environment includes users from domains, ensure that the logon account associated with the Db2 instance service is a domain account. This ensures that the Db2 instance has the appropriate privileges to enumerate groups at the domain level.

This feature might be used in combination with setting **DBINSTPROF** to point to a remote LAN drive on the same computer that contains the registry.

Redistributing data in a database partition group

To create an effective redistribution plan for your database partition group and redistribute your data, issue the **REDISTRIBUTE DATABASE PARTITION GROUP** command or call the 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

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 (or remsh for HP-UX). If **DB2RSHCMD** is not set, rsh (or remsh for HP-UX) is used. The ssh remote shell program is used to prevent the transmission of passwords in clear text in UNIX operating system environments.

If a command runs on one database partition server and you want it to run on all of them, use **db2_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.

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.

Specifying the rah and db2_all commands

You can specify **rah** command from the command line as the parameter, or in response to the prompt if you do not specify any parameter.

Use the prompt method if the command contains the following special characters:

```
| & ; < > ( )  { } [ ] unsubstituted $
```

If you specify the command as the parameter on the command line, you must enclose it in double quotation marks if it contains any of the special characters just listed.

Note: On Linux and UNIX operating systems, the command is added to your command history just as if you typed it at the prompt.

All special characters in the command can be entered normally (without being enclosed in quotation marks, except for \). If you require a \ in your command, you must type two backslashes (\\).

Note: On Linux and UNIX operating systems, if you are not using a Korn shell, all special characters in the command can be entered normally (without being enclosed in quotation marks, except for ", \, unsubstituted $, and the single quotation mark (')). If you require one of these characters in your command, you must precede them by three backslashes (\\\). For example, if you require a \ in your command, you must type four backslashes (\\\\).

If you require a double quotation mark (") in your command, you must precede it by three backslashes, for example, \\\".

Note:

1. On Linux and UNIX operating systems, you cannot include a single quotation mark (') in your command unless your command shell provides some way of entering a single quotation mark inside a singly quoted string.
2. On Windows, you cannot include a single quotation mark (') in your command unless your command window provides some way of entering a single quotation mark inside a singly quoted string.

When you run any korn-shell shell-script that contains logic to read from stdin in the background, explicitly redirect stdin to a source where the process can read without getting stopped on the terminal (SIGTTIN message). To redirect stdin, you can run a script with the following form:

```
shell_script </dev/null &
```

if there is no input to be supplied.

In a similar way, always specify `</dev/null` when running **db2_all** in the background. For example:

```
db2_all ";run_this_command" </dev/null &
```

By doing this you can redirect stdin and avoid getting stopped on the terminal.

An alternative to this method, when you are not concerned about output from the remote command, is to use the "daemonize" option in the **db2_all** prefix:

```
db2_all ";daemonize_this_command" &
```

Running commands in parallel (Linux, UNIX)

By default, the command is run sequentially at each computer, but you can specify to run the commands in parallel using background rshells by prefixing the command with certain prefix sequences. If the rshell is run in the background, then each command puts the output in a buffer file at its remote computer.

Note: The information in this section applies to Linux and UNIX operating systems only.

This process retrieves the output in two pieces:

1. After the remote command completes.
2. After the rshell terminates, which might be later if some processes are still running.

The name of the buffer file is `/tmp/$USER/rahout` by default, but it can be specified by the environment variables $**RAHBUFDIR** or $**RAHBUFNAME**.

When you specify that you want the commands to be run concurrently, by default, this script prefixes an additional command to the command sent to all hosts to check that $**RAHBUFDIR** and $**RAHBUFNAME** are usable for the buffer file. It creates $**RAHBUFDIR**. To suppress this, export an environment variable `RAHCHECKBUF=no`. You can do this to save time if you know that the directory exists and is usable.

Before using **rah** to run a command concurrently at multiple computers:

- Ensure that a directory /tmp/$USER exists for your user ID at each computer. To create a directory if one does not exist, run:

  ```
  rah ")mkdir /tmp/$USER"
  ```

- Add the following line to your .kshrc (for Korn shell syntax) or .profile, and also type it into your current session:

  ```
  export RAHCHECKBUF=no
  ```

- Ensure that each computer ID at which you run the remote command has an entry in its .rhosts file for the ID which runs **rah**; and the ID which runs **rah** has an entry in its .rhosts file for each computer ID at which you run the remote command.

Monitoring rah processes (Linux, UNIX)

While any remote commands are still running or buffered output is still being accumulated, processes started by **rah** monitor activity to write messages to the terminal indicating which commands have not been run, and retrieve the buffered output.

About this task

Note: The information in this section applies to Linux and UNIX operating systems only.

The informative messages are written at an interval controlled by the environment variable **RAHWAITTIME**. Refer to the help information for details on how to specify this. All informative messages can be suppressed by exporting RAHWAITTIME=0.

The primary monitoring process is a command whose command name (as shown by the **ps** command) is **rahwaitfor**. The first informative message tells you the pid (process id) of this process. All other monitoring processes appear as **ksh** commands running the **rah** script (or the name of the symbolic link). If you want, you can stop all monitoring processes by the command:

```
kill pid
```

where *pid* is the process ID of the primary monitoring process. Do not specify a signal number. Leave the default of 15. This does not affect the remote commands at all, but prevents the automatic display of buffered output. Note that there might be two or more different sets of monitoring processes executing at different times during the life of a single execution of **rah**. However, if at any time you stop the current set, then no more are started.

If your regular login shell is not a Korn shell (for example /bin/ksh), you can use **rah**, but there are some slightly different rules on how to enter commands containing the following special characters:

```
"  unsubstituted $ '
```

For more information, type rah "?". Also, in a Linux or UNIX operating system, if the login shell at the ID which executes the remote commands is not a Korn shell, then the login shell at the ID which executes **rah** must also not be a Korn shell. (**rah** decides whether the shell of the remote ID is a Korn shell based on the local ID). The shell must not perform any substitution or special processing on a string enclosed in single quotation marks. It must leave it exactly as is.

Extension of the rah command to use tree logic (AIX and Solaris)

To enhance performance, rah has been extended to use tree_logic on large systems. That is, rah will check how many database partitions the list contains, and if that number exceeds a threshold value, it constructs a subset of the list and sends a recursive invocation of itself to those database partitions.

At those database partitions, the recursively invoked rah follows the same logic until the list is small enough to follow the standard logic (now the "leaf-of-tree" logic) of sending the command to all database partitions on the list. The threshold can be specified by the **RAHTREETHRESH** environment variable, or defaults to 15.

In the case of a multiple-logical-database partitions-per-physical-database partition system, **db2_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 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 3. 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

Table 3. Environment variables that control the **rah** *command (continued)*

Name	Meaning	Default
$**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
$**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**

Table 3. Environment variables that control the **rah** *command (continued)*

Name	Meaning	Default
$**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
```

Setting the default environment profile for rah on Windows

To set the default environment profile for the **rah** command, use a file called `db2rah.env`, which should be created in the instance directory.

About this task

Note: The information in this section applies to Windows only.

The file should have the following format:

```
; This is a comment line
DB2INSTANCE=instancename
DB2DBDFT=database
; End of file
```

You can specify all the environment variables that you need to initialize the environment for **rah**.

Determining problems with rah (Linux, UNIX)

This topic gives suggestions on how to handle some problems that you might encounter when you are running **rah**.

Note: The information in this section applies to Linux and UNIX operating systems only.

1. **rah** hangs (or takes a very long time)

   This problem might be caused because:

   - **rah** has determined that it needs to buffer output, and you did not export RAHCHECKBUF=no. Therefore, before running your command, **rah** sends a command to all computers to check the existence of the buffer directory, and to create it if it does not exist.
   - One or more of the computers where you are sending your command is not responding. The **rsh** command will eventually time out but the time-out interval is quite long, usually about 60 seconds.

2. You have received messages such as:

   - Login incorrect
   - Permission denied

   Either one of the computers does not have the ID running **rah** correctly defined in its /etc/hosts file, or the ID running **rah** does not have one of the computers correctly defined in its .rhosts file. If the **DB2RSHCMD** registry variable has been configured to use ssh, then the ssh clients and servers on each computer might not be configured correctly.

   Note: You might need to have greater security regarding the transmission of passwords in clear text between database partitions. This will depend on the remote shell program you are using. **rah** uses the remote shell program specified by the **DB2RSHCMD** registry variable. You can select between the two remote shell programs: ssh (for additional security), or rsh (or remsh for HP-UX). If this registry variable is not set, rsh (or remsh for HP-UX) is used.

3. When running commands in parallel using background remote shells, although the commands run and complete within the expected elapsed time at the computers, **rah** takes a long time to detect this and put up the shell prompt.

   The ID running **rah** does not have one of the computers correctly defined in its .rhosts file, or if the **DB2RSHCMD** registry variable has been configured to use ssh, then the ssh clients and servers on each computer might not be configured correctly.

4. Although **rah** runs fine when run from the shell command line, if you run **rah** remotely using rsh, for example,

   ```
   rsh 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.

Dropping database partitions

You can drop a database partition that is not being used by any database and free the computer for other uses.

Before you begin

Verify that the database partition is not in use by issuing the **DROP DBPARTITIONNUM VERIFY** command or the sqledrpn API.

- If you receive message SQL6034W (Database partition not used in any database), you can drop the database partition.
- If you receive message SQL6035W (Database partition in use by database), use the **REDISTRIBUTE DATABASE PARTITION GROUP** command to redistribute the data from the database partition that you are dropping to other database partitions from the database alias.

Also ensure that all transactions for which this database partition was the coordinator have all committed or rolled back successfully. This might require doing crash recovery on other servers. For example, if you drop the coordinator partition, and another database partition participating in a transaction crashed before the coordinator partition was dropped, the crashed database partition will not be able to query the coordinator partition for the outcome of any indoubt transactions.

Procedure

To drop a database partition using the command line:

Issue the **STOP DBM** command with the **DROP DBPARTITIONNUM** parameter to drop the database partition.
After the command completes successfully, the system is stopped. Then start the database manager with the **START DBM** command.

Dropping a database partition from an instance (Windows)

On Windows, use the **db2ndrop** command to drop a database partition server from an instance that has no databases. If you drop a database partition server, its database partition number can be reused for a new database partition server.

About this task

Exercise caution when you drop database partition servers from an instance. If you drop the instance-owning database partition server zero (0) from the instance, the instance becomes unusable. If you want to drop the instance, use the **db2idrop** command.

Note: Do not use the **db2ndrop** command if the instance contains databases. Instead, use the **STOP DBM DROP DBPARTITIONNUM** command. This ensures that the database is correctly removed from the database partition. **DO NOT EDIT** the `db2nodes.cfg` file, since changing the file might cause inconsistencies in the partitioned database environment.

If you want to drop a database partition that is assigned the logical port 0 from a computer that is running multiple logical database partitions, you must drop all the other database partitions assigned to the other logical ports before you can drop the database partition assigned to logical port 0. Each database partition server must have a database partition assigned to logical port 0.

The command has the following parameters:

```
db2ndrop /n:dbpartitionnum /i:instance_name
```

/n:*dbpartitionnum*
The unique database partition number (*dbpartitionnum*) to identify the database partition server. This is a required parameter. The number can be from zero (0) to 999 in ascending sequence. Recall that database partition zero (0) represents the instance-owning computer.

/i:instance_name

The instance name (*instance_name*). This is an optional parameter. If not given, the default is the current instance (set by the **DB2INSTANCE** registry variable).

Chapter 10. Redistributing data across database partitions

Redistributing data is a task that you might perform in a partitioned database environment after adding or removing database partitions or when an undesirable proportion of data is appearing on a particular partition so as to rebalance or reconfigure the distribution.

Data redistribution

Data redistribution is a database administration operation that can be performed to primarily move data within a partitioned database environment when partitions are added or removed. The goal of this operation is typically to balance the usage of storage space, improve database system performance, or satisfy other system requirements.

Data redistribution can be performed by using one of the following interfaces:

- **REDISTRIBUTE DATABASE PARTITION GROUP** command
- ADMIN_CMD built-in procedure
- STEPWISE_REDISTRIBUTE_DBPG built-in procedure
- 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.

Determining if data redistribution is needed

Determining the current data distribution for a database partition group or table can be helpful in determining if data redistribution is required. Details about the current data distribution can also be used to create a custom distribution map that specifies how to distribute data.

About this task

If a new database partition is added to a database partition group, or an existing database partition is dropped from a database partition group, perform data redistribution to balance data among all the database partitions.

If no database partitions have been added or dropped from a database partition group, then data redistribution is usually only indicated when there is an unequal distribution of data among the database partitions of the database partition group. Note that in some cases an unequal distribution of data can be desirable. For example, if some database partitions reside on a powerful machine, then it might be beneficial for those database partitions to contain larger volumes of data than other partitions.

Procedure

To determine if data redistribution is needed:

1. Get information about the current distribution of data among database partitions in the database partition group.

   Run the following query on the largest table (alternatively, a representative table) in the database partition group:

   ```
   SELECT DBPARTITIONNUM(column_name), COUNT(*) FROM table_name
         GROUP BY DBPARTITIONNUM(column_name)
         ORDER BY DBPARTITIONNUM(column_name) DESC
   ```

   Here, *column_name* is the name of the distribution key for table *table_name*.

   The output of this query shows how many records from *table_name* reside on each database partition. If the distribution of data among database partitions is not as desired, then proceed to the next step.
2. Get information about the distribution of data across hash partitions.

Run the following query with the same *column_name* and *table_name* that were used in the previous step:

```
SELECT HASHEDVALUE(column_name), COUNT(*) FROM table_name
      GROUP BY HASHEDVALUE(column_name)
      ORDER BY HASHEDVALUE(column_name) DESC
```

The output of this query can easily be used to construct the distribution file needed when the **USING DISTFILE** parameter in the **REDISTRIBUTE DATABASE PARTITION GROUP** command is specified. Refer to the **REDISTRIBUTE DATABASE PARTITION GROUP** command reference for a description of the format of the distribution file.

3. Optional: If the data requires redistribution, you can plan to do this operation during a system maintenance opportunity.

   When the **USING DISTFILE** parameter is specified, the **REDISTRIBUTE DATABASE PARTITION GROUP** command uses the information in the file to generate a new partition map for the database partition group. This operation results in a uniform distribution of data among database partitions.

   If a uniform distribution is not desired, you can construct your own target partition map for the redistribution operation. The target partition map can be specified by using the **USING TARGETMAP** parameter in the **REDISTRIBUTE DATABASE PARTITION GROUP** command.

Results

After doing this investigation, you will know if your data is uniformly distributed or not or if data redistribution is required.

Prerequisites for data redistribution

Before data redistribution can be performed successfully for a set of tables within a database partition group, certain prerequisites must be met.

The following is a list of mandatory prerequisites:

- Authorization to perform data redistribution from the supported data redistribution interface of choice.
- A significant amount of time during a period of low system activity in which to perform the redistribution operation.
- All tables containing data to be redistributed as part of a data redistribution operation must be in a NORMAL state. For example, tables cannot be in LOAD PENDING state or other inaccessible load table states. To check the states of tables, establish a connection to each partition in the database partition group and issue the **LOAD QUERY** command. The output of this command contains information about the state of the table. The documentation of the **LOAD QUERY** command explains the meaning of each of the table states and how to move tables from one state to another.
- All tables within the database partition being redistributed must have been defined with a distribution key. If a new database partition is added to a single-partition system, data redistribution cannot be performed until all of the tables within the partitions have a distribution key. For tables that were created using the CREATE TABLE statement and have definitions that do not contain a distribution key, you must alter the table by using the ALTER TABLE statement to add a distribution key before redistributing the data.

- Replicated materialized query tables contained in a database partition group must be dropped before you redistribute the data. Store a copy of the materialized query table definitions so that they can be recreated after data redistribution completes.
- If a non-uniform redistribution is desired a distribution map must be created as a target distribution map to be used a parameter to the redistribute interface.
- A backup of the database must be created by using the **BACKUP DATABASE** command. This backup is not a mandatory prerequisite however it is strongly recommended that it be done.
- A connection must be established to the database from the catalog database partition.
- Adequate space must be available to rebuild all indexes either during or after the data redistribution. The **INDEXING MODE** command parameter affects when the indexes are rebuilt.
- When the **NOT ROLLFORWARD RECOVERABLE** command parameter is specified, adequate space should be available for writing status information to control files used by IBM Service for problem determination. The control files are generated in the following paths and should be manually deleted when the data redistribution operation is complete:
  - On Linux and UNIX operating systems: **diagpath**/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.

Log space requirements for data redistribution

To successfully perform a data redistribution operation, adequate log file space must be allocated to ensure that data redistribution is not interrupted. Log space requirements are less of a concern when you specify the **NOT ROLLFORWARD RECOVERABLE** command parameter, since there is minimal logging during that type of data redistribution.

The quantity of log file space required depends on multiple factors including which options of the **REDISTRIBUTE DATABASE PARTITION GROUP** command are used.

When the redistribution is performed from any supported interface where the data redistribution is roll-forward recoverable:

- The log must be large enough to accommodate the INSERT and DELETE operations at each database partition where data is being redistributed. The heaviest logging requirements will be either on the database partition that will lose the most data, or on the database partition that will gain the most data.
- If you are moving to a larger number of database partitions, use the ratio of current database partitions to the new number of database partitions to estimate the number of INSERT and DELETE operations. For example, consider redistributing data that is uniformly distributed before redistribution. If you are moving from four to five database partitions, approximately twenty percent of the four original database partitions will move to the new database partition. This means that twenty percent of the DELETE operations will occur on each of the four original database partitions, and all of the INSERT operations will occur on the new database partition.
- Consider a nonuniform distribution of the data, such as the case in which the distribution key contains many NULL values. In this case, all rows that contain a NULL value in the distribution key move from one database partition under the old distribution scheme and to a different database partition under the new distribution scheme. As a result, the amount of log space required on those two database partitions increases, perhaps well beyond the amount calculated by assuming uniform distribution.
- The redistribution of each table is a single transaction. For this reason, when you estimate log space, you multiply the percentage of change, such as twenty percent, by the size of the largest table. Consider, however, that the largest table might be uniformly distributed but the second largest table, for example, might have one or more inflated database partitions. In such a case, consider using the non-uniformly distributed table instead of the largest one.

Note: After you estimate the maximum amount of data to be inserted and deleted at a database partition, double that estimate to determine the peak size of the active log. If this estimate is greater than the active log limit of 1024 GB, then the data redistribution must be done in steps. For example, use the STEPWISE_REDISTRIBUTE_DBPG procedure with a number of steps proportional to how much the estimate is greater than active log limit. You might also set the **logsecond** database configuration parameter to -1 to avoid most log space problems.

When the redistribution is performed from any supported interface where the data redistribution is not roll-forward recoverable:

- Log records are not created when rows are moved as part of data redistribution. This behavior significantly reduces log file space requirements; however, when this option is used with database roll-forward recovery, the redistribute operation log record cannot be rolled forward, and any tables processed as part of the roll-forward operation remain in UNAVAILABLE state.
- If the database partition group undergoing data redistribution contains tables with long-field (LF) or large-object (LOB) data in the tables, the number of log records generated during data redistribution will be higher, because a log record is created for each row of data. In this case, expect the log space requirement per database partition to be roughly one third of the amount of data moving on that partition (that is, data being sent, received, or both).

Restrictions on data redistribution

Restrictions on data redistribution are important to note before proceeding with data redistribution or when troubleshooting problems related to data redistribution.

The following restrictions apply to data redistribution:

- Data redistribution on partitions where tables do not have partitioning key definitions is restricted.
- When data redistribution is in progress:
  - Starting another redistribution operation on the same database partition group is restricted.
  - Dropping the database partition group is restricted.
  - Altering the database partition group is restricted.
  - Executing an ALTER TABLE statement on any table in the database partition group is restricted.
  - Creating new indexes in the table undergoing data redistribution is restricted.
  - Dropping indexes defined on the table undergoing data redistribution is restricted.
  - Querying data in the table undergoing data redistribution is restricted.
  - Updating the table undergoing data redistribution is restricted.
- Updating tables in a database undergoing a data redistribution that was started using the **REDISTRIBUTE DATABASE PARTITION GROUP** command where the **NOT ROLLFORWARD RECOVERABLE** command parameter was specified is restricted. Although the updates can be made, if data redistribution is interrupted the changes made to the data might be lost and so this practice is strongly discouraged.
- When the **REDISTRIBUTE DATABASE PARTITION GROUP** command is issued and the **NOT ROLLFORWARD RECOVERABLE** command parameter is specified:
  - Data distribution changes that occur during the redistribution are not roll-forward recoverable.
  - If the database is recoverable, the table space is put into the BACKUP PENDING state after accessing the first table within the table space. To remove the table from this state, you must take a backup of the table space changes when the redistribution operation completes.
  - During data redistribution, the data in the tables in the database partition group being redistributed cannot be updated - the data is read-only. Tables that are actively being redistributed are inaccessible.
- For typed (hierarchy) tables, if the **REDISTRIBUTE DATABASE PARTITION GROUP** command is used and the **TABLE** parameter is specified with the value ONLY, then the table name is restricted to being the name of the root table only. Sub-table names cannot be specified.
- Data redistribution is supported for the movement of data between database partitions. For partitioned tables, however, movement of data between ranges of a data partitioned table is restricted unless both of the following are true:
  - The partitioned table has an access mode of FULL ACCESS in the SYSTABLES.ACCESS_MODE catalog table.
  - The partitioned table does not have any partitions currently being attached or detached.

- For replicated materialized query tables, if the data in a database partition group contains replicated materialized query tables, you must drop these tables before you redistribute the data. After data is redistributed, you can recreate the materialized query tables.
- For database partitions that contain multi-dimensional-clustered tables (MDCs) use of the **REDISTRIBUTE DATABASE PARTITION GROUP** command is restricted and will not proceed successfully if there are any multi-dimensional-clustered tables in the database partition group that contain rolled out blocks that are pending cleanup. These MDC tables must be cleaned up before data redistribution can be resumed or restarted.
- Dropping tables that are currently marked in the Db2 catalog views as being in the state "Redistribute in Progress" is restricted. To drop a table in this state, first run the **REDISTRIBUTE DATABASE PARTITION GROUP** command with the **ABORT** or **CONTINUE** parameters and an appropriate table list so that redistribution of the table is either completed or aborted.

Best practices for data redistribution

Data redistribution can be optimally performed when best practices for data redistribution are followed.

Consider the following best practices when planning your data redistribution:

- Ensure that all documented data redistribution prerequisites have been met.

  By default, redistribution operations that are not roll-forward recoverable perform a *precheck* and proceed only if the verification completes successfully. To verify the prerequisites without launching the redistribution operation, specify the **PRECHECK ONLY** command parameter.
- Gather information and metrics about your database environment.

  If performance changes after the redistribution, you can use the information and metrics to identify the reason for the change.
- Back up the database before you perform the data redistribution.

  This is especially important if the redistribution operation is not roll-forward recoverable; if a catastrophic failure occurs during the redistribution and the database is lost or a table is corrupted, you can restore the database from this backup.
- Perform the redistribution during a planned outage, if possible. The instance does not need to be stopped; quiescing the database is sufficient.

  By default, redistribution operations that are not roll-forward recoverable force all users off the database and put the database into a quiesced mode. You must still ensure that any applications or users that are allowed to access the quiesced database are not making updates, for the following reasons:
  - If a disaster occurs, recovering data changes that occurred during the redistribution is complex and, in some cases, not possible.
  - Redistribution typically uses a lot of resources on the servers. Parallel querying of data might cause both the redistribution and the queries to slow down significantly. Redistributing the data online can also cause lock timeouts or deadlocks.
- Perform a redistribution operation that is not roll-forward recoverable.

  Data is moved in bulk instead of by internal insert and delete operations. This reduces the number of times that a table must be scanned and accessed, which results in better performance.

Log records are not required for each of the insert and delete operations. This means that you do not need to manage large amounts of active log space and log archiving space in your system when performing data redistribution.

- A uniform distribution of data might not always result in the best database performance. If a uniform distribution is not desired, then you can construct your own target partition map for the redistribution operation.

  In general if you redistribute data in a frequently accessed table such that infrequently accessed data is on few database partitions in the database partition group, and the frequently accessed data is distributed over a larger number of database partitions, you can improve data access performance and throughput for the most frequently run applications that access this data.

Data redistribution mechanism

Data redistribution can be performed by using different methods in different interfaces however internally the mechanism by which the data is moved is the same. It can be helpful to understand this mechanism so that you are aware of automatic changes being made within the Db2 database environment.

Data redistribution involves the use of the available source distribution map and target distribution map to identify hash database partitions that have been assigned to a new location. The new location is identified by a new database partition number. All rows that correspond to a database partition that have a new location are moved from the database partition specified in the source distribution map to the database partition specified in the target distribution map.

Data redistribution internally invokes a utility that performs the following ordered actions:

1. Obtains a new distribution map ID for the target distribution map, and inserts it into the SYSCAT.PARTITIONMAPS catalog view.
2. Updates the REDISTRIBUTE_PMAP_ID column in the SYSCAT.DBPARTITIONGROUPS catalog view for the database partition group with the new distribution map ID.
3. Adds any new database partitions to the SYSCAT.DBPARTITIONGROUPDEF catalog view.
4. Sets the IN_USE column in the SYSCAT.DBPARTITIONGROUPDEF catalog view to 'D' for any database partition that is to be dropped, if the **DROP DBPARTITIONNUM** command parameter was specified.
5. Commits all catalog updates.
6. Creates database files for all new database partitions if the **ADD DBPARTITIONNUM** command parameter is specified; also might create table spaces in the new database partitions.
7. Redistributes the data on a table-by-table basis for every table in the database partition group, in the following steps:
   a. Puts the table spaces into the BACKUP PENDING state, if the utility did not put them into that state already.
   b. Locks the row for the table in the SYSTABLES catalog table.
   c. Invalidates all packages that involve this table. The distribution map ID associated with the table changes because the table rows are redistributed. Because the packages are invalidated, the compiler must obtain the new database partitioning information for the table and generate packages accordingly.
   d. Locks the table in super exclusive mode (with a z-lock).

e. Redistributes data by using bulk data movement operations.
f. If the redistribution operation succeeds, the distribution map ID for the table is updated in SYSCAT.TABLES. The utility issues a COMMIT for the table and continues with the next table in the database partition group. If the operation fails before the table is fully redistributed, the utility fails. Any partially redistributed tables are left in the REDIST_IN_PGRS state and the table is inaccessible until the redistribute operation is either continued or aborted.

Deletes database files and deletes entries in the SYSCAT.DBPARTITIONGROUPDEF catalog view for database partitions that were previously marked to be dropped.

8. Updates the database partition group record in the SYSCAT.DBPARTITIONGROUPS catalog view to set PMAP_ID to the value of REDISTRIBUTE_PMAP_ID and REDISTRIBUTE_PMAP_ID to NULL.
9. Deletes the old distribution map from the SYSCAT.PARTITIONMAPS catalog view.
10. Does a COMMIT for all changes.

When these steps are done data redistribution is complete. For more information about the success or failure status of the data redistribution and each of the individual data redistributions, review the redistribution log file.

Redistributing data across database partitions by using the REDISTRIBUTE DATABASE PARTITION GROUP command

The **REDISTRIBUTE DATABASE PARTITION GROUP** command is the recommended interface for performing data redistribution.

Procedure

To redistribute data across database partitions in a database partition group:

1. Optional: Perform a backup of the database. See the **BACKUP DATABASE** command.

   It is strongly recommended that you create a backup copy of the database before you perform a data redistribution that is not roll-forward recoverable.
2. Connect to the database partition that contains the system catalog tables. See the CONNECT statement.
3. Issue the **REDISTRIBUTE DATABASE PARTITION GROUP** command.

   Note: In previous versions of the Db2 database product, this command used the **NODEGROUP** keyword instead of the **DATABASE PARTITION GROUP** keywords.

   Specify the following arguments:

   database partition group name
   You must specify the database partition group within which data is to be redistributed.

   UNIFORM
   OPTIONAL: Specifies that data is to be evenly distributed. **UNIFORM** is the default when no distribution-type is specified, so if no other distribution type has been specified, it is valid to omit this option.

   USING DISTFILE *distfile-name*
   OPTIONAL: Specifies that a customized distribution is desired and the

file path name of a distribution file that contains data that defines the desired data skew. The contents of this file is used to generate a target distribution map.

USING TARGETMAP *targetmap-name*
OPTIONAL: Specifies that a target data redistribution map is to be used and the name of file that contains the target redistribution map.

For details, see the **REDISTRIBUTE DATABASE PARTITION GROUP** command-line utility information.

4. Allow the command to run uninterrupted. When the command completes, perform the following actions if the data redistribution proceeded successfully:
   - Take a backup of all table spaces in the database partition group that are in the BACKUP PENDING state. Alternatively, a full database backup can be performed.

     Note: Table spaces are only put into the BACKUP PENDING state if the database is recoverable and the **NOT ROLLFORWARD RECOVERABLE** command parameter is used in the **REDISTRIBUTE DATABASE PARTITION GROUP** command.
   - Recreate any replicated materialized query tables dropped before redistribution.
   - Execute the **RUNSTATS** command if the following conditions are met:
     – The **STATISTICS NONE** command parameter was specified in the **REDISTRIBUTE DATABASE PARTITION GROUP** command, or the **NOT ROLLFORWARD RECOVERABLE** command parameter was omitted. Both of these conditions mean that the statistics were not collected during data redistribution.
     – There are tables in the database partition group possessing a statistics profile.

     The **RUNSTATS** command collects data distribution statistics for the SQL compiler and optimizer to use when choosing data access plans for queries.
   - If the **NOT ROLLFORWARD RECOVERABLE** command parameter was specified, delete the control files located in the following paths :
     – On Linux and UNIX operating systems: **diagpath**/redist/*db_name*/*db_partitiongroup_name*/*timestamp*/
     – On Windows operating systems: **diagpath**\redist*db_name**db_partitiongroup_name**timestamp*\

Results

Data redistribution is complete and information about the data redistribution process is available in the redistribution log file. Information about the distribution map that was used can be found in the Db2 explain tables.

Redistributing database partition groups using the STEPWISE_REDISTRIBUTE_DBPG procedure

Data redistribution can be performed using built-in procedures.

Procedure

To redistribute a database partition group using the STEPWISE_REDISTRIBUTE_DBPG procedure:

1. Analyze the database partition group regarding log space availability and data skew using the ANALYZE_LOG_SPACE procedure.

   The ANALYZE_LOG_SPACE procedure returns a result set (an open cursor) of the log space analysis results, containing fields for each of the database partitions of the given database partition group.
2. Create a data distribution file for a given table using the GENERATE_DISTFILE procedure.

   The GENERATE_DISTFILE procedure generates a data distribution file for the given table and saves it using the provided file name.
3. Create and report the content of a stepwise redistribution plan for the database partition group using the STEPWISE_REDISTRIBUTE_DBPG procedure.
4. Create a data distribution file for a given table using the GET_SWRD_SETTINGS and SET_SWRD_SETTINGS procedures.

   The GET_SWRD_SETTINGS procedure reads the existing redistribute registry records for the given database partition group.

   The SET_SWRD_SETTINGS procedure creates or makes changes to the redistribute registry. If the registry does not exist, it creates it and add records into it. If the registry already exists, it uses *overwriteSpec* to identify which of the field values need to be overwritten. The *overwriteSpec* field enables this function to take NULL inputs for the fields that do not need to be updated.
5. Redistribute the database partition group according to the plan using the STEPWISE_REDISTRIBUTE_DBPG procedure.

   The STEPWISE_REDISTRIBUTE_DBPG procedure redistributes part of the database partition group according to the input and the setting file.

Example

The following is an example of a CLP script on AIX®:

```
# ------------------------------------------------------------------------------------
# Set the database you wish to connect to
# ------------------------------------------------------------------------------------
dbName="SAMPLE"

# ------------------------------------------------------------------------------------
# Set the target database partition group name
# ------------------------------------------------------------------------------------
dbpgName="IBMDEFAULTGROUP"

# ------------------------------------------------------------------------------------
# Specify the table name and schema
# ------------------------------------------------------------------------------------
tbSchema="$USER"
tbName="STAFF"

# ------------------------------------------------------------------------------------
# Specify the name of the data distribution file
# ------------------------------------------------------------------------------------
distFile="$HOME/sqllib/function/$dbName.IBMDEFAULTGROUP_swrdData.dst"

export DB2INSTANCE=$USER
export DB2COMM=TCPIP

# ------------------------------------------------------------------------------------
# Invoke call statements in clp
# ------------------------------------------------------------------------------------
db2start
db2 -v "connect to $dbName"

# ------------------------------------------------------------------------------------
```

```
# Analysing the effect of adding a database partition without applying the changes - a 'what if'
# hypothetical analysis
#
# - In the following case, the hypothesis is adding database partition 40, 50 and 60 to the
#   database partition group, and for database partitions 10,20,30,40,50,60, using a respective
#   target ratio of 1:2:1:2:1:2.
#
# NOTE: in this example only partitions 10, 20 and 30 actually exist in the database
#       partition group
# ------------------------------------------------------------------------------------
db2 -v "call sysproc.analyze_log_space('$dbpgName', '$tbSchema', '$tbName', 2, ' ',
'A', '40,50,60', '10,20,30,40,50,60', '1,2,1,2,1,2')"

# ------------------------------------------------------------------------------------
# Analysing the effect of dropping a database partition without applying the changes
#
# - In the following case, the hypothesis is dropping database partition 30 from the database
#   partition group, and redistributing the data in database partitions 10 and 20 using a
#   respective target ratio of 1 : 1
#
# NOTE: In this example all database partitions 10, 20 and 30 should exist in the database
#       partition group
# ------------------------------------------------------------------------------------
db2 -v "call sysproc.analyze_log_space('$dbpgName', '$tbSchema', '$tbName', 2, ' ',
'D', '30', '10,20','1,1')"

# ------------------------------------------------------------------------------------
# Generate a data distribution file to be used by the redistribute process
# ------------------------------------------------------------------------------------
db2 -v "call sysproc.generate_distfile('$tbSchema', '$tbName', '$distFile')"

# ------------------------------------------------------------------------------------
# Write a step wise redistribution plan into a registry
#
# Setting the 10th parameter to 1, may cause a currently running step wise redistribute
# stored procedure to complete the current step and stop, until this parameter is reset
# to 0, and the redistribute stored procedure is called again.
# ------------------------------------------------------------------------------------
db2 -v "call sysproc.set_swrd_settings('$dbpgName', 255, 0, ' ', '$distFile', 1000,
12, 2, 1, 0, '10,20,30', '50,50,50')"

# ------------------------------------------------------------------------------------
# Report the content of the step wise redistribution plan for the given database
# partition group.
# ------------------------------------------------------------------------------------
db2 -v "call sysproc.get_swrd_settings('$dbpgName', 255, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)"

# ------------------------------------------------------------------------------------
# Redistribute the database partition group "dbpgName" according to the redistribution
# plan stored in the registry by set_swrd_settings. It starting with step 3 and
# redistributes the data until 2 steps in the redistribution plan are completed.
# ------------------------------------------------------------------------------------
db2 -v "call sysproc.stepwise_redistribute_dbpg('$dbpgName', 3, 2)"
```

Monitoring a data redistribution operation

You can use the **LIST UTILITIES** command to monitor the progress of data redistribution operations on a database.

Procedure

Issue the **LIST UTILITIES** command and specify the **SHOW DETAIL** parameter:

```
list utilities show detail
```

Results

The following is an example of the output from this command:

```
ID                              = 1
Type                            = REDISTRIBUTE
Database Name                   = RDST819
Partition Number                = 11
Description                     = RDST_V10_015 UNIFORM ADD NODES
                                  COMPACT ON SPACE REUSE RECORD LEVEL
                                  INDEXING MODE INCREMENTAL
Start Time                      = 02-20-2007 23:21:33.785819
State                           = Executing
Invocation Type                 = User
Progress Monitoring:
   Estimated Percentage Complete = 8
   Summary:
      Total Work                = 1965600
      Completed Work            = 155221
      Total Number Of Tables    = 15
      Tables Completed          = 0
      Tables In Progress        = 3

   Current Table 1:
      Description               = "NEWTON  "."RDST_V10_015A"
      Total Work                = 655200 bytes
      Completed Work            = 55001 bytes

   Current Table 2:
      Description               = "NEWTON  "."RDST_V10_015B"
      Total Work                = 450200 bytes
      Completed Work            = 54220 bytes

   Current Table 3:
      Description               = "NEWTON  "."RDST_V10_015C"
      Total Work                = 978901 bytes
      Completed Work            = 46000 bytes
```

Redistribution event log files

During data redistribution event logging is performed. Event information is logged to event log files which can later be used to perform error recovery.

When data redistribution is performed, information about each table that is processed is logged in a pair of event log files. The event log files are named *database-name.database-partition-group-name.timestamp*.log and *database-name.database-partition-group-name.timestamp*.

The log files are located as follows:

- The *homeinst*/sqllib/redist directory on Linux and UNIX operating systems
- The *db2instprof*\instance\redist directory on Windows operating systems, where *db2instprof* is the value of the **DB2INSTPROF** registry variable

The following is an example of the event log file names:

```
SAMPLE.IBMDEFAULTGROUP.2012012620240204
SAMPLE.IBMDEFAULTGROUP.2012012620240204.log
```

These files are for a redistribution operation on a database named SAMPLE with a database partition group named IBMDEFAULTGROUP. The files were created on January 26, 2012 at 8:24 PM local time.

The three main uses of the event log files are as follows:

- To provide general information about the redistribute operation, such as the old and new distribution maps.
- Provide users with information that helps them determine which tables have been redistributed so far by the utility.
- To provide information about each table that has been redistributed, including the indexing mode being used for the table, an indication of whether the table was successfully redistributed or not, and the starting and ending times for the redistribution operation on the table.

Recovery from errors related to data redistribution

Recovery from failures and errors that occur during data redistribution generally requires that you consult the redistribution event log file. The log file contains useful information about data redistribution processing, including information about which table or tables, if any, failed to be processed successfully.

Possible reasons that a redistribution might fail include:

- A documented prerequisite for successful data redistribution was not met.
- A documented restriction on data redistribution was encountered that resulted in the interruption of the data redistribution.
- During data redistribution a table to be processed was encountered in a restricted access state, such as LOAD PENDING.
- Any other problem documented in the event log.

When you have identified and resolved the cause of the failure, perform the recovery by using the same redistribution interfaces that were used when the operation failed. For example, if the **REDISTRIBUTE DATABASE PARTITION GROUP** command was used, once the problem has been addressed you can begin data redistribution again by reissuing the command with one of the following options:

CONTINUE
: This option indicates that the redistribution operation should continue until all tables specified in the original **REDISTRIBUTE DATABASE PARTITION GROUP** command are redistributed.

ABORT
: This option indicates that the redistribution operation should be aborted and that all tables that have been redistributed, or partially redistributed, so far should be returned to the state they were in before the **REDISTRIBUTE DATABASE PARTITION GROUP** command was first run on the database partition group.

These options cannot be specified unless a previous data redistribution operation failed or completed without redistributing all tables in the database partition group. The latter case can occur if the **TABLE** command parameter is used and only a subset of tables is specified. In these cases, the value of the REDISTRIBUTE_PMAP_ID column in the SYSCAT.DBPARTITIONGROUPS table will have a value different from -1.

If an interface other than the **REDISTRIBUTE DATABASE PARTITION GROUP** command was used, continuation or abortion of data redistribution is possible by redistributing data again using the appropriate parameter value for the specified interface. See the reference information for the interface for the correct parameter values.

Examples of redistribute event log file entries

Examples of common redistribute event log files entries and descriptions of them provide a useful reference that can be consulted when troubleshooting errors or interruptions that occur during data redistribution.

There are two event log files created each time you redistribute a database partition group. The file named *database-name.database-partition-group-name.timestamp* provides a brief summary of the event. For example, SAMPLE.IBMDEFAULTGROUP.2012012622083174 contains:

```
  Data Redistribution Utility :
  ____________________________

The following options have been specified :
Database partition group name : IBMDEFAULTGROUP
Data Redistribution option : U
Redistribute database partition group : uniformly
No. of partitions to be added : 0
List of partitions to be added :
No. of partitions to be dropped : 1
List of partitions to be dropped :
2
The execution of the Data Redistribution operation on :

 Begun at  Ended at  Table (poolID;objectID)
 ________  ________  ______________________
 22.08.32            "DB2INST1"."CL_SCHED" (2;4)
           22.08.33  "DB2INST1"."CL_SCHED" (2;4)
 22.08.33            "DB2INST1"."DEPARTMENT" (2;5)
           22.08.35  "DB2INST1"."DEPARTMENT" (2;5)
 22.08.35            "DB2INST1"."EMPLOYEE" (2;6)
           22.08.36  "DB2INST1"."EMPLOYEE" (2;6)
 ...
 22.09.13            "DB2INST1"."PRODUCTSUPPLIER" (4;10)
           22.09.15  "DB2INST1"."PRODUCTSUPPLIER" (4;10)

--All tables have been successfully redistributed.--
```

The file named *database-name.database-partition-group-name.timestamp*.log provides more detailed log entries. The following examples illustrate some common log file entries. Although each field value in the redistribute log file entries is not defined, the examples illustrate the main fields and most common or most important field values.

Example 1: First event log entry dumped during a redistribute operation

```
2012-01-26-22.08.32.607340-300 I1850E893            LEVEL: Event
PID     : 27700                TID : 46912984049984  PROC : db2sysc 0
INSTANCE: db2inst1             NODE : 000            DB   : SAMPLE
APPHDL  : 0-74                 APPID: *N0.db2inst1...
AUTHID  : DB2INST1             HOSTNAME: ...
EDUID   : 65                   EDUNAME: db2agent (SAMPLE) 0
FUNCTION: Db2, relation data serv, sqlrdrin, probe:3852
CHANGE  : DB PART MAP ID : IBMDEFAULTGROUP : FROM "1" : TO "3" : success
IMPACT  : None
DATA #1 : String, 24 bytes
HexDump Old Map Entries:
DATA #2 : Dumped object of size 65536 bytes at offset 0, 61 bytes
/home/db2inst1/sqllib/redist/27700.65.000.dump.bin
DATA #3 : String, 24 bytes
HexDump New Map Entries:
DATA #4 : Dumped object of size 65536 bytes at offset 65672, 61 bytes
/home/db2inst1/sqllib/redist/27700.65.000.dump.bin
```

The first event log entry dumped during a redistribute operation provides information about the distribution map files with which the utility will be working. In this case, the old distribution map for partition group IBMDEFAULTGROUP has an id of 1 and the new distribution map has an id of 3. A hexdump of each distribution map file is made to the `redist` directory in the instance path, and the names of the dump files are included in the entry. In this example, the file named `27700.65.000.dump.bin` contains both distribution maps.

Example 2: Event log associated with the start of the redistribute operation

```
2012-01-26-22.08.32.625956-300 I2744E774             LEVEL: Event
PID     : 27700                TID : 46912984049984  PROC : db2sysc 0
INSTANCE: db2inst1             NODE : 000            DB   : SAMPLE
APPHDL  : 0-74                 APPID: *N0.db2inst1...
AUTHID  : DB2INST1             HOSTNAME: ...
EDUID   : 65                   EDUNAME: db2agent (SAMPLE) 0
FUNCTION: Db2, relation data serv, sqlrdrInitLogfileInfo, probe:1802
START   : REDIST DB PART GROUP : IBMDEFAULTGROUP : success
IMPACT  : None
DATA #1 : String, 28 bytes
Partitioning Option: UNIFORM
DATA #2 : String, 23 bytes
Statistics: USE PROFILE
DATA #3 : String, 22 bytes
Indexing Mode: REBUILD
DATA #4 : String, 17 bytes
 PRECHECK MODE: Y
DATA #5 : String, 16 bytes
 QUIESCE MODE: Y
```

This entry indicates that the start of the redistribute operation has completed successfully and that redistribution of tables is about to begin. The partitioning option, statistics option, indexing mode, precheck mode, and quiesce mode to be used for the redistribution operation are also shown.

In this example, the partitioning option is `UNIFORM`, which indicates that data will be redistributed uniformly. Other possible values for this option include `TARGETMAP`, `CONTINUE` and `ABORT`.

The statistics option is `USE PROFILE`, which is the default statistics collection mode and means that if the table has a statistics profile, then statistics will be collected according to that profile. Otherwise, statistics will not be collected. If the value of this option is `NONE`, this indicates that the **STATISTICS NONE** option was specified, which means that no statistics are to be collected for the table, regardless of whether the table has a statistics profile defined or not.

In this example, the indexing mode is `REBUILD`, which is the default indexing mode and means that indexes on each table will be rebuilt during data redistribution. If the value of this option is `DEFERRED`, it means that the user specified the **INDEXING MODE DEFERRED** option, which results in indexes being marked invalid during redistribution. Such indexes must be rebuilt after data redistribution is complete.

In this example, the precheck mode is `Y`, which is the default precheck mode for data redistributions that are **NOT ROLLFORWARD RECOVERABLE**. This mode indicates that the redistribution operation begins only if the verification completes successfully.

In this example, the quiesce mode is `Y`, which is the default quiesce mode for data redistributions that are **NOT ROLLFORWARD RECOVERABLE**. This mode indicates that the

redistribution operation forces all users off the database and puts it into a quiesced mode.

Example 3: Event log associated with start of redistributing a table

```
2012-01-26-22.08.32.843840-300 I4072E541            LEVEL: Event
PID     : 27700                TID : 46912874998080  PROC : db2sysc 0
INSTANCE: db2inst1             NODE : 000            DB   : SAMPLE
APPHDL  : 0-113                APPID: *N0.DB2....
AUTHID  : DB2INST1             HOSTNAME: ...
EDUID   : 181                  EDUNAME: db2agent (SAMPLE) 0
FUNCTION: Db2, database utilities - Redistribute, sqlurRedistributeTableByID
, probe:8743
START   : REDIST TABLE : "DB2INST1"."CL_SCHED" : success
IMPACT  : None
```

This entry indicates that the start of redistributing table "DB2INST1"."CL_SCHED" was successful.

Example 4: Event log associated with successful completion of redistribution for a table

```
2012-01-26-22.08.33.659785-300 I4614E541            LEVEL: Event
PID     : 27700                TID : 46912874998080  PROC : db2sysc 0
INSTANCE: db2inst1             NODE : 000            DB   : SAMPLE
APPHDL  : 0-113                APPID: *N0.DB2....
AUTHID  : DB2INST1             HOSTNAME: ...
EDUID   : 181                  EDUNAME: db2agent (SAMPLE) 0
FUNCTION: Db2, database utilities - Redistribute, sqlurRedistributeTableByID
, probe:9350
STOP    : REDIST TABLE : "DB2INST1"."CL_SCHED" : success
IMPACT  : None
```

This entry indicates that table "DB2INST1"."CL_SCHED" has been successfully redistributed. If an error had occurred during processing of this table, this entry would indicate failure.

Example 5: Event log associated with successful completion of redistribution of a database partition group

```
2012-01-26-22.09.16.746325-300 I28994E515            LEVEL: Event
PID     : 27700                TID : 46912984049984  PROC : db2sysc 0
INSTANCE: db2inst1             NODE : 000            DB   : SAMPLE
APPHDL  : 0-74                 APPID: *N0.db2inst1....
AUTHID  : DB2INST1             HOSTNAME: ...
EDUID   : 65                   EDUNAME: db2agent (SAMPLE) 0
FUNCTION: Db2, relation data serv, sqlrdrdt, probe:1308
STOP    : REDIST DB PART GROUP : IBMDEFAULTGROUP : success
IMPACT  : None
```

This entry indicates that redistribution of database partition group IBMDEFAULTGROUP completed successfully. If the operation had not completed successfully, this entry would indicate failure.

These example entries can be a helpful reference when you consult your log files to resolve errors that occur during data redistribution or to verify that data redistribution complete successfully.

Scenario: Redistributing data in new database partitions

This scenario shows how to add new database partitions to a database and redistribute data between the database partitions. The **REDISTRIBUTE DATABASE PARTITION GROUP** command is demonstrated as part of showing how to redistribute data on different table sets within a database partition group.

About this task

Scenario:

A database DBPG1 has two database partitions, specified as (0, 1) and a database partition group definition (0, 1).

The following table spaces are defined on database partition group DBPG_1:

- Table space TS1 - this table space has two tables, T1 and T2
- Table space TS2 - this table space has three tables defined, T3, T4, and T5

Starting in Version 9.7, you can add database partitions while the database is running and while applications are connected to it. However, the operation can be performed offline in this scenario by changing the default value of the **DB2_FORCE_OFFLINE_ADD_PARTITION** registry variable to TRUE.

Procedure

To redistribute data between the database partitions in DBPG1:

1. Identify objects that must be disabled or removed before the redistribution.
   a. Replicate MQTs: This type of MQT is not supported as part of the redistribution operation. They must be dropped before running the redistribution and recreated afterward.

   ```
   SELECT tabschema, 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 90 and 8 on page 90. 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;
```

Index

M

P

Q

R

S

T

U

W

Printed in USA